Fall 2010

Efficient modular arithmetic units for low power cryptographic applications

Rajashekhar Reddy Modugu

Department:

EFFICIENT MODULAR ARITHMETIC UNITS FOR LOW POWER CRYPTOGRAPHIC APPLICATIONS

by

RAJASHEKHAR REDDY MODUGU

A THESIS

Presented to the Faculty of the Graduate School of the

MISSOURI UNIVERSITY OF SCIENCE AND TECHNOLOGY

In Partial Fulfillment of the Requirements for the Degree

MASTER OF SCIENCE IN COMPUTER ENGINEERING

2010

Approved by:

Dr. Minsu Choi, Advisor
Dr. Yiyu Shi
Dr. Sahra Sedighsarvestani

PUBLICATION THESIS OPTION

This thesis consists of following articles that have been published as follows, and the papers were formatted in the style used by the university.

The first paper presented in pages 02-15 entitled "A FAST LOW-POWER MODULO 2N+1 MULTIPLIER DESIGN" was published in the proceedings of the IEEE IMTC(Instrumentation and Measurement Technology Conference), 2009.

The second paper presented in pages 17-28 entitled "AN EFFICIENT IDEA CRYPTO-HARDWARE USING NOVEL MODULAR ARITHMETIC COMPONENTS" was published in the proceedings of the IEEE IMTC(Instrumentation and Measurement Technology Conference), 2010.

The third paper presented in pages 31-55 entitled "Efficient On-line Self-Checking Modulo 2n+1 Multiplier Design" was accepted for Special Issue of IEEE Transactions on Instrumentation and Measurement, 2010.

ABSTRACT

The demand for high security in energy constrained devices such as mobiles and PDAs is growing rapidly. This leads to the need for efficient design of cryptographic algorithms which offer data integrity, authentication, non-repudiation and confidentiality of the encrypted data and communication channels. The public key cryptography is an ideal choice for data integrity, authentication and non-repudiation whereas the private key cryptography ensures the confidentiality of the data transmitted. The latter has an extremely high encryption speed but it has certain limitations which make it unsuitable for use in certain applications. Numerous public key cryptographic algorithms are available in the literature which comprise modular arithmetic modules such as modular addition, multiplication, inversion and exponentiation. Recently, numerous cryptographic algorithms have been proposed based on modular arithmetic which are scalable, do word based operations and efficient in various aspects.

The modular arithmetic modules play a crucial role in the overall performance of the cryptographic processor. Hence, better results can be obtained by designing efficient arithmetic modules such as modular addition, multiplication, exponentiation and squaring.

This thesis is organized into three papers, describes the efficient implementation of modular arithmetic units, application of these modules in International Data Encryption Algorithm (IDEA). Second paper describes the IDEA algorithm implementation using the existing techniques and using the proposed efficient modular units. The third paper describes the fault tolerant design of a modular unit which has online self-checking capability.

ACKNOWLEDGMENTS

It gives me great pleasure to thank all the people who have supported me and made this thesis possible. I would like to thank Dr. Minsu Choi, who has been a great advisor throughout my Master's program and it has been a pleasure working with him. He has been a continuous source of motivation and has helped me develop my skills.

I would like to express my sincere gratitude to Dr. Yiyu Shi and Dr. Sahra Sedighsarvestani who are serving as committee members. I would like to thank Dr. Beetner, Dr. McCracken, Dr. Al-Assadi and Dr. Ali Hurson who have taught me excellent courses during my Master's program. I am grateful to the department secretary, Mrs. Regina Kohout who has guided me through departmental obstacles and paperwork. I give sincere thanks to Jun Wu for associating with me in research and publications.

I would like to also thank my friends Vikram Surendra, Murali Bottu, Dr. Bharath, Dr. Suhas, Arun Sharma, Komal and many more friends who have supported me continuously throughout my Degree program and provided me a refreshing environment. Most importantly, I would like to thank my parents, Krishna Reddy, Dhana Lakshmi, my brother, Sathish Reddy whose continuous support made this degree possible.

TABLE OF CONTENTS

Page

LIST OF ILLUSTRATIONS

Figure **Page**

PAPER I

PAPER II

PAPER III

LIST OF TABLES

1. INTRODUCTION

Cryptography has become an integral part of most of the security applications and low-power embedded applications. The ability to secure and the performance of the cryptographic algorithm are the major factors that decide the overall efficiency of the system. Very secured cryptographic algorithms have been designed and it is really difficult to crack these algorithms. The performance of these algorithms can be improved by designing very efficient hardware models. This thesis focuses on the novel hardware implementation of a cryptographic algorithm. Experiments are performed to check the overall performance. The fault tolerant design of this multiplier is analyzed and implemented using residue codes.

This thesis is comprised of three research publications. The first paper describes the efficient implementation of modulo 2^n+1 multiplier which is the basic building block of International Data Encryption Algorithm (IDEA). For the efficient implementation of modulo multiplier, efficient compressors and a newly designed sparse tree adder is used.

The second paper describes the design of hardware implementation of the IDEA cipher using novel modulo 2^n+1 multipliers. It shown that the proposed modulo 2^n+1 multiplier improves the performance of the various cryptographic algorithms used in secure communication systems of networked instrumentation and distributed measurement systems. Efficient compressors and sparse tree based inverted end around carry adders are used to reduce the delay and complexity of the multiplier. Simulations are performed on the known implementation and the proposed implementation.

The third paper describes the online self-checking model of the modulo 2^n+1 multiplier based on residue codes is presented. 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 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 different values of the input length and simulated to get the experimental results.

PAPER

I. A FAST LOW-POWER MODULO 2^N+1 MULTIPLIER DESIGN

Rajashekhar Modugu[1] and Minsu Choi[1]

Nohpill Park[2]

[1]Department of Electrical & Computer Engineering

Missouri University of Science and Technology

Rolla, MO 65409, USA

{rrmt4b, choim}@mst.edu

[2]Department of Computer Science

Oklahoma State University

Stillwater, OK 74078-1053, USA

npark@cs.okstate.edu

Abstract— Modulo 2^n+1 multipliers are the basic building blocks in many security applications such as International Data Encryption Algorithm in cryptography. In this paper, a fast low-power hardware implementation of modulo 2^n+1 multiplier is proposed. Novel hardware implementations for a previously proposed algorithm are shown by using the efficient compressors and modulo carry look-ahead adders as the basic building blocks. The modulo carry lookahead adder uses the sparse-tree adder technique. The resulting implementations are compared both qualitatively and quantitatively, in standard CMOS cell technology, with the existing implementations. The results show that the proposed implementation is faster and consume less power than similar hardware implementations making them a viable option for efficient designs.

Index Terms— Compressors; International Data Encryption Algorithm (IDEA); modulo multiplier; sparse-tree adder;

1. INTRODUCTION

In the recent years, the number of 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. Various cryptographic systems have been studied and implemented to encode and to decode the data. International Data Encryption Algorithm (IDEA) [1] is one of the most reliable cryptographic algorithms used for transmission of the data. The ability to perform fast encoding and decoding operations is then still a major issue for the implementation of IDEA, particularly from a hardware point of view.

The three 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. As the first two operations take less time and are easy to implement, improving the delay and power efficiency of the modulo 2^n+1 multiplication operation leads to significant increase in the performance of the entire IDEA cipher. Although numerous algorithms [2-5] of the modulo 2^n+1multiplier were proposed, hardware implementation of the same needs considerable effort. More recently, Vergos and Efstathiou [5] proposed an efficient algorithm for computing modulo 2^n+1 multiplication, in which the partial product reduction block, which contributes most to the overall delay, is designed using full adders. Hence, the demand for efficient implementation of the partial product reduction block is continuously increasing.

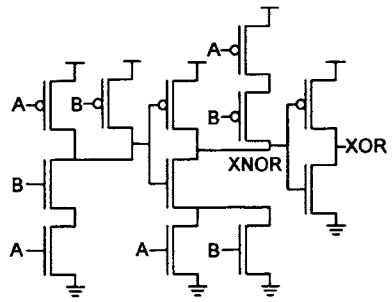
In this paper, a new hardware implementation of the partial product reduction module of the mod 2^n+1 multiplier is proposed. And also, the final stage addition module is redesigned using more efficient carry lookahead adder technique. The resulting hardware implementation is faster and consumes less power than existing ones.

The paper is organized as follows; Section II introduces multiplexor-based compressors. In Section III, the algorithm for computation of modulo 2^n+1 multiplication is given. Section IV discusses the proposed implementation of the multiplier, with implementation example for the input word length, n=16, which is used in IDEA cipher. A comparison of this implementation to a recently proposed implementation is made in Section V. Conclusions are drawn in Section VI.

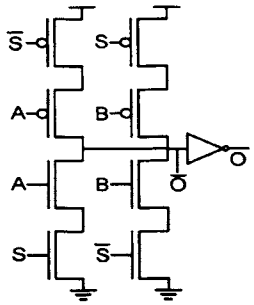
2. COMPRESSORS

2.1. MUX VS. XOR

Existing CMOS designs of 2-1 MUX and 2-input XOR are shown in Fig.1. According to [6], the CMOS implementation of MUX, it performs better in terms of power and delay compared to 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. The efficient implementation of compressors [7] is achieved by using both output and its complement of these gates. This also reduces the total number of garbage outputs.



(a)



(b)

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

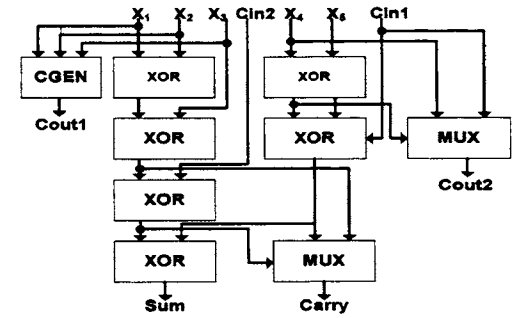
2.2. DESCRIPTION OF COMPRESSORS

A $(p,2)$ compressor with p inputs $X1, X2..Xp$ and two output bits Sum and Carry along with carry input bits and carry output bits is governed by the equation:

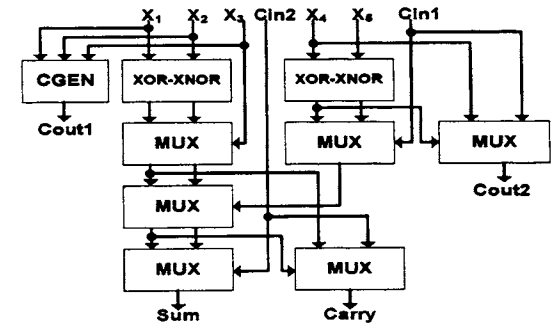
$$\sum_{i=1}^{p} X_i + \sum_{i=1}^{t} (C_{in})_i = Sum + 2(Carry + \sum_{i=1}^{t} (C_{out})_i)$$

Efficient design of the existing XOR-based 5:2 compressor [8-9], which takes 5 inputs and 2 carry inputs, is shown in Fig. 2a. The critical path delay of this compressor is $4\Delta - XOR$ (delay denoted by Δ).

The newly designed compressors use multiplexers in place of XOR gates, resulting in high speed arithmetic. 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 the CMOS implementation of the MUX if both the select bit and its complement are generated in the previous stage then its output is 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 is eliminated which reduces the overall delay in the critical path. The new MUX-based design of 5:2 compressor [7] is shown in Fig. 2b, the delay of which is $\Delta - XOR + 3\Delta - MUX$. CGEN block used in Fig. 2 can be obtained from the equation $\mathrm{Cout1} = (\mathrm{x1} + \mathrm{x2}) \cdot \mathrm{x3} + \mathrm{x1} \cdot \mathrm{x2}$.



(a)



(b)

Fig. 2 5:2 compressors; (a) Existing design (b) New design

3. ALGORITHM FOR IMPLEMENTATION OF MODULO 2^N+1 MULTIPLIER

The algorithm for computation of $X \cdot Y \bmod 2^n + 1$ is described below. From the architectural characteristic comparisons [5], the algorithm presented in [5] is considered as the best existing algorithm for the computation of $X \cdot Y \bmod 2^n + 1$ in the literature. Hence, this algorithm is used for the proposed implementation of the modulo multiplier. According to the algorithm it takes two n+1 bit unsigned numbers as inputs and gives one n+1 bit output. The proposed implementation can be adapted to IDEA cipher [1-2], in which the $mod\ 2^n + 1$ multiplication module takes two n-bit inputs and gives one n-bit output, by assigning the most significant bits of the inputs zeros and neglecting the most significant bit of the output.

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 '1' only when the inputs are 2^n and 2^n, respectively. $X \cdot Y \bmod 2^n + 1$ can be represented as follows:

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

The n × n partial product matrix is derived from the initial partial product matrix in Fig. 3, based on several observations. This n × n partial product matrix is shown in Fig. 5. First observation is, the initial partial product matrix can be divided into four groups A, B, C and D in which the terms in only one group can be different from '0'. Groups A, B, D and C are different from '0', if inputs (X, Y) are in the form of (0Z, 0Z), (1Z, 0Z), (0Z, 1Z) and (10..0,10..0), respectively (here 'Z' is a 16-bit vector). Hence the four groups can be integrated into a single group by performing logical OR operation (denoted by v) instead of adding the bits arithmetically. Logical OR operation is performed on the terms of the groups B, D and A in the columns with weight 2^n up to 2^{2n-2} and on the two terms of the groups B and D with weight 2^{2n-1} (the ORed terms of the groups B and D are represented by qi, where $q_i = p_{n,i} V p_{i,n}$). Since $|2^{2n-1}|_{2^n+1} = 2^{n-1} + 1$, the term with weight$2^{2n-1}$, q_{n-1}, can be substituted by two terms q_{n-1} in the columns with weight 2^{n-1} and 1, respectively, and ORed with any term of the group A

there. Moreover, since $|2^{2n}|_{2^n+1} = 1$, the term $p_{n,n}$can be ORed with $p_{0,0}$. The modified partial product matrix is shown in Fig. 4.

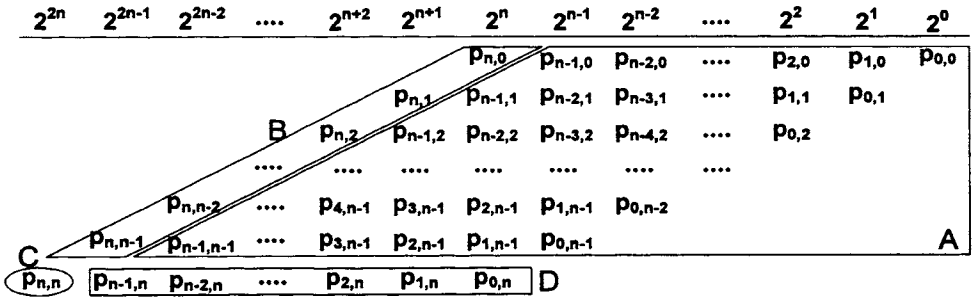


Fig. 3 Initial partial product matrix

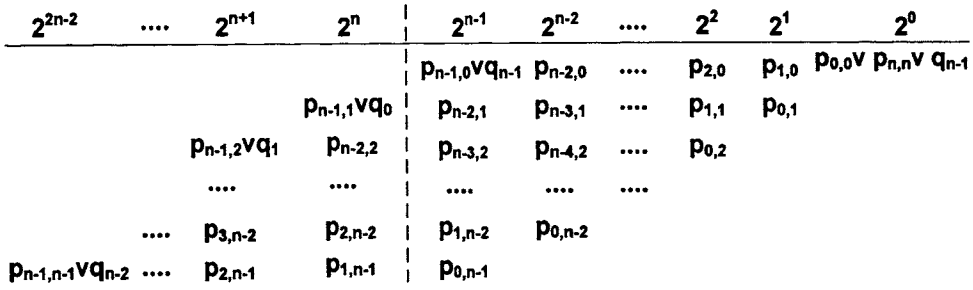


Fig. 4 Modified partial product matrix

Second observation is repositioning of the partial product terms in the modified partial product matrix, with weight greater than 2^{n-1} based on the following equation:

$$|s2^i|_{2^n+1} = |-s2^{|i|_n}|_{2^n+1} = |(2^n + 1 - s)2^{|i|_n}|_{2^n+1}$$
$$= |\bar{s}2^{|i|_n} + 2^n 2^{|i|_n}|_{2^n+1} \quad (1)$$

Equation (1) shows that the repositioning of each bit results in a correction factor of $2^n 2^{|i|_n}$. In the first partial product vector, there is only one such bit and in the second

partial product vector 2 bits need to be repositioned and so on. Hence the correction factor for the entire partial product matrix would be:

$$COR1 = 2^n\left(2(1 + 2 + 2^2 + \cdots\cdots + 2^{n-2}) - (n-1)\right)$$
$$= 2^n(2^n - n - 1) \quad (2)$$

The n × n partial product matrix along with the equation (2) results in n+1 operands. These partial product terms can be reduced into two final summands Sum array and Carry array using a Carry Save Adder (CSA) tree. Suppose the carry out bit at i^{th} stage of CSA is c_i with weight 2^n, this carryout can be reduced into:

$$|c_i 2^n|_{2^n+1} = |-c_i|_{2^n+1} = |2^n + \bar{c_i}|_{2^n+1}$$

Therefore the carry output bits at the most significant bit position of each stage can be used as carry input bits of the next stage. In an n-1 stage CSA in [5] produces n-1 such carry out bits. Hence there will be a second correction factor (3). And the overall correction factor using this algorithm is:

$$COR2 = \left|2^n(n-1)\right|_{2^n+1} \quad (3)$$

The final correction factor will be the sum of $COR1$ and $COR2$. The constant '3' in equation (4) will be the final partial product.

$$COR = COR1 + COR2 = |2^n(n-1) + 2^n(2^n - n - 1)|_{2^n+1}$$
$$= |2^n(2^n - 2)|_{2^n+1} = 3 \quad (4)$$

2^{n-1}	2^{n-2}	2^{n-3}	2^2	2^1	2^0
$PP_0 = p_{n-1,0} \vee q_{n-1}$	$p_{n-2,0}$	$p_{n-3,0}$	$p_{2,0}$	$p_{1,0}$	$p_{0,0} \vee q_{n-1} \vee p_{n,n}$
$PP_1 = p_{n-2,1}$	$p_{n-3,1}$	$p_{n-4,1}$	$p_{1,1}$	$p_{0,1}$	$p_{n-1,1} \vee q_0$
$PP_2 = p_{n-3,1}$	$p_{n-4,2}$	$p_{n-5,2}$	$p_{0,2}$	$p_{n-1,2} \vee q_1$	$p_{n-2,2}$
....	....	....	....	....	....
$PP_{n-2} = p_{1,n-2}$	$p_{0,n-2}$	$p_{n-1,n-2} \vee q_{n-3}$	$p_{4,n-2}$	$p_{3,n-2}$	$p_{2,n-2}$
$PP_{n-1} = p_{0,n-1}$	$p_{n-1,n-1} \vee q_{n-2}$	$p_{n-2,n-1}$	$p_{3,n-1}$	$p_{2,n-1}$	$p_{1,n-1}$

Fig. 5 Final n × n partial product matrix

4. PROPOSED IMPLEMENTATION OF THE MOD 2^N+1 MULTIPLIER

The proposed implementation of the modulo multiplier consists of three modules. First module is to generate partial products, second module is to reduce the partial products to two final operands and the last module is to add the Sum and Carry operands from partial products reduction to get the final result.

4.1. PARTIAL PRODUCTS GENERATION

From the above $n \times n$ partial product matrix (shown in Fig. 5), it is possible to observe that the partial product generation requires AND, OR and NOT gates. The most complex function of partial product generation module is $\overline{p_{n-1,n-1} V\, q_{n-2}}$, where $p_{i,j} = a_i b_j$ *and* $q_i = p_{n,i} V p_{i,n}$.

4.2. PARTIAL PRODUCTS REDUCTION

This is the most important module which largely determines the critical path delay and the overall performance of the multiplier. Hence this module needs to be designed so as to get minimum delay and consume less power.

The existing implementations [4-5, 11] use full adders (FA) and half adders (HA) to construct this module. The series of full adders in any column can be replaced by the novel compressors that take the same number of inputs. In the proposed implementation use of suggested compressors is done which not only reduces the delay and power consumption but also the area of the circuit. For a modulo 2^{16}+1 multiplier in IDEA cipher the FA implementation requires fifteen full adders in series in any column, these fifteen full adders can be replaced by two 7:2 compressors, one 5:2 compressor and two 3:2 compressors.

Computation of the correction factor COR for compressor implementation involves computing only $COR2$, because $COR1$ is obtained based on repositioning of the

partial product term, which is same for both implementations. The correction factor $COR2$ computation for FA implementation which has $n-1$ stages of additions is shown in [5]. And the $COR2$ computation for the proposed multiplier implementation using the compressors also yields the same result. Since, any (p, 2) compressor can be primarily designed using $(p-2)$ FAs which give $p-2$ carry outs with 2^n weight. Hence, the overall correction factor COR computation for FA implementation and compressor implementation yield the same result i.e, 3 as shown in (4).

4.3. FINAL STAGE ADDITION

The partial product reduction module gives one n-bit carry vector and one n-bit sum vector which need to be added in the final stage addition module. Very efficient parallel prefix adders are designed to do this operation [2].

Suppose S and C are sum and carry vectors produced after the partial product reduction section. As it is shown in the work of Zimmerman [2] that:

$$|S + C + 1|_{2^n+1} = |S + C + \overline{Cout}|_{2^n} \quad (5)$$

The constant '1' in the above equation can be obtained from the final partial product term COR (4) which is the constant '3'. Hence the new final partial product is the constant '2'. The equation (5) can be implemented using an inverted End-Around-Carry adder [2, 4-5]. Even though the propagation delay of this adder is in the order of logn, it has a drawback of high interconnect complexity and high fan-out. This can be overcome by sparse tree adder [10] based on the prefix network logic. The sparse tree adder takes the carry for every four bits instead of taking it at every stage and using a carry select block for selecting the final carry after the prefix network. This sparse tree adder was proven to be much more efficient in terms of both delay and power when compared with the existing prefix tree based adders [12]. Hence this sparse tree can be used to design Inverted-End-Around-Carry adder. The newly designed Inverted-End-Around-Carry adder using sparse tree adder structure is shown in Fig. 6. The proposed implementation of the modulo $2^{16}+1$ multiplier for IDEA cipher is shown in Fig. 7.

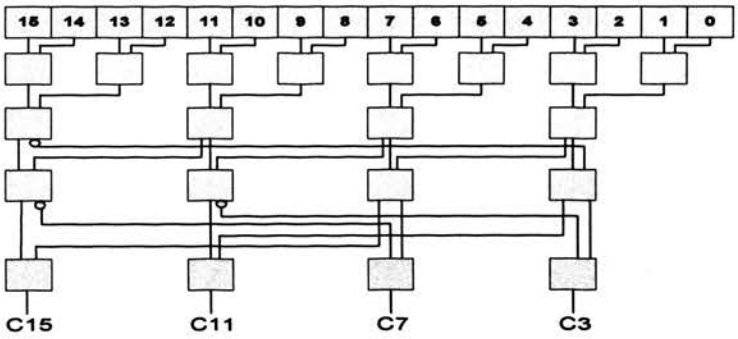


Fig. 6 Inverted EAC adder implemented using sparse tree structure

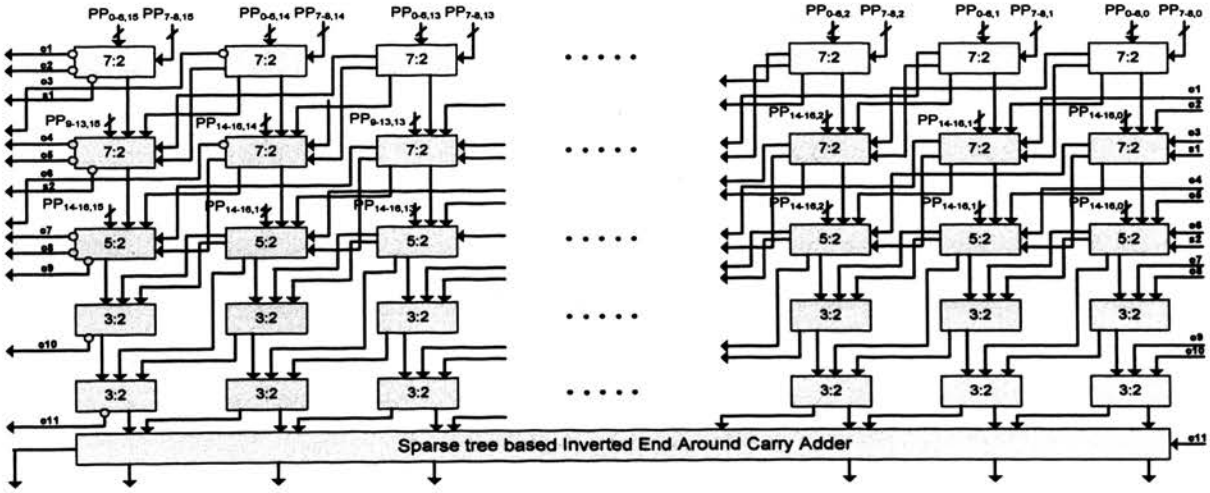


Fig. 7 Proposed implementation of the mod $2^{16}+1$ multiplier using efficient compressor

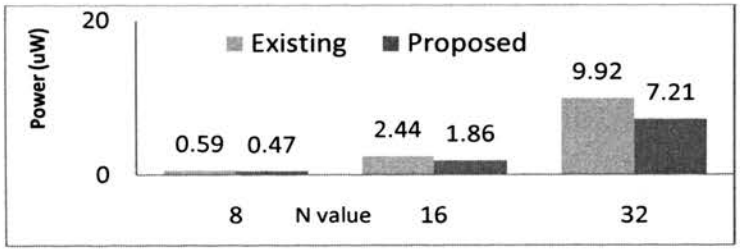
5. SIMLATION AND RESULTS

5.1. SIMULATION ENVIRONMENT

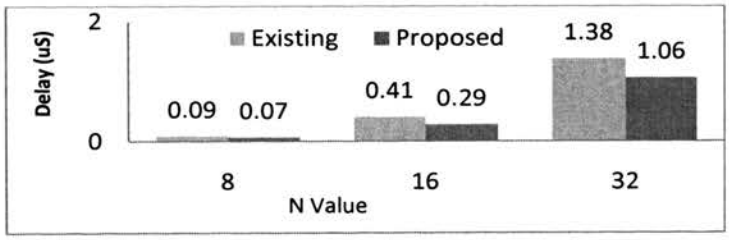
All the simulations have been carried out using Mentor Graphics design suite. The calculation of power and delay are carried out using the Eldo simulation tool. The power and delay calculations are done using the 0.18μ CMOS technology. The simulations are performed at 1.8V with all inputs fed at a frequency of 100 MHz.

5.2. SIMULATION RESULTS

The proposed and existing implementations of the modulo 2^n+1 multipliers for different values of n (8, 16, 32) are done with CMOS tsmc018 technology. The power and delay comparisons with the existing implementations are given in Fig. 8a and Fig. 8b, respectively. For n=16 case which is used in the IDEA cipher, the power and delay reduction ratios are 23.77% and 29.27%, respectively.



(a)



(b)

Fig. 8 (a) Power (b) Delay comparisons of existing and proposed multipliers

6. CONCLUSIONS

In conclusion, an efficient implementation of modulo 2^n+1 multiplier is proposed. The use of novel compressors in place of full adders resulted in considerable savings in terms of delay and power. The final stage adder is redesigned using sparse tree adder which has less interconnection complexity. Simulations have been performed on the proposed implementation and on the existing implementation. The proposed implementation is proven to perform better than the existing one in every aspect, (i.e, delay, power and power delay product).

7. REFERENCES

[1] Zimmermann, R., Curiger, A., Bonnenberg, H., Kaeslin, H., Felber, N., and Fichtner, W., "A 177 Mb/s VLSI implementation of the international data encryption algorithm," IEEE J. Solid-State Circuits, 1994, 29, (3), pp. 303–307.

[2] Zimmermann, R., "Efficient VLSI implementation of modulo ($2^n \pm 1$) addition and multiplication," IEEE trans. Comput, 2002, 51, pp. 1389-1399.

[3] Sousa, L., and Chaves, R., "A universal architecture for designing efficient modulo 2^n+1 multipliers," IEEE Trans. Circuits Syst. I, 2005, 52, pp. 1166–1178.

[4] Efstathiou, C., and Vergos, H.T., Dimitrakopoulos, G, and Nikolos, D.: "Efficient diminished-1 modulo 2^n+1 multipliers," IEEE Trans. Comput, 2005, 54, pp. 491–496.

[5] Vergos, H.T., and Efstathiou, C., "Design of efficient modulo 2^n+1 multipliers," IET Comput. Digit. Tech, 2007, 1, (1), pp. 49–57.

[6] Zimmermann, R., and Fichtner, W., "Low-power logic styles: CMOS versus pass-transistor logic," *IEEE J. Solid- State Circuits*, vol. 32, pp. 1079–1090, July 1997

[7] Veeramachaneni, S., Avinash, L., Rajashekhar, M., and Srinivas, M.B., "Efficient Modulo ($2^k \pm 1$) Binary to Residue Converters," System-on-Chip for Real-Time Applications, The 6th International Workshop on Dec. 2006 pp.195 – 200.

[8] Chang, C.H., Gu, J., and Zhang, M., "Ultra low-voltage lowpower CMOS 4-2 and 5-2 compressors for fast arithmetic circuits," IEEE J. Circuits and Systems I, Volume: 51, Issue: 10 pp: 1985- 1997,2004.

[9] Rouholamini, M., Kavehie, O., Mirbaha, A.P., Jasbi, S.J., and Navi, K., "A New Design for 7:2 Compressors," Computer Systems and Applications, 2007. AICCSA '07. IEEE/ACS International Conference on 13-16 May 2007 Page(s):474 – 478.

[10] 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, Volume 38, Issue 5, May 2003 Page(s):689 – 695.

[11] Zhongde, W., Graham, A., and William, C., "An efficient tree architecture for modulo 2^n+1 multiplication," VLSI Signal Processing 14(3): 241-248 (1996).

[12] Kogge, P., and Stone, H.S., "A parallel algorithm for the efficient solution of a general class of recurrence equations," *IEEE Trans. Comput,* vol. C-22, pp. 786–793, Aug 1973.

II. AN EFFICIENT IDEA CRYPTO-HARDWARE USING NOVEL MODULAR ARITHMETIC COMPONENTS

Rajashekhar Modugu and Minsu Choi
Department of Electrical & Computer Engineering
Missouri University of Science and Technology
Rolla, MO 65409, USA
{rrmt4b, choim}@mst.edu

Abstract— The cryptographic algorithms such as International Data Encryption Algorithm (IDEA) have found various applications in secure transmission of the data in networked instrumentation and distributed measurement systems. Modulo 2^n+1 multiplier and squarer play a pivotal role in the implementation of these algorithms. In this paper, an efficient hardware design of the IDEA using novel modulo 2^n+1 multiplier and squarer as the basic modules is proposed for faster, smaller and low-power IDEA circuits. Novel hardware implementation of the modulo 2^n+1 multiplier is shown by using the efficient compressors and sparse tree based inverted end around carry adders is given. The novel modules are applied on IDEA algorithm and the resulting implementation is compared both qualitatively and quantitatively with the IDEA implementation using the existing multiplier implementations. The measurement results show that the proposed implementation is faster and smaller and also consume less power than similar hardware implementations making it a viable option for efficient hardware designs.

Index Terms— Modulo $2^n + 1$ multiplier; International Data Encryption Algorithm (IDEA); Sparse-tree adder; Power/area/speed measurement;

1. INTRODUCTION

The demand for high security in communications channels, networked instrumentation and distributed measurement systems is ever growing rapidly. The confidentiality and security requirements are becoming more and more important to protect the data transmitted and received. This leads to the need for efficient design of cryptographic algorithms which offer data integrity, authentication, non-repudiation and confidentiality of the encrypted data across the communication channels. Various cryptographic algorithms have been studied and implemented to ensure security of these systems. In this paper, modulo 2^n+1 multiplier has been much focus as it has found its important role in IDEA algorithm. For example, the three major operations that decide the overall performance and delay of the IDEA [1, 4, 15] are modulo 216 addition, bitwise- XOR and modulo $2^{16}+1$ multiplication and the GF(2n) Montgomery multiplication and modular exponentiation can be implemented using repeated multiplication and squaring of the vectors. Among these operations, improving the delay and power efficiency of the modulo 2^n+1 multiplication operation leads to significant increase in the performance of the entire cryptographic cipher.

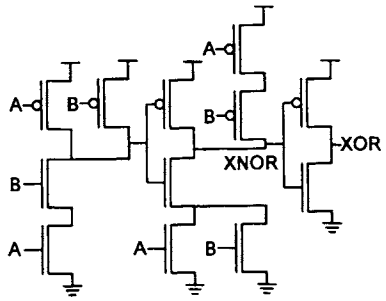
Numerous hardware implementations of the IDEA algorithm are proposed in the literature using different modulo $2^{16}+1$ multiplier architectures. The IDEA algorithm has been implemented in software [3] on Intel Pentium II 445 MHz with encryption rate of 23:53 Mb/Sec. Later, IDEA was realized on hardware chip by curiger et.all [1] with encryption rates up to 177 Mb/sec. By using a bit-serial implementation [4], which enables the IDEA to be fully, pipelined the encryption rates reached 500 Mb/sec with 125 MHz clock rate. The efficiency of the IDEA cipher can still be improved if efficient basic modules such as modulo multipliers and adders are used. The efficient implementation of the modulo 2^n+1 multiplier based on novel compressors and sparse tree based inverted end around carry adders is presented in [7]. Even though the architecture of the modulo multiplier is very efficiently proposed in [6], the hardware implementation and optimization are considerably improved in [7]. This is resulted by replacing the full adder arrays with the novel compressors and the final stage adder with the sparse tree based inverted end around carry adder.

The paper is organized as follows; Section II introduces multiplexor-based compressors. In Section III, the hardware implementation of modulo 2^n+1 multiplier is given. Section IV discusses the proposed implementation of the IDEA cipher which uses modulo $2^{16}+1$ multiplier. A comparison of this implementation to a recently proposed implementation is made in Section V. Conclusions are drawn in Section VI.

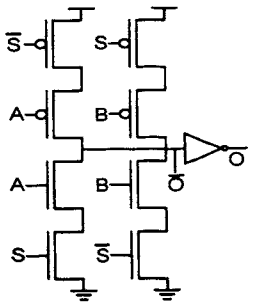
2. COMPRESSORS

2.1. MUX VS. XOR

Existing CMOS designs of 2-1 MUX and 2-input XOR are shown in Fig.1. According to [6], the CMOS implementation of MUX, it performs better in terms of power and delay compared to 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. The efficient implementation of compressors [7] is achieved by using both output and its complement of these gates. This also reduces the total number of garbage outputs.



(a)



(b)

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

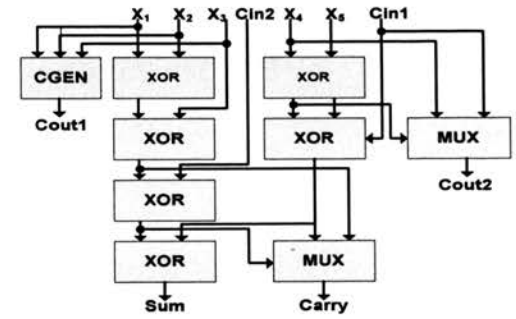
2.2. DESCRIPTION OF COMPRESSORS

A $(p,2)$ compressor with p inputs $X1,X2..Xp$ and two output bits Sum and Carry along with carry input bits and carry output bits is governed by the equation:

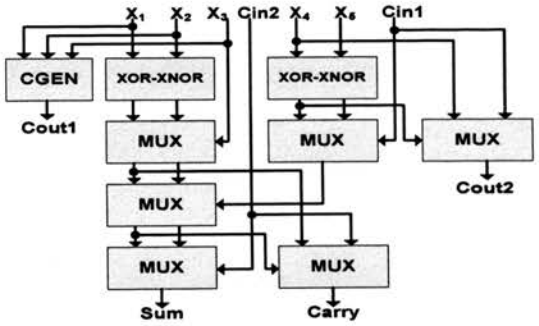
$$\sum_{i=1}^{p} X_i + \sum_{i=1}^{t} (C_{in})_i = Sum + 2(Carry + \sum_{i=1}^{t} (C_{out})_i)$$

Efficient design of the existing XOR-based 5:2 compressor [8-9], which takes 5 inputs and 2 carry inputs, is shown in Fig. 2a. The critical path delay of this compressor is $4\Delta - XOR$ (delay denoted by Δ).

The newly designed compressors use multiplexers in place of XOR gates, resulting in high speed arithmetic. 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 the CMOS implementation of the MUX if both the select bit and its complement are generated in the previous stage then its output is 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 is eliminated which reduces the overall delay in the critical path. The new MUX-based design of 5:2 compressor [7] is shown in Fig. 2b, the delay of which is $\Delta - XOR + 3\Delta - MUX$. CGEN block used in Fig. 2 can be obtained from the equation $Cout1 = (x1 + x2) \cdot x3 + x1 \cdot x2$.



(a)



(b)

Fig.2 5:2 compressors; (a) Existing design (b) New design

3. HARDWARE IMPLEMENTATION OF THE MOD 2^N+1

The hardware implementation of the modulo multiplier consists of three modules. First module is to generate partial products, second module is to reduce the partial products to two final operands and the last module is to add the Sum and Carry operands from partial products reduction to get the final result.

3.1. PARTIAL PRODUCTS GENERATION

The $n \times n$ partial products matrix is obtained from the n+1-bit input vectors. This partial product matrix is generated after repositioning the bits of the initial partial product matrix based on several observations presented in [6]. The partial products bits can be computed from AND, OR and NOT gates. The most complex function of partial product generation module is $\overline{p_{n-1,n-1} V\, q_{n-2}}$, where $p_{i,j} = a_i b_j \; and \; q_i = p_{n,i} V p_{i,n}$.

3.2. PARTIAL PRODUCTS REDUCTION

This is the most important module which largely determines the critical path delay and the overall performance of the multiplier. Hence this module needs to be designed so as to get minimum delay and consume less power.

The implementations from the literature [5, 6, and 13] use full adders (FA) and half adders (HA) to construct this module. The series of full adders in any column can be replaced by the novel compressors that take the same number of inputs. In the proposed implementation use of suggested compressors is done which not only reduces the delay and power consumption but also the area of the circuit. For a modulo $2^{16} + 1$ multiplier in IDEA cipher the Carry Save Adder (CSA) array implementation using Full Adders requires fifteen full adders in series in any column, these fifteen full adders can be replaced by two 7:2 compressors, one 5:2 compressor and two 3:2 compressors.

Correction factor computation is an important step while generating the partial products matrix. The full adder implementation [6] and the compressor based implementations [7] result in the same value. Because of the space constraints, computation of the correction factor COR for full adder implementation [6] is not given in this paper. COR computation for compressor implementation involves computing only COR2, because COR1 is obtained based on repositioning of the partial product term, which is same for both implementations. The correction factor COR2 computation for FA implementation which has n-1 stages of additions is shown in [6]. And the COR2 computation for the proposed multiplier implementation using the compressors also yields the same result. Since, any (p, 2) compressor can be primarily designed using (p-2) FAs which give p-2 carry outs with 2n weight. Hence, the overall correction factor COR computation for CSA array FA implementation and compressor implementation yield the same result i.e, 3 as shown in [5].

3.3. FINAL STATE ADDITION

The partial product reduction module gives one n-bit carry vector and one n-bit sum vector which need to be added in the final stage addition module. Very efficient parallel prefix adders are designed to do this operation [2].

Suppose S and C are sum and carry vectors produced after the partial product reduction section. As it is shown in the work of Zimmerman [2] that:

$$|S + C + 1|_{2^n+1} = |S + C + \overline{Cout}|_{2^n} \quad (1)$$

The equation (1) can be implemented using an inverted End- Around-Carry adder [2, 5, and 6]. Even though the propagation delay of this adder is in the order of $log_2 n$, it has a drawback of high interconnect complexity and high fan-out. This can be overcome by sparse tree adder [12, 16] based on the prefix network logic. The sparse tree adder generates the carry for every four bits instead of generating it at every stage and using a carry select block for selecting the final carry after the prefix network. This sparse tree adder was proven to be much more efficient in terms of both delay and power when

compared with the existing prefix tree based adders [14]. Hence this sparse tree can be used to design Inverted-End-Around-Carry adder.

The newly designed Inverted-End-Around-Carry adder using sparse tree adder structure is used in the final stage addition of the modulo multiplier is shown in Fig. 3. The proposed implementation of the modulo $2^{16}+1$ multiplier for IDEA cipher is shown in Fig. 3 and $R_{16}R_{15}.....R_2R_1R_0$ represents the final product of the modulo $2^{16}+1$ multiplier.

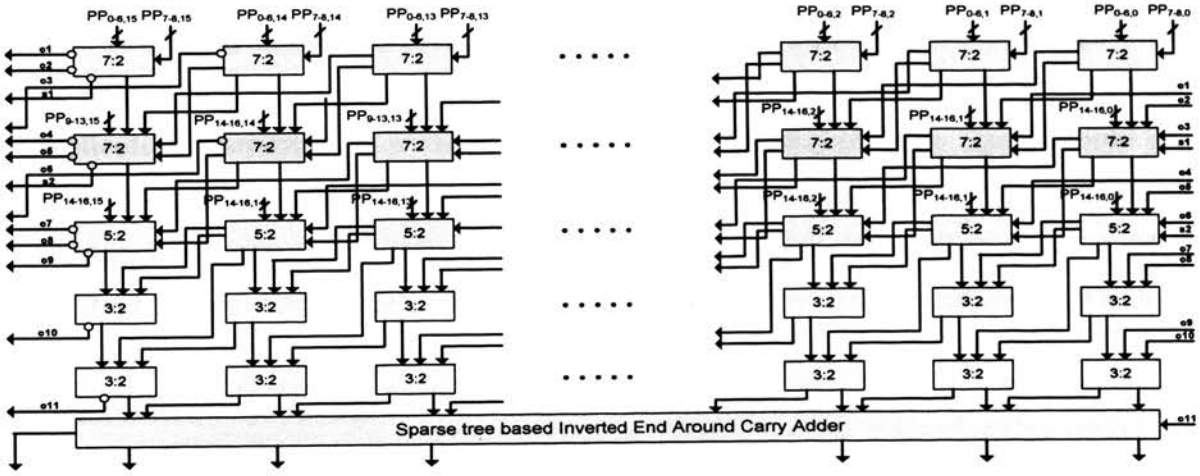


Fig. 3 Proposed implementation of the mod $2^{16}+1$ multiplier using efficient compressor

4. NOVEL IMPLEMENTATION OF INTERNATIONAL DATA ENCRYPTION ALGORITHM (IDEA) USING MODULO 2^N+1

The modulo 2^n+1 computation is an integral part of the International Data Encryption Algorithm (IDEA) where n = 16 [1, 4, 15]. Three major operations that decide the overall delay and performance of IDEA cipher are:

1) modulo 2^{16} addition,

2) bitwise-XOR and 3) modulo 2^{16} + 1 multiplication.

As the first two operations take less time and are easy to implement, the delay and power efficiency of the entire IDEA cipher depends significantly on the modulo $2^{16}+1$ multiplication operation. Hence, the IDEA cipher is implemented using the proposed modulo multiplier and compared with the existing implementations. To encrypt a data block using IDEA cipher, the data should be processed through three modulo multiplication operations in a single round and the manipulated data again should pass through seven such rounds iteratively and a final output transformation to produce the final encrypted output. The IDEA cipher takes 64-bit input data and produces a 64-bit cipher text with a 128-bit key. The encryption and decryption algorithms in IDEA are almost identical except they utilize two different sets of sub key generated by the same key with different processes.

The IDEA encryption and decryption processes consist of eight rounds of data manipulation using sub keys and a final output transformation stage. In this cipher, all the operations are carried out on 16-bit sub-blocks. In the encryption process, the input data block of 64-bits is divided into 4 sub blocks of 16-bits each (X_1, X2, X3, X4). 52 sub keys for the encryption process are generated from the original 128-bit key by shifting a part of it. Out of the 52 sub keys, six different sub keys (i.e. $Z^{(r)}_1$;$Z^{(r)}_2$;$Z^{(r)}_3$,$Z^{(r)}_4$,$Z^{(r)}_5$ and $Z^{(r)}_6$, where r is the round number) are used for each round and the remaining 4 sub keys are used in the final output transformation stage. The 16-bit outputs at each round are represented as $Y^{(r)}_1$, $Y^{(r)}_2$, $Y^{(r)}_3$, $Y^{(r)}_4$ and W_1,W_2,W_3,W_4 are the outputs of the final output stage transformation. The 52 subkeys used for the decryption process are obtained using a different algorithm [17]. As shown in Fig. 4, the critical path consists of three modulo $2^{16}+1$ multiplication operations, two modulo 2^{16} addition operations and two 16-bit XOR

operations in each round. In the final output transformation stage, critical path consists of a single modulo $2^{16}+1$ multiplication operation. The throughput of the IDEA cipher can be improved, if the delay of the modulo $2^{16}+1$ multiplication operation is reduced in the pipelined implementation of the IDEA cipher. Fig. 4 shows the datapath of encryption process of the IDEA cipher and datapath of a single round with 4 pipeline stages with the proposed modulo multiplier.

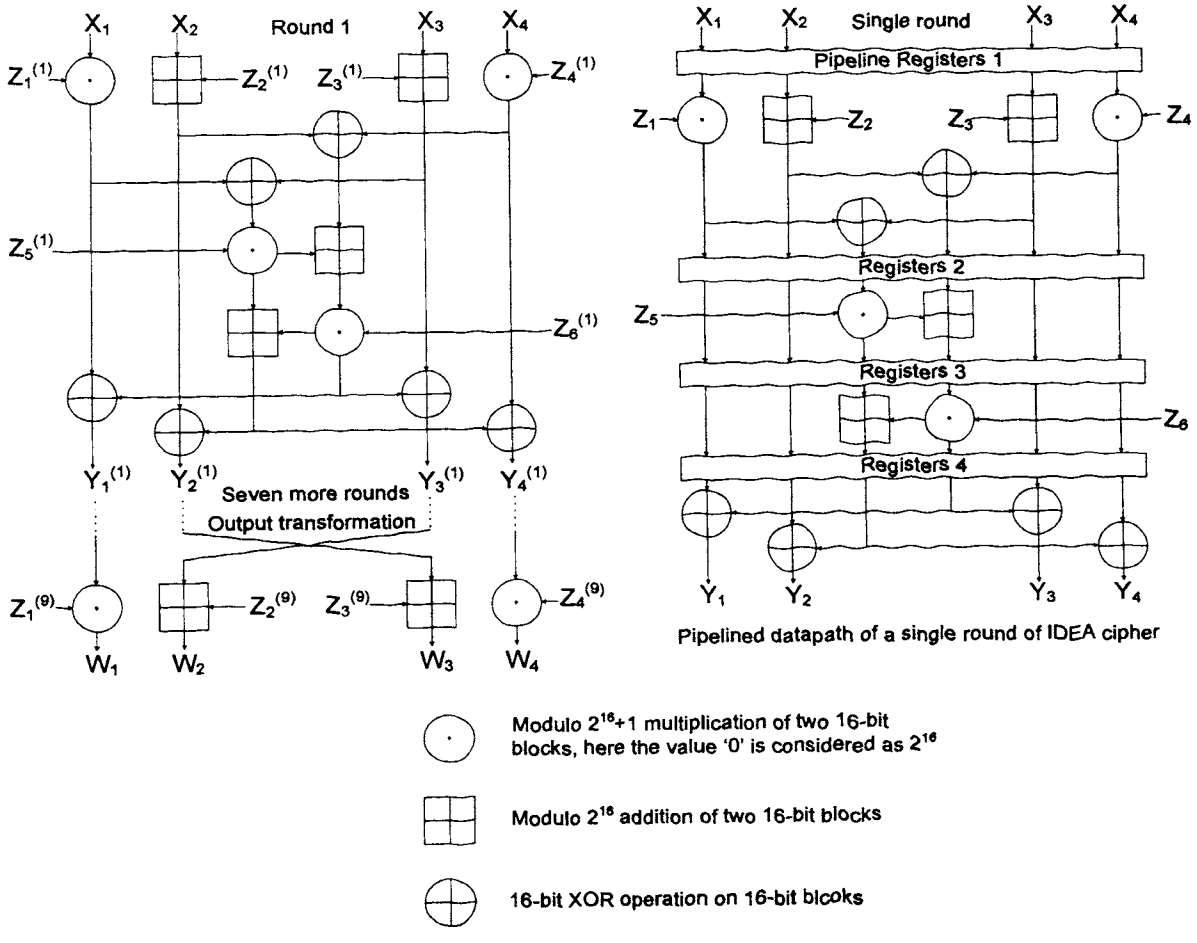


Fig. 4 Datapath of IDEA cipher with 4 pipeline stages

5. SIMULATION AND RESULTS

This particular design of the IDEA cipher with four pipeline stages using novel modulo 2n+1 multipliers is used to analyze and compare with the well-known IDEA cipher implementations. The use of the novel modulo multiplier improves the throughput and performance of the IDEA cipher significantly. The hardware implementation of the modulo multiplier consists of three modules. First module is to generate partial products, second module is to reduce the partial products to two final operands and the last module is to add the Sum and Carry operands from partial products reduction to get the final result.

5.1. SIMULATION ENVIRONMENT

All the simulations have been carried out using Mentor Graphics design suite. The IDEA cipher design is specified using verilog HDL and the multiplier descriptions are mapped on a 0.18 ¹m 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, the layouts are iteratively generated to get the circuits with minimum area. The calculation of power and delay are carried out using the Eldo simulation tool. The simulations are performed at 1.8V with all inputs fed at a frequency of 25 MHz.

5.2. SIMULATION RESULTS

The IDEA cipher is implemented using both the proposed multiplier and the multipliers presented in [6]. Various performance measurements for the IDEA cipher using both the proposed multiplier and the existing multiplier are parametrically obtained and listed in Table 1.

TABLE 1. COMPARISON OF THE PERFORMANCE MEASUREMENTS FOR IDEA CIPHER

Performance Measurement	Using proposed Multipliers	Using the multipliers in [6]	% Reduction
Encryption Rate Mb/sec	460.25	412.15	11.25
Critical Path delay	4.372	5.168	15.4
Area of the cipher (mm^2)	3.68	4.22	12.79

6. CONCLUSION

Hardware implementation of the IDEA cipher using novel modulo 2^n+1 multipliers is presented in this paper. It is shown that the proposed modulo 2^n+1 multiplier improves the performance of the various cryptographic algorithms used in secure communication systems of networked instrumentation and distributed measurement systems. Efficient compressors and sparse tree based inverted end around carry adders are used to reduce the delay and complexity of the multiplier. Simulations are performed on the known implementation and the proposed implementation. The presented implementation is proven to perform better than the existing one in various aspects, (i.e, throughput and critical path delay).

7. REFERENCES

[1] Zimmermann, R., Curiger, A., Bonnenberg, H., Kaeslin, H., Felber, N., and Fichtner, W., "A 177 Mb/s VLSI implementation of the international data encryption algorithm," IEEE J. Solid-State Circuits, 1994, 29, (3), pp. 303-307

[2] Zimmerman, R., "Efficient VLSI implementation of modulo $2^n \pm 1$ addition and multiplication," IEEE trans. Comput, 2002, 51, pp. 1389-1399.

[3] Cheung, K.H., Tsoi, P.H.W., Leong, A., and Leong, M.P., "Tradeoffs in parallel and serial implementations of the international data encryption algorithm IDEA," Lecture Notes in Computer Science, vol. 2162, pp. 333–, 2001.

[4] Cheung, K.H., Tsoi, P.H.W., Leong, A., and Leong, M.P., "A bit-serial implementation of the international data encryption algorithm IDEA," 2000 IEEE Symposium on Field-Programmable Custom Computing Machines, pp. 122 –131, 2000.

[5] Efstathiou, C., Vergos, H.T., Dimitrakopoulos, G., and Nikolos, D., "Efficient diminished-1 modulo 2^n+1 multipliers," IEEE Trans. Comput, 2005, 54, pp. 491-496.

[6] Vergos, H.T., and Efstathiou, C., "Design of efficient modulo 2^n+1 multipliers," IET Comput. Digit. Tech, 2007, 1, (1), pp. 49-57.

[7] Rajashekar, M., Nohpill, P., and Minsu, C., "A Fast Low-Power Modulo 2^n+1 Multiplier Design," 2009 IEEE International Instrumentation and Measurement Technology Conference, pp.951-956, May 2009.

[8] Zimmermann, R., and Fichtner, W., "Low-power logic styles: CMOS versus pass-transistor logic," IEEE J. Solid- State Circuits, vol. 32, pp. 1079-1090, July 1997.

[9] Veeramachaneni, S., Avinash, L., Rajashekhar, M., and Srinivas, M.B., "Efficient Modulo $2^k \pm 1$ Binary to Residue Converters System-on-Chip for Real-Time Applications," The 6th International Workshop on Dec. 2006 pp.195 - 200.

[10] Chang, C.H., Gu, J., and Zhang, M., "Ultra low-voltage lowpower CMOS 4-2 and 5-2 compressors for fast arithmetic circuits," IEEE J. Circuits and Systems I, Volume: 51, Issue: 10 pp: 1985- 1997, 2004.

[11] Rouholamini, M., Kavehie, O., Mirbaha, A.P., Jasbi, S.J., and Navi, K., "A New Design for 7:2 Compressors," Computer Systems and Applications, 2007. AICCSA '07. IEEE/ACS International Conference on 13-16 May 2007 Page(s):474 – 478.

[12] 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, Volume 38, Issue 5, May 2003 Page(s):689 - 695.

[13] Zhongde, W., Graham, A., Jullien, B., and William, C., "An efficient tree architecture for modulo 2^n+1 multiplication," VLSI Signal Processing 14(3): 241-248 (1996).

[14] Kogge, P., and Stone, H.S., "A parallel algorithm for the efficient solution of a general class of recurrence equations," IEEE Trans. Comput, vol. C-22, pp. 786-793, Aug 1973.

[15] Curiger, C., "VINCI: VLSI Implementation of the New Secret-keyBlock Cipher IDEA," Proc. of the Custom Integrated Circuits Conference, San Diego, USA, May 1993.

[16] Yan, S., Dongyu, Z., Minxuan, Z., and Shaoqing, L., "High Performance Low-Power Sparse-Tree Binary Adders," 8th International Conference on Solid state and Integrated Circuit Technology, ICSICT 2006.

[17] Yi-Jung, C., Dyi-Rong, D., and Yunghsian, S.H., "Improved Modulo (2^n+1) Multiplier for IDEA," J. Inf. Sci. Eng. 23(3): 911-923 (2007).

III. EFFICIENT ON-LINE SELF-CHECKING MODULO 2^N+1 MULTIPLIER DESIGN

Rajashekhar Modugu and Minsu Choi
Department of Electrical & Computer Engineering
Missouri University of Science and Technology
Rolla, MO 65409, USA
{rrmt4b, choim}@mst.edu

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 model is capable of 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.

1. INTRODUCTION

In the recent years, the number of 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; 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, 12]. 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 [13], 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 [25, 33, 34]. 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 on-line, 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 [14, 15, 16, 17] using different coding schemes such as parity prediction, 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 [28, 29]. Parity code schemes for memory systems and register files may achieve fault secure property at low hardware

cost; sometimes they fail to achieve the fault secure property in arithmetic circuits [20, 22]. In the self-checking arithmetic circuits using parity prediction scheme [24] 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% to 50% when parity prediction codes are used [19]. 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 [30, 31, and 32]. These self-checking circuits use parity prediction schemes and arithmetic code schemes in the design by trading off with the hardware overhead, performance penalty and fault secureness. Few of the self-checking circuits are described as follows. In [23], self-checking adder circuits based on arithmetic codes were proposed [26, 27]. These checkers using arithmetic codes suffered from hardware complexity and overhead. An alternative method to design self-checking adders [24] 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 [14], 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 [25], 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 [17], 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 [17, 40], 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 sub circuit 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, a new hardware implementation of the self-checking modulo $2^n + 1$ multiplier based on residue codes is proposed. 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% to 45% for different values of n.

The paper is organized as follows; Section II introduces multiplexor-based compressors, which are used in the self-checking multiplier design [5]. Section III 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 IV. Conclusions are drawn in Section V.

2. PRELIMINARIES AND REVIEWS

2.1. MUX VS. XOR

Multiplexor (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 $X\bar{Y} + \bar{X}Y$. The same XOR can be implemented using MUX with inputs $X, \bar{X}$ 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 $X_1, X_2 \ldots\ldots X_{p-1}, X_p$ 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:

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

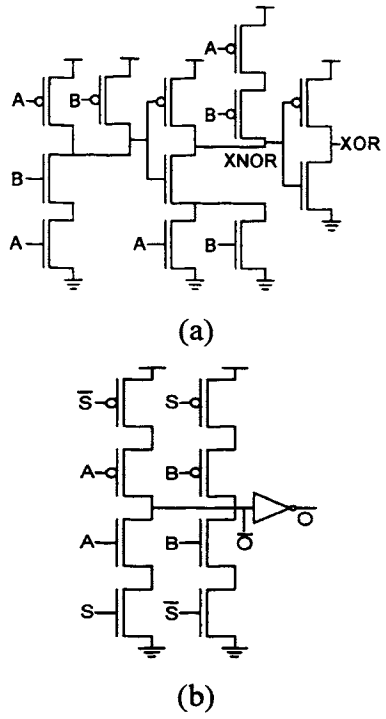
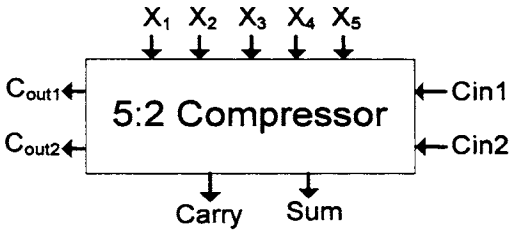
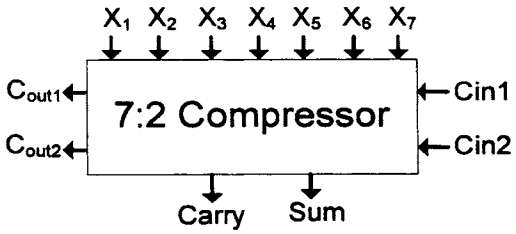


Fig.1 CMOS implementation of 2-input (a) XOR (b) MUX

For example, a (5, 2) compressor takes 5 inputs and 2 carry inputs and generate a Sum and Carry bit along with two carry out 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].



(a)



(b)

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

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 their complements 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 Δ (*XOR*) $+3\Delta$ (*MUX*) and 4Δ (*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 Fig. 3, Fig. 4 can be obtained from the equation $Cout1 = (x1 + x2) \cdot x3 + x1 \cdot x2$.

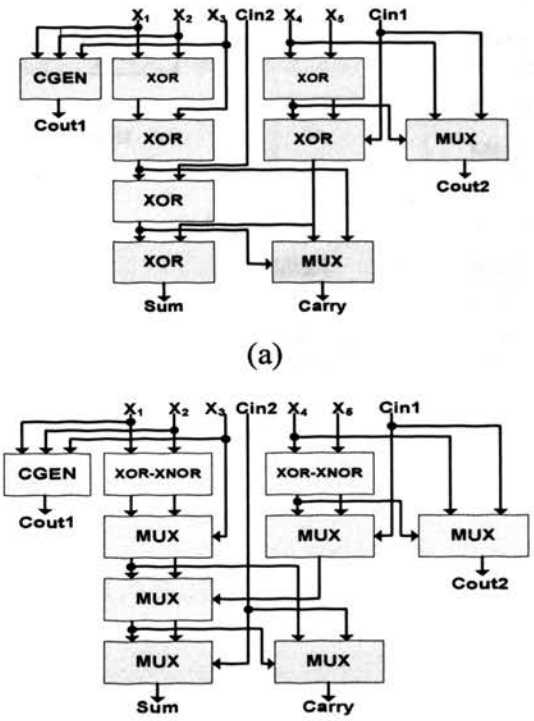


Fig. 3 5:2 compressors; (a) Existing design (b) New design

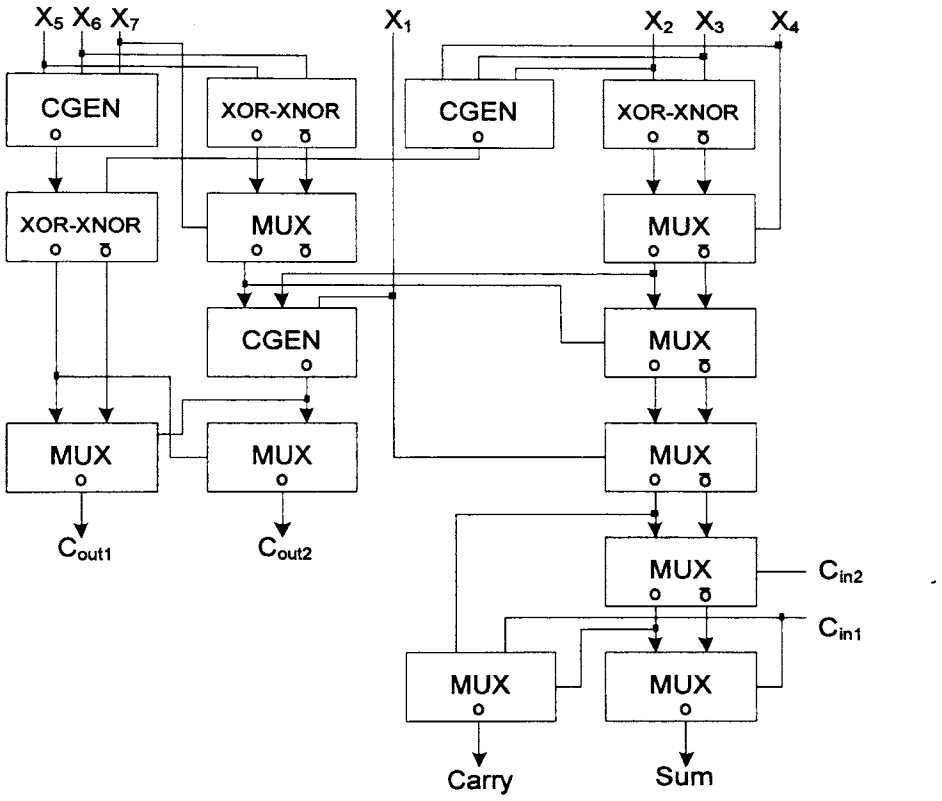


Fig. 4 Proposed MUX-based design of 7:2 compressor

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 [41] is delay effective and has regular structure suitable for efficient hardware implementation. The binary addition of two numbers using a parallel prefix network is done as follows. Let $A = a_{n-1}a_{n-2} \ldots a_1a_0$ and $B = b_{n-1}b_{n-2} \ldots b_1b_0$ 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 ± as follows:

$(g_i, p_i) \circ (g_{i-1}, p_{i-1}) = (g_i + p_i.g_{i-1}, p_i.p_{i-1}) = (g_{i:i-1}, p_{i:i-1})$ where + is the logical *OR* operator and. is the logical *AND* operator.

The carry outs (C_i) for all the bit positions can be 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 carry out of the addition and add it to the least significant bit of the sum vector. Similarly, in inverted End Around Carry adders the carry out is inverted and fed back to the least significant bit of the sum vector. The parallel prefix network based Inverted EAC adder [42] achieves the addition of the input operands by recirculating the generate and the propagate bits at each existing level in *log2n* 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:

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

In the above equation $\overline{(G_i, P_i)} = (\bar{G}, P)$,

where $(G_i, P_i) = (g_i, p_i) \circ (g_{i-1}, p_{i-1}) \circ \ldots (g_1, p_1) \circ (g_0, p_0)$.

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

In some cases it is not possible to compute (G^*_i, P^*_i) in *log2n* stages, then in these cases the equations in (2) are transformed into the equivalent ones as shown in Eq. (4) by using the following property [42]:

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

$$G^x = g + p.\bar{G} = \overline{\overline{g + p.\bar{G}}} = \overline{\bar{g}.(\bar{p} + G)}$$
$$= \overline{(\bar{g}.\bar{p}) + \bar{g}G} = \overline{\bar{p} + \bar{g}G} \quad (3)$$

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

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:

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

The new carry outs can be computed using the following equation:

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

Hence, the transformations used above to achieve the parallel prefix computation in *log2n* stages result in more number of carry merge cells and thereby adding more number of interstage wires. Parallel prefix adders suffer from excessive inter-stage 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 K^{th} (K = 4, 8 ...) carry is computed. The value of K is chosen based on the sparseness of the tree, generally for 16 bit and 32-bit adders K is chosen as 4 [36, 37, 38]. The higher value of K results in higher value of non-critical path delay compared to critical path delay of $O(log2n)$ 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 4, the carries are computed for bit positions -1, 3, 7 and 11. Here, bit position -1 corresponds to the inverted carry out $(\overline{(G_{15}, P_{15})})$ of the bit position 15. The carry out equations for the 16-bit sparse tree IEAC are as follows:

$$C^*_{-1} = \overline{(G_{15}, P_{15})} = \overline{(g_{15}, p_{15}) \circ (g_{14}, p_{14}) \circ \dots \dots \circ (g_1, p_1) \circ (g_0, p_0)}$$

$$C^*_3 = (G_3, P_3) \circ \overline{(G_{15:4}, P_{15:4})} = (g_3, p_3) \circ \dots\dots \circ (g_0, p_0) \circ \overline{(g_{15}, p_{15}) \circ \dots \dots \circ (g_4, p_4)}$$
$$= \overline{(\overline{P_3}, \overline{G_3}) \circ (G_{15:4}, P_{15:4})}$$

$$C^*_7 = (G_7, P_7) \circ \overline{(G_{15:8}, P_{15:8})} = (g_7, p_7) \circ \dots\dots \circ (g_0, p_0) \circ \overline{(g_{15}, p_{15}) \circ \dots \dots \circ (g_8, p_8)}$$
$$= \overline{(\overline{P_7}, \overline{G_7}) \circ (G_{15:8}, P_{15:8})}$$

$$C_{11}^{*} = (G_{11}, P_{11}) \circ \overline{(G_{15:12}, P_{15:12})}$$
$$= (g_{11}, p_{11}) \circ \cdots\cdots \circ (g_0, p_0) \circ \overline{(g_{15}, p_{15}) \circ \ldots\ldots \circ (g_{12}, p_{12})}$$
$$= \overline{(P_{11}, G_{11}) \circ (G_{15:12}, P_{15:12})}$$

Figure 5 shows the finalized 16-bit sparse tree Inverted EAC adder. From Fig. 5, observe that all the carry outs are computed in *log2n* stages with less number of carry merge cells and reduced inter-stage wiring intensity [36]. The implementation of the sparse tree based EAC is similar to IEAC shown in Fig. 5, except the carry is not inverted.

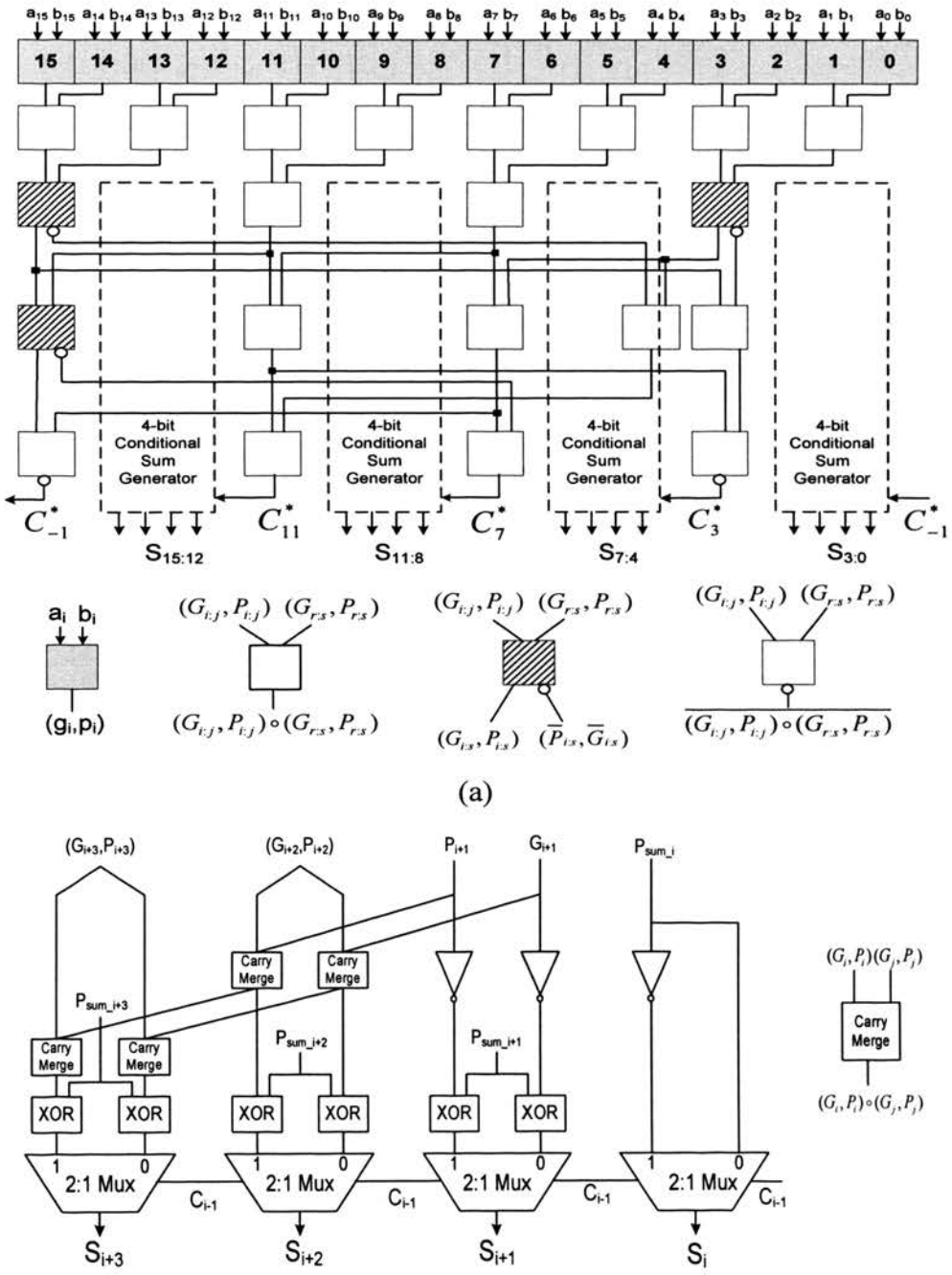


Fig. 5 (a) 16-bit sparse tree based Inverted EAC adder (b) 4-bit conditional sum generator

The Conditional Sum Generator (CSG) shown in Fig. II-C is implemented using ripple carry adder logic, 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 1-in-4 select line generate the final sum vector. The conditional sum generator is shown in Fig. 5 (b). The final sum is generated in *log2n* stages in IEAC sparse tree adder with less number of cells and less inter-stage 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+1bits. 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 '1' only when the inputs are 2^n and 2^n, respectively. $X \cdot Y \ mod\ 2^n + 1$ 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), observe that it results in an $(2^n + 1) \times (n + 1)$ partial products matrix. This matrix is modified into an $n \times n$ partial products matrix based on several assumptions [43]. The conversion of the $(2^n + 1) \times (n + 1)$ partial products matrix into $n \times n$ partial products matrix results in a correction factor of 3. 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 3 is added to the n x 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. This 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 products reduction module have to be added in the final stage addition module. A part of the correction factor 1 left behind is used in the final stage addition to take advantage of the following equation.

$$|S + C + 1|_{2^n+1} = \left|S + C + \overline{Cout}\right|_{2^n} \tag{7}$$

From (7), observe that the inverted carry out 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 Inverted End Around Carry (EAC) modulo $2n$ 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.

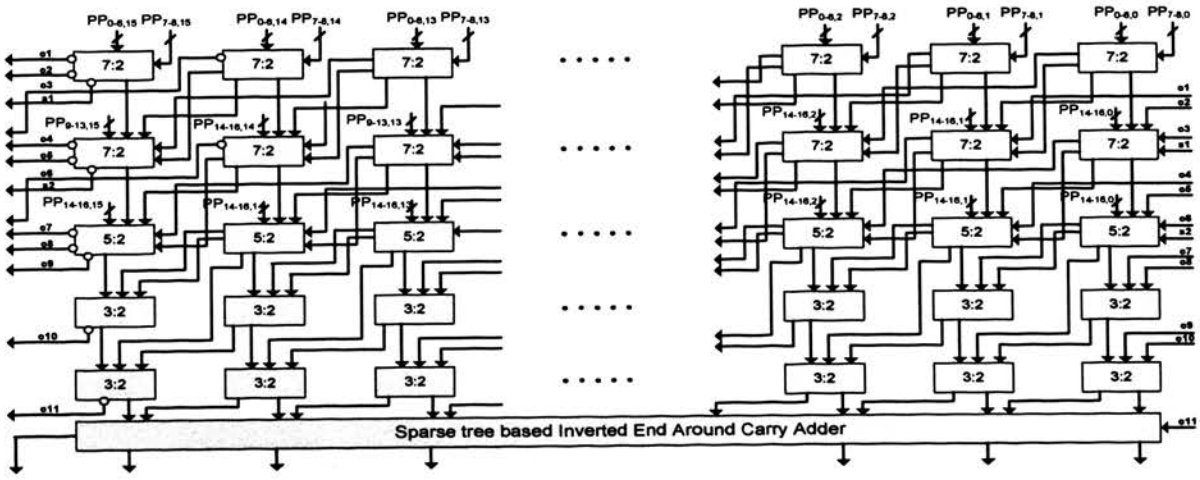


Fig. 6 Hardware implementation of the modulo 2^{16} + 1 multiplier

3. PROPOSED SELF-CHECKING MODULO $2^N + 1$ MULTIPLIER DESIGN

For any self-checking circuit to detect online-errors, the circuit has to comply with a set of properties. These properties are described below [17].

- Code disjointness: If each non-code input is mapped to a non-code 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 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. Using the fault secure property, for an input fault the circuit either generates a correct output or detects the fault. In the self-testing circuits faults can be detected by applying an input vector.

In this section, the analysis of the self-checking multipliers using residue codes, which includes fixing a fault model and selection of the check bases is given. The analysis of the integer multipliers is 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 fixed fault model and appropriate check bases are designed for the same in [15]. The brief explanation of the self-checking two's complement multiplier [15] is given below. Let the inputs and outputs of self-checking multiplier circuit S are encoded using the error-detecting code I(O) and 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 (i) [S^f (i)]. Then, S is fault-secure for fault set F () for any input i ∈ I and for any fault $f \in F, S(i) \neq S^f(i)$ implies $S^f(i) \notin O$. The basic block diagram of the multiplier with residue code check is shown in Fig. 7.

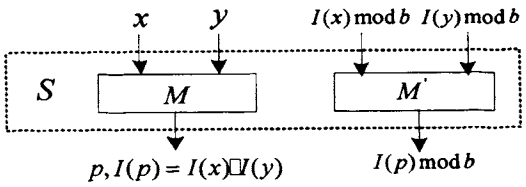


Fig. 7 A block diagram of the multiplier with residue code check

In the above figure, M represents the multiplier and M' represents the modulo multiplier with residue check base b ∈ N. $I(x) \mod b, I(y) \mod b$ and $I(p) \mod b$ represent the residues of the inputs and output with check base b. The efficient implementation of the modulo generators is given 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 in and only if

$$[p^f, I(p) mod\ p] \notin O$$
$$\Leftrightarrow I(p) mod\ b \neq I(p^f) mod\ b$$
$$\Leftrightarrow |I(p) - I(p^f)| mod\ b \neq 0$$

To simplify the analysis, let us assume a fault model for the multiplier with residue codes. For the fault set F, assume fault caused by a primitive block in the multiplier circuit S [15]. 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:

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

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

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

Hence, to achieve low hardware costs minimum value of a check base is selected which satisfies the above result. The selection of the check bases in the residue code based circuits is an important task. In selecting the check bases, first the absolute error set $E(M)$ caused by the single faults have to be characterized. For different set 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 will 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 [15] is given below.

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 $F \leftrightarrow S$ is self-testing for every fault from the set F.

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, $\text{mod}_{Mult}\ b$ computes the residue of the multiplier output i.e, $I(p)$ mod b. Dual Rail Check compares the output from $\text{mod}_{Mult}\ b$ 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 many not be always achieved. This guarantee the self-testing property of the multiplier circuit, the structure of the checkers has to be properly selected.

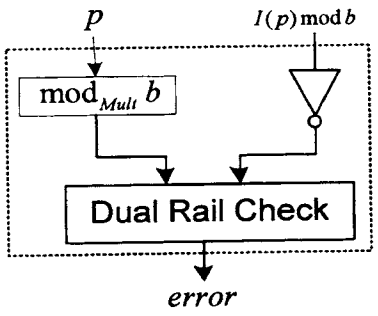


Fig. 8 A block diagram of the residue code checker

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 [15], the check base b is fixed as 3 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. If the fault causes an error set $E(M)$ given by:

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

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

$$b \in N \setminus \{\gamma 2^i | \gamma \in [1:5], i \in \{0, N\}\} \quad (10)$$

where N denotes the set of all natural numbers.

Hence, the smallest check base that satisfies (10) and to achieve the fault secureness for the given multiplier is 7. Since, modulo 3 generator is very popular, the check base 3 is used based on an 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 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 secured 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 [19]. As shown in Fig. 3, the fault model includes any fault affecting the gates that generate the outputs sum and carry outs.

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 are well studied in the literature [21]. In residue code based self-checking multipliers [15, 19, 21, and 23], 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 [19] is shown in Fig. 9 (i.e, in the modulo A generator block is either 2^k-1 or 2^k+1). From the figure, 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 against its check part. In this case, a dual-rail checker is used.

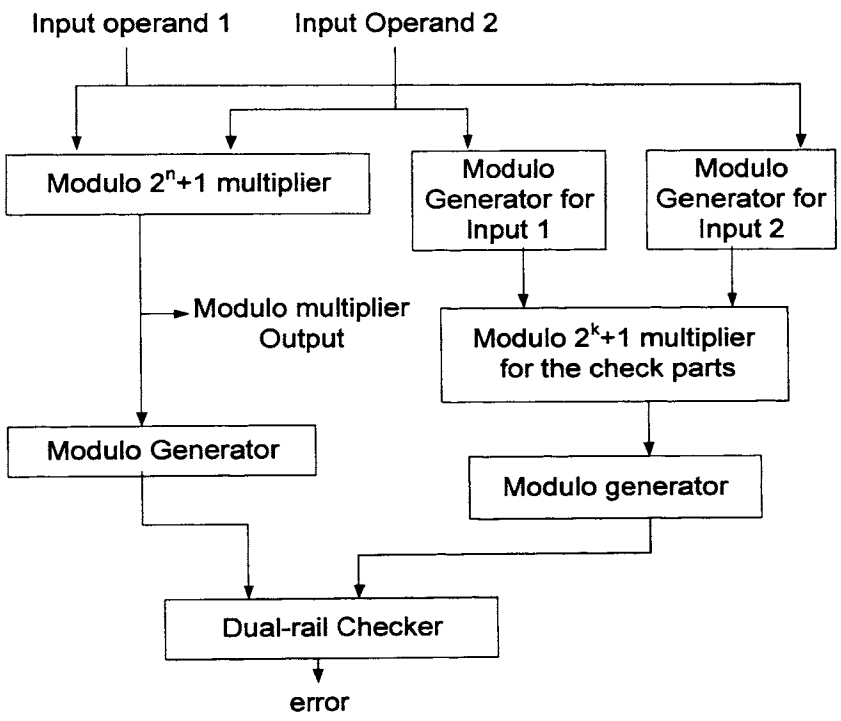


Fig. 9 A block diagram of the self-checking modulo 2^n+ 1 multiplier

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 [15, 21]. The hardware implementation and structure of the modulo multiplier is similar to integer multipliers [3].

In the array multipliers implementation reported in [35], 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 [19, 21]. 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 $\pm 2.2^i$. Hence, errors produced on both of these signals give an

arithmetic value of $\pm 3.2^i$. 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 [19, 21] 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 ci resulting in an error with final arithmetic value $\pm 2i+1$. In the second case, only a subset of the carries c_j ($j > i$) may structurally depend on ci producing an output error given below:
- $\pm[a_0(2^{i+1} \pm 2^{i+2} \ldots\ldots \pm 2^{i+k}) + a_1(2^{i1+1} \pm 2^{i1+2} \ldots\ldots \pm 2^{i1+k1}) + a_2(2^{i2+1} \pm 2^{i2+2} \ldots\ldots \pm 2^{i2+k2}) + \cdots + a_m(2^{im+1} \pm 2^{im+2} \ldots\ldots \pm 2^{im+km})]$ where $a_0, a_1 \ldots\ldots a_m \in [0,1]$ and $im + km < n, n$ is the width of the operands used in the adder.

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 [21, 15].

In the modulo 2n + 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 $\pm K.\ 2^i$ (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 4th or 8th bit. Hence, the error analysis gives same output error as proven in [21]. 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 Fig. 10 and Fig. 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.

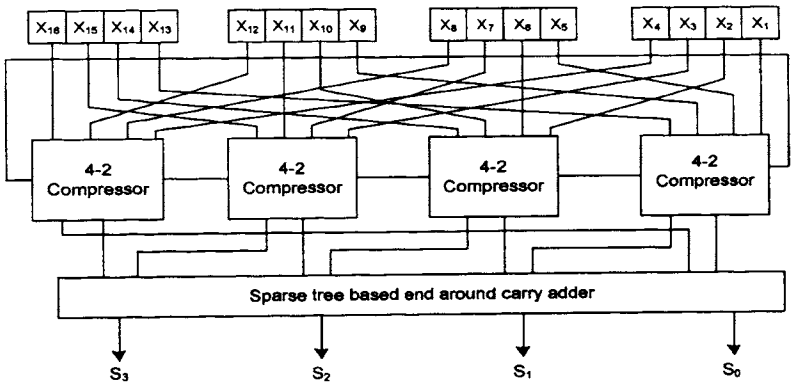


Fig. 10 Modulo generator with check base 2^4 - 1 for input width=16

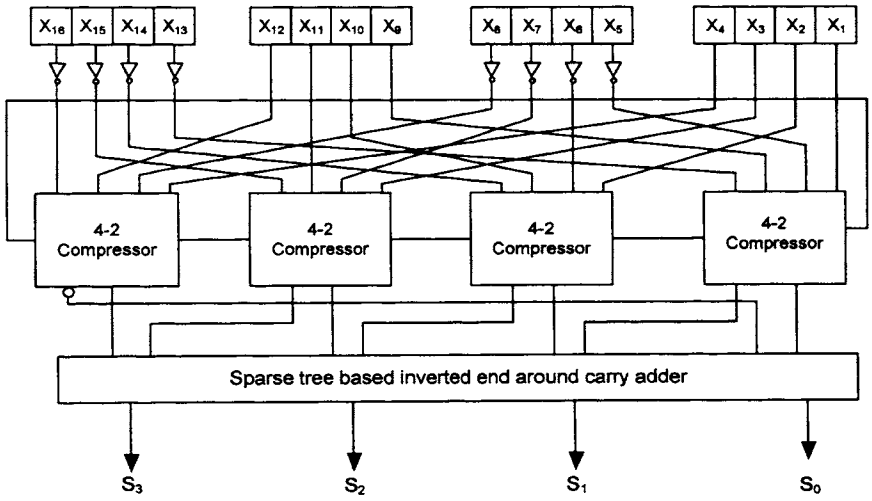


Fig. 11 Modulo generator with check base $2^4 + 1$ for input width=16

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 use of the compressors in the modulo generators and modulo multipliers result in good savings in terms of area overhead and delay. The self-checking two's complement multiplier is clearly studied for a fixed fault model and appropriate check bases are designed for the same in [15].

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 [39] 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 [39], 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.

$$A_{MWS} = A_{MG} + A_{Mm} + A_{DRC}$$

$$T_{MWS} = T_{MG} + T_{Mm} + T_{DRC}$$

$$A_{MW} = A_{MM}$$

$$T_{MW} = T_{MM}$$

In the above equations A_{MWS}, 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}, 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 the below table.

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 overhead	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

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, the layouts are iteratively generated to get the circuits with minimum area.

TABLE 2. EXPERIMENTAL RESULTS SHOWING THE AREA AND DELAY COMPARISON OF MODULO $2^N + 1$ MULTIPLIERS WITH AND WITHOUT SELF-CHECKING PROPERTY

n	$A_1(\mu m^2)$	$A_2(\mu m^2)$	% Area overhead	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	1608847	194624	21	15.315	15.621	2

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 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, modulo multipliers are efficiently designed using novel compressors. The final stage addition modules of the modulo multipliers and modulo generators are 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 the subsequent gates and generate an error at the output of the modulo multiplier. These selfchecking modulo multipliers are analyzed using unit-gate model and compared with the modulo multipliers without self-checking property. These models are designed for different values of the input length and simulated to get the experimental results. The results show that the proposed self-checking multiplier results in 20% to 45% area overhead and 2% to 7% performance penalty for n = 64 to 8.

6. REFERENCES

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

[2] Beckmann, P.E., and Musicus, B.R., "Fast fault-tolerant digital convolution using a polynomial residue number system," IEEE Transactions on Signal Processing, vol. 41, issue 7, pp. 2300-2313.

[3] Rajashekar, M., Nohpill, P., and Minsu, C., "A Fast Low-Power Modulo 2^n+1 Multiplier Design," 2009 IEEE International Instrumentation and Measurement Technology Conference, pp.951-956, May 2009.

[4] Rajashekar, M., Nohpill, P., Yong-Bin, K., and Minsu, C., "Efficient On-line Self-Checking Modulo 2^n+1 Multiplier Design," submitted to the 24th IEEE International Symposium on Defect and Fault Tolerance in VLSI Systems, October 2009.

[5] Zimmermann, R., and Fichtner, W., "Low-power logic styles: CMOS versus pass-transistor logic," IEEE J. Solid-State Circuits, vol. 32, pp. 1079-1090, July 1997.

[6] Veeramachaneni, S., Avinash, L., Rajashekhar, M., and Srinivas, M.B., "Efficient Modulo $2^n\pm1$ Binary to Residue Converters," The 6th International Workshop on System-on-Chip for Real-Time Applications, pp.195-200, Dec. 2006.

[7] Chang, C.H., Gu, J., and Zhang, M., "Ultra low-voltage lowpower CMOS 4-2 and 5-2 compressors for fast arithmetic circuits," IEEE J. Circuits and Systems I, Volume: 51, Issue: 10 pp: 1985- 1997, 2004.

[8] Rouholamini, M., Kavehie, O., Mirbaha, A.P., Jasbi, S.J., and Navi, K., "A New Design for 7:2 Compressors," Computer Systems and Applications, 2007. AICCSA '07. IEEE/ACS International Conference on 13-16 May 2007, pp:474 - 478.

[9] Curiger, C., "VINCI: VLSI Implementation of the New Secret-keyBlock Cipher IDEA," Proc. of the Custom Integrated Circuits Conference, San Diego, USA, May 1993

[10] Zimmermann, R., Curiger, A., Bonnenberg, H., Kaeslin, H., Felber, N., and Fichtner, W., "A 177 Mb/s VLSI implementation of the international data encryption algorithm," IEEE J. Solid-State Circuits, 1994, 29, (3), pp. 303-307

[11] Sklavos, N., and Koufopavlou, O., "Asynchronous low power VLSI implementation of the International Data Encryption Algorithm," Electronics, Circuits and Systems, 2001. ICECS 2001. The 8th IEEE International Conference on, vol.3, no, pp.1425-1428 vol.3, 2001.

[12] Leibowitz, L.M., "A simplified binary arithmetic for the Fermat number transform," IEEE Trans. Acoust. Speech Signal Process, 1976, 24, pp. 35659.

[13] Vasudevan, D.P., and Lala, P.K., "A technique for modular design of self-checking carry-select adder," Defect and Fault Tolerance in VLSI Systems, 2005. DFT 2005. 20th IEEE International Symposium on, vol, no, pp. 325-333, 3-5 Oct. 2005.

[14] Sparmann, U., and Reddy, S.M., "On the effectiveness of residue code checking for parallel two's complement multipliers," Fault-Tolerant Computing, 1994. FTCS-24. Digest of Papers, Twenty-Fourth International Symposium on, vol, no, pp.219-228, 15-17 Jun 1994.

[15] Marienfeld, D., Sogomonyan, E.S., Ocheretnij, V., and Gossel, M., "New Self-checking Output-Duplicated Booth Multiplier with High Fault Coverage for Soft Errors," Test Symposium, 2005. Proceedings. 14th Asian, vol, no, pp.76-81, 21-21 Dec. 2005.

[16] Hunger, M., and Marienfeld, D., "New Self-Checking Booth Multipliers," International Journal of Applied Mathematics and Computer Science Vol. 18, No. 3, 2008.

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

[18] Noufal, I.A., and Nicolaidis, M., "A CAD framework for generating self-checking multipliers based on residue codes," Design, Automation and Test in Europe Conference and Exhibition 1999. Proceedings, vol, no, pp.122-129, 1999.

[19] Garcia, O.N., and Rao, T.R.N., "On the method of checking logical operations," Proc. 2nd Annual Princeton Conf. Inform. Sci. Sys, pp. 89-95, 1968.

[20] Sparmann, U., "On the check base selection problem for fast adders," VLSI Test Symposium, 1993. Digest of Papers, Eleventh Annual 1993 IEEE, vol, no, pp.62-65, 6-8 Apr 1993.

[21] Sellers, F.F., Hsiao, M.Y., and Bearnson, L.W., "Error Detecting Logic for Digital Computers," New-York, Mc GRAWHILL 1968.

[22] Peterson, W.W.W., "On checking an adder," IBM J.Res.Develop, Vol.2, pp.166-168, Apr. 1958.

[23] Fujiwara, E., and Haruta, K., "Fault-tolerant Arithmetic Logic Unit Using Parity Based Codes," Transactions of the IECE of Japan, pp.653-660, October 1981.

[24] Marienfeld, D., Sogomonyan, E.S., Ocheretnij, V., and Gossel, M., "A new self-checking multiplier by use of a code-disjoint sum-bit duplicated adder," Test Symposium, Proceedings. Ninth IEEE European, vol, no, pp. 30-35, 23-26 Mady 2004. ETS 2004. Proceedings.

[25] Sayers, I.L., and Kinniment, D.J., "Low-cost residue codes and their application to self-checking VLSI systems," Computers and Digital Techniques, IEE Proceedings E, vol.132, no.4, pp.197-202, July 1985.

[26] Debany, W.H., Macera, A.R., Daskiewich, D.E., Gorniak, M.J., Kwiat, K.A., and Dussault, H.B., "Effective concurrent test for a parallel-input multiplier using modulo 3," VLSI Test Symposium, 1992. '10th Anniversary. Design, Test and Application: ASICs and Systems-on-a-Chip', Digest of Papers, 1992 IEEE, vol, no, pp.280-285, 7-9 Apr 1992.

[27] Heckelman, R., and Bhavsar, D., "Self-testing VLSI," Solid-State Circuits Conference. Digest of Technical Papers. 1981 IEEE International, vol.XXIV, no, pp. 174-175, Feb 1981.

[28] Slegel T.J., and Veracca, R.J., "Design and performance of the IBM Enterprise System/9000 Type 9121 vector facility," IBM J. Res. Develop, vol. 35, pp. 367-381, May 1991.

[29] Nicolaidis, M., "Efficient implementations of self-checking adders and ALUs," Fault-Tolerant Computing, 1993. FTCS-23. Digest of Papers, The Twenty-Third International Symposium on, vol, no, pp.586-595, 22-24 Jun 1993.

[30] Kundu, S., and Reddy, S.M., "Embedded totally self-checking checkers: a practical design," Design and Test of Computers, IEEE, vol.7, no.4, pp.5-12, Aug 1990

[31] Xrysovalantis, K., Dimitris, N., Foukarakis, G., and Gnardellis, T., "New efficient totally self-checking Berger code checkers," Integration 28(1): 101-118 (1999).

[32] Kumar, K., and Lala, P.K.,"On-line Detection of Faults in Carry-Select Adders," ITC, pp.912, International Test Conference 2003 (ITC'03), 2003.

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

[34] Behrooz, P., "Computer Arithmetic: Algorithms and Hardware Designs," Oxford University Press, New York, 2000.

[35] 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, Volume 38, Issue 5, May 2003 Page(s):689 - 695.

[36] Grad, J., and Stine, J.E., "A Multi-Mode Low-Energy Binary Adder," Fortieth Asilomar Conference on Signals, Systems and Computers, Oct. 29 2006-Nov. 1 2006 pp. 2065-2068.

[37] Yan, S., Dongyu, Z., Minxuan, Z., and Shaoqing, L., "High Performance Low-Power Sparse-Tree Binary Adders," Solid-State and Integrated Circuit Technology, 2006. ICSICT '06. 8th International Conference on, vol, no, pp.1649-1651, 23-26 Oct 2006.

[38] Tyagi, A., "A reduced-area scheme for carry-select adders," IEEE Trans. Comput, 1993, 42, (10), pp. 1163170.

[39] Shivakumar, P., Keckler, S., Kistler, M., Burger, D., and Alvisi, L., "Modeling the effect of technology trends on the soft error rate of combinatorial logic," Proceedings of the International Conference on Dependable Systems and Networks, pp. 389–398, 2002.

[40] Kogge, P., and Stone, H.S., "A parallel algorithm for the efficient solution of a general class of recurrence equations," IEEE Trans. Comput, vol. C-22, pp. 786-793, Aug 1973.

[41] Vergos, H.T., Efstathiou, C., and Nikolos, D., "Diminished-one modulo 2^n+1 adder design," Computers, IEEE Transactions on, vol.51, no.12, pp. 1389-1399, Dec 2002.

[42] Vergos, H.T., Efstathiou, C., "Design of efficient modulo 2^n+1 multipliers," IET Comput. Digit. Tech, 2007, 1, (1), pp. 49-57.

VITA

Rajashekhar Reddy Modugu was born on January 19, 1987 in Bethavole, Andhra Pradesh, India. Raja completed his school education in Bharathi Vidya Mandir, Kodada, India. He did his intermediate education at Vignan Junior College, Vadlamudi, India. He completed his Bachelor of Technology (B. Tech) in Electronics and Communication Engineering from International Institute of Information Technology, Hyderabad, India in May 2008. He started his Master of Science program in Electrical and Computer Engineering at Missouri University of Science and Technology in August 2008. He graduated in December 2010. He has published several conference proceedings and IEEE Journal articles.