Design And Performance Analysis Of 32-bit Array Multiplier Using Optimized Carry Select Adder

P. Ram Sirisha, Dr. A. M. Prasad

Abstract— **In this paper, design of three different array multipliers by using three different Carry Select Adder logics are presented. Those three adders are - Conventional Carry Select Adder, Binary to Excess-1 converter (BEC) based Carry Select Adder and Optimized Carry Select Adder. Previously in the literature, multiplier was designed using carry look-ahead adder, ripple carry adder and Conventional carry select adder logic. In this work, the same multiplier is designed using Binary to Excess-1 converter (BEC) based Carry Select Adder, Optimized Carry Select Adder. All the redundant logic expressions present in Conventional Carry Select Adder are eliminated and a new logic formulation is designed by scheduling the carry select operation before the final-sum calculation which results in Optimized Carry Select Adder. The multipliers presented in this paper are all modeled using Verilog and simulated using ModelSim simulator. The three performance parameters i.e., Area, Delay and Power consumption for three multipliers are obtained using Xilinx 13.2 ISE Design Suite. The comparison of three performance parameters of three different multipliers is presented. The results analysis shows that the multiplier designed using Optimized carry select adder has better performance in terms of area, delay and power consumption.**

Index Terms— **Array Multiplier, Carry Select Adder (CSLA), Conventional Carry Select Adder, Binary to Excess-1 converter (BEC) based CSLA and Optimized CSLA, Verilog simulation.**

I. INTRODUCTION

Multipliers play an important role in various electronic applications like Digital signal processing, in which multipliers are used to perform various algorithms like FIR, IIR etc. In high performance systems such as microprocessor, DSP etc addition and multiplication of two binary numbers is fundamental and most often used arithmetic operations. Area efficient, low power and high performance VLSI systems are increasingly used in biomedical instrumentation [1],[2], multi-standard wireless receivers, portable and mobile devices.

The major challenge for VLSI designer is to reduce area of chip by using efficient optimization techniques. Speed of operation is one of the major constraints in designing DSP processors and today's general-purpose processors. Most of today's commercial electronic products are portable like Mobile, Laptops etc. that require more battery backup. So, there is a need to reduce power consumption. So, an efficient multiplier has to be designed to improve the performance of today's microprocessors and complex digital signal processing systems. An efficient multiplier should have characteristics like high accuracy, high speed, less area required and low power consumption. The basic principle used for multiplication is to evaluate partial products and accumulation of shifted partial products. In order to perform this operation number of successive addition operations is required. Therefore one of the major components required to design a multiplier is Adder. The adder can be ripple carry adder, carry look-ahead adder, carry select adder or any other adder[12][13]. However, using an efficient adder for the multiplier improves the overall performance of the multiplication operation.

V.Vijayalakshmi *et al*.[10] proposed a multiplier using carry look-ahead adder and carry select adder (CSLA) which involves higher delay and area. Hasan Krad and Aws Yousif Al-Taie [11] suggested multiplier using carry look-ahead adder and ripple carry adder, it has marginally higher delay. So, to improve the performance of a multiplier an efficient adder is required. By eliminating redundant logic operations present in Conventional CSLA [6] and sequencing the logic operations [9] results in Optimized CSLA. In this paper, a multiplier with better performance interms of area, delay and power is designed using Optimized CSLA.

II. ARRAY MULTIPLIER

The basic principle used for multiplication is to evaluate partial products and accumulation of shifted partial products. In multiplication, multiplicand is added to itself a number of times as specified by the multiplier to generate product. In case of multiplier with CSLA, all partial product additions as well as final addition is carried out by using carry select logic. Fig.1 shows the schematic of a 32- bit array multiplier. In Array multiplier, almost identical cells array is used for generation of the bit-products and accumulation. All bit-products are generated in parallel and collected through an array.

A. Multiplication Algorithm

Let the size of the product register be 64 bits and the multiplicand registers size be 32 bits. Store the multiplier in the least significant half of the product register. Clear the most significant half of the product register.

Repeat the following steps for 32 times:

1. If the least significant bit of the product register is "1" then add the multiplicand to the most significant half of the product register.

Manuscript received August, 2015.

***P. Ram Sirisha**, Electronics and Communication Engineering, University College Of Engineering Kakinada, JNTUK, Kakinada, India.*

***Dr. A. M. Prasad**, Professor and HOD, Electronics and Communication Engineering, University College Of Engineering Kakinada, JNTUK, Kakinada, India.*

ISSN: 2278 – 7798

2. Shift the content of the product register one bit to the right and the shifted-out bit has to be placed at the least significant half of the product register.
3. Shift-in the carry bit into the most significant bit of the product register.

Table I shows the complete multiplication process and gives an algorithm steps with the help of an example. In this algorithm, S(i) represents sum of each product term, B(i)A represents each product term and P(i) represents individual bit-term of final product [3].

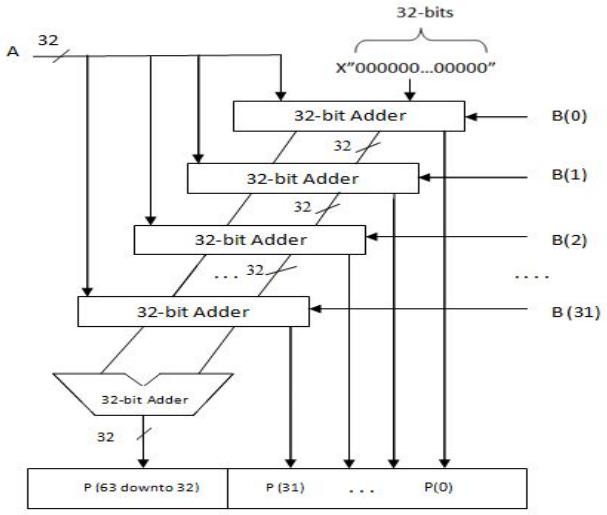

Fig.1. 32-bit Array Multiplier

Table I
Algorithm For 4-bit Multiplier Operation

A (Multiplier) B (Multiplicand)	1 0 0 1 0 1 1 1	Algorithm Steps
S (0) +B(0)A	0 0 0 0 1 0 0 1	Step-1
S(1) Shift right by one bit	1 0 0 1 0 1 0 0 |1 → P (0)	Step-2
S(1) +B(1)A	0 1 0 0 1 0 0 1	Step-3
S(2) Shift right by one bit	1 1 0 1 0 1 1 0 |1 → P (1)	Step-4
S(2) +B(2)A	0 1 1 0 1 0 0 1	Step-5
S(3) Shift right by one bit	1 1 1 1 0 1 1 1 |1 → P (2)	Step-6
S(3) +B(3)A	0 1 1 1 0 0 0 0	Step-7
S(4) Shift right by one bit	0 1 1 1 0 0 1 1 |1 → P (3)	Step-8
Final Product P (7:0)	0 0 1 1 1 1 1 1	Step-9

As noted earlier, in this paper, three different 32-bit array multipliers are designed using three different carry select adder logics i.e., Conventional CSLA logic, BEC based CSLA logic and optimized CSLA logic. These three different 32-bit carry select adders are used in place of 32-bit adder shown in Fig.1. That means to design an array multiplier with conventional CSLA the 32-bit adder shown in Fig.1 has to be replaced with 32-bit Conventional CSLA. In the same way to design an array multiplier with BEC based CSLA and optimized CSLA [9], the 32-bit adder shown in Fig.1 has to be replaced with 32-bit BEC based CSLA and optimized CSLA respectively. The Array Multiplier designed in this paper, multiplies two 32-bit unsigned integer values and gives a product term of 64-bit value. The three different carry select adder logics are presented in the following sections.

III. CARRY SELECT ADDER

The concept of CSLA is to compute alternative results in parallel and subsequently selecting the correct result with single or multiple stage hierarchical techniques. In CSLA both sum and carry bits are calculated for two alternatives C_{in}= 0 and 1. Once C_{in} is delivered, the correct computation is chosen using a multiplexer to produce the desired output. Instead of waiting for C_{in} to calculate the sum, the sum is correctly output as soon as C_{in} gets there. The time taken to compute the sum is then avoided which results in good improvement in speed.

The CSLA consists of two units: 1) the *sum* and *carry* generator unit (SCG) and 2) the *sum* and *carry* selection unit [4]. Most of the logic resources of CSLA are consumed by SCG unit. For efficient implementation of the SCG unit Different logic designs have been suggested. A study is made on the logic designs suggested for the SCG unit of conventional and BEC-based CSLAs of [5]. The main aim of this study is to identify data dependence and redundant logic operations. Accordingly, all redundant logic operations are removed and the logic operations are sequenced based on their data dependence.

The multipath carry propagation feature of the CSLA is fully exploited in the SQRT-CSLA [8], it is composed of a chain of CSLAs. CSLAs of increasing size are used in the SQRT-CSLA to extract the maximum concurrence in the carry propagation path. Using the SQRT-CSLA design, large-size adders are implemented with significantly less delay than a single-stage CSLA of same size. A 16-bit SQRT-CSLA design is shown in Fig. 2, where the 2-bit RCA, 2-bit CSLA, 3-bit CSLA, 4-bit CSLA, and 5-bit CSLA are used.

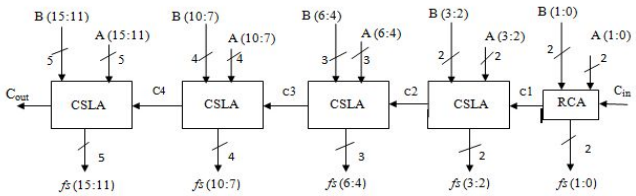

Fig.2. 16-bit Multistage CSLA

A. Conventional Carry Select Adder

A conventional carry select adder (CSLA) has an RCA–RCA structure that generates a pair of *sum* words and *output carry* bits corresponding the anticipated input-carry (c_{in} = 0 and 1) and selects one out of each pair for *final-sum* and *final-output-carry* [6]. A ripple carry adder (RCA) uses a simple design, but carry propagation delay (CPD) is the main concern in this adder. Carry look-ahead and carry select (CS) methods have been suggested to reduce the CPD of adders. A conventional CSLA has less CPD than an RCA, but the design is not attractive since it uses a dual RCA
The structure of Conventional CSLA and internal structure of RCA is shown in Fig. 3(a) and Fig. 3(b). The SCG unit of the

ISSN: 2278 – 7798

conventional CSLA [6] is composed of two *n*-bit RCAs, where *n* is the adder bit-width. The logic operation of the *n*-bit RCA is performed in four stages:
1) *half-sum* generation (HSG);
2) *half-carry* generation (HCG);
3) *full-sum* generation (FSG); and
4) *full-carry* generation (FCG)

If two *n*-bit operands are added in the conventional CSLA, then RCA-1 and RCA-2 generate two *n*-bit *sums* (s^0 and s^1) and two output-carry (and) corresponding to input-carry (c_{in}= 0 and c_{in}= 1), respectively. Logic expressions of RCA-1 and RCA-2 of the SCG unit of the *n*-bit CSLA are given below.

$$s_0^0(i) = A(i) \text{ xor } B(i) \qquad c_0^0(i) = A(i) \text{ and } B(i) \tag{1a}$$

$$s_1^0(i) = s_0^0(i) \text{ xor } c_1^0(i-1) \tag{1b}$$

$$c_1^0(i) = c_0^0(i) \text{ or } (s_0^0(i) \text{ and } c_1^0(i-1)) \qquad c_{out}^0 = c_1^0(n-1) \tag{1c}$$

$$s_0^1(i) = A(i) \text{ xor } B(i) \qquad c_0^1(i) = A(i) \text{ and } B(i) \tag{2a}$$

$$s_1^1(i) = s_0^1(i) \text{ xor } c_1^1(i-1) \tag{2b}$$

$$c_1^1(i) = c_0^1(i) \text{ or } (s_0^1(i) \text{ and } c_1^1(i-1)) \qquad c_{out}^1 = c_1^1(n-1) \tag{2c}$$

where $c_1^0(-1) = 0$, $c_1^1(-1) = 1$, and $0 \le i \le (n-1)$.

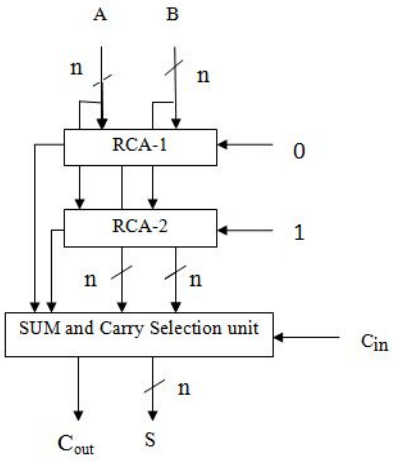

Fig. 3(a). *n* – bit Conventional CSLA

The logic expressions shown in (1a)–(1c) and (2a)–(2c) i.e.,{(i), (i)} and {(i), (i)} are identical. These redundant logic operations have to be removed to have an optimized design for RCA-2, in which the HSG and HCG of RCA-1is shared to construct RCA-2. Based on this, [7] and [8] have used an add-one circuit instead of RCA-2 in the CSLA, in which a BEC circuit is used in [8] for the same purpose.

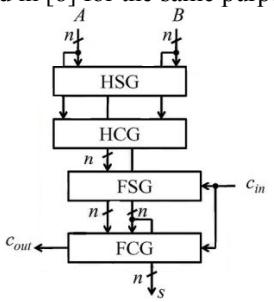

Fig. 3(b). Internal structure of RCA

B. Binary to Excess-1Based CSLA

The structure of BEC based CSLA is shown in Fig. 4, For = 0, the RCA calculates *n*-bit *sum* and . The BEC unit receives and from the RCA and generates (*n* + 1)-bit excess-1 code. The most significant bit (MSB) of BEC represents , in which *n* least significant bits (LSBs) represent . The logic expressions of the RCA are the same as those given in (1a)–(1c). The logic expressions of the BEC unit of the *n*-bit BEC-based CSLA are given as

$$s_1^1(0) = \overline{s_1^0(0)} \qquad c_1^1(0) = s_1^0(0) \tag{3a}$$

$$s_1^1(i) = s_1^0(i) \text{ xor } c_1^1(i-1) \tag{3b}$$

$$c_1^1(i) = s_1^0(i) \,.\, c_1^1(i-1) \tag{3c}$$

$$c_{out}^1 = c_1^0(n-1) \text{ xor } c_1^1(n-1) \tag{3d}$$

for $1 \le i \le (n-1)$.

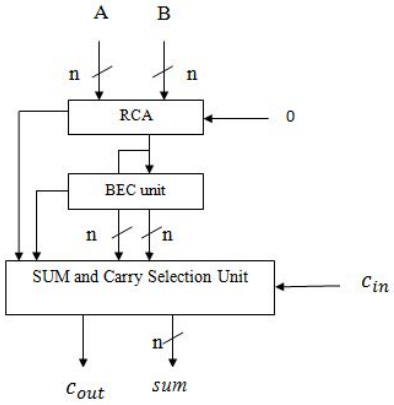

Fig. 4. BEC Based CSLA

From (1a)–(1c) and (3a)–(3d), it is clear that, in the case of the BEC-based CSLA, depends on , whereas in the case of the conventional CSLA there is no dependence on Therefore the BEC method increases data dependence in the CSLA.

C. Optimized CSLA

From (1a)–(1c) and (2a)–(2c), it is interesting to note that the logic expressions of and are identical except the terms and since (= =). and depend on{,, }, where = = . Since and have no dependence on and , the logic operation of and can be scheduled before and , and the select unit can select one from the set (,) for the *final-sum* of the CSLA. It is not an efficient approach to reject one sum-word after the calculation because significant amount of logic resource is spent for calculating (,). To solve this problem, the required carry word has to be selected from the anticipated carry words {and } to calculate the *final-sum*. The selected carry word is added with the *half-sum* () to generate the *final-sum* (*s*).

All the redundant logic operations of (1a)–(1c) and (2a)–(2c) are removed and logic expressions of (1a)–(1c) and (2a)–(2c) are rearranged based on their dependence which results in an area–delay and power-efficient design for the CSLA. The logic formulation for optimized CSLA is given as

ISSN: 2278 – 7798

$$s_0(i) = A(i) \text{ xor } B(i) \qquad c_0(i) = A(i) \text{ and } B(i) \tag{4a}$$

$$c_1^0(i) = ((c_1^0(i-1) \text{ and } s_0(i)) \text{ or } c_0(i)) \text{ for } \{c_1^0(-1) = 0\} \tag{4b}$$

$$c_1^1(i) = ((c_1^1(i-1) \text{ and } s_0(i)) \text{ or } c_0(i)) \text{ for } \{c_1^1(-1) = 1\} \tag{4c}$$

$$c(i) = c_1^0(i) \text{ if } (c_{in} = 0) \tag{4d}$$

$$c(i) = c_1^1(i) \text{ if } (c_{in} = 1) \tag{4e}$$

$$c_{out} = c(n-1) \tag{4f}$$

$$s(0) = s_0(0) \text{ xor } c_{in} \qquad s(i) = s_0(i) \text{ xor } c(i-1) \tag{4g}$$

The logic formulation for optimized CSLA is given (4a)–(4g), and its structure is shown in Fig. 5. It consists of one HSG unit, one FSG unit, one CG unit, and one CS unit. The CG unit is composed of two CGs (CG_0 and CG_1) corresponding to input-carry '0' and '1'.

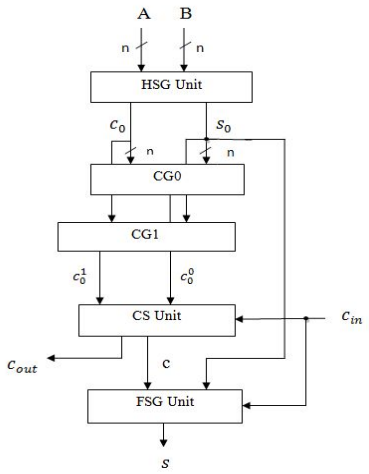

Fig. 5. Optimized CSLA

The HSG unit receives two *n*-bit operands (*A* and *B*) and generate *half-sum* word and *half-carry* word of width *n* bits each Both CG_0 and CG_1 receive and from the HSG unit and generate two *n*-bit full-carry words and corresponding to input-carry '0' and '1', respectively. The logic circuits of CG_0 and CG_1 are optimized to take advantage of the fixed input-carry bits.

Using the control signal c_{in}, the CS unit selects one final carry word from the two carry words available at its input line. It selects when $c_{in} = 0$; otherwise, it selects . The CS unit can be implemented using an *n*-bit 2-to-l MUX. However, from the truth table of the CS unit it is found that carry words and follow a specific bit pattern. If (*i*) = '1', then (*i*) = '1', irrespective of (*i*) and (*i*), for $0 \le i \le n-1$. This feature is used for logic optimization of the CS unit. The optimized design of the CS unit is shown in Fig. 6, which is composed of *n* AND–OR gates.

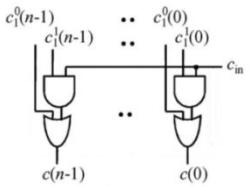

Fig. 6.Optimized design of CS Unit

The final carry word *c* is obtained from the CS unit. The MSB of *c* is sent to output as c_{out}, and $(n-1)$ LSBs are XORed with $(n-1)$ MSBs of *half-sum* (s_0) to obtain $(n-1)$ MSBs of *final-sum*(*s*). The LSB of s_0 is XORed with c_{in} to obtain the LSB of *s*.

IV. SIMULATION RESULTS

The Simulation results obtained for three different 32-bit array multipliers i.e., 32-bit Array multiplier designed using Conventional CSLA, BEC based CSLA and Optimized CSLA are shown in Fig. 7(a),7(b) and 7(c). For simulation Modelsim XE III 6.4b tool is used. The multipliers use 32-bit values as shown in simulation waveforms.

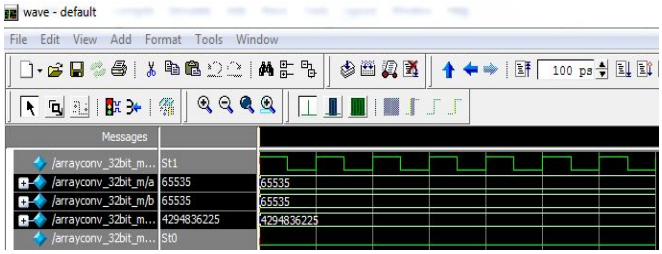

Fig.7(a). Simulation result of 32-bit array multiplier using Conventional CSLA

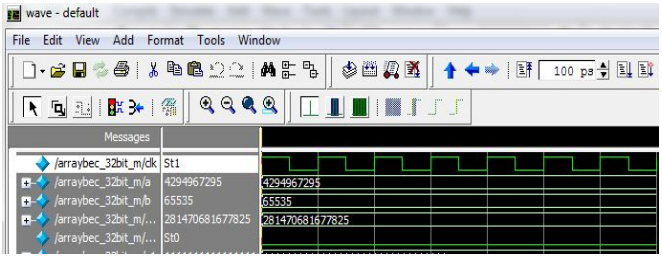

Fig. 7(b). Simulation result of 32-bit array multiplier using BEC based CSLA

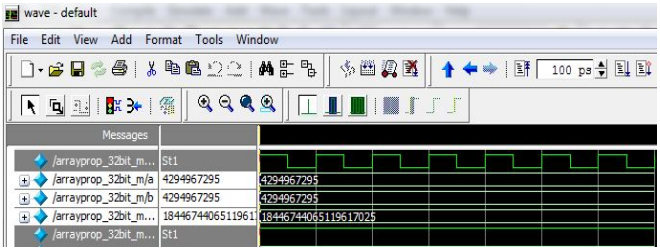

Fig. 7(c). Simulation result of 32-bit array multiplier using Optimized CSLA

IV. PERFORMANCE ANALYSIS

In this section, the results obtained from Synthesis and Simulation reports are presented. The objective of this project is to evaluate the performance of three Array multipliers (one by using Conventional CSLA, second by using BEC based CSLA and third by using Optimized CSLA on the basis of Area required, Speed of operation and power consumption. Table II, gives the area, delay and power consumption details of three multipliers.

ISSN: 2278 – 7798

Table II
Synthesis and simulation results for area,
Delay and power consumption of 32-bit array multiplier

32-bit Array Multiplier	Area		Delay (ns)	Power (mW)
	Slices	LUTs		
Conventional CSLA	2042	3949	94.147	447.81
BEC based CSLA	1941	3783	90.582	377.42
Optimized CSLA	1896	3680	72.412	271.88

A. Area Analysis

From Table II and figure 8(a), it is clear that the multiplier with Optimized CSLA requires less number of Slices and LUTs than the multiplier with Conventional CSLA and BEC based CSLA because its logic has no need to generate two sum words. The multiplier with optimized CSLA requires less area than the multiplier of [10].The area analysis of multipliers with Conventional CSLA, BEC based CSLA and Optimized CSLA are represented in chart shown in Fig. 8(a).

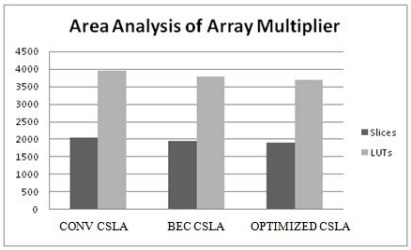

Fig. 8(a). Area Analysis

B. Delay Analysis

As shown in Table II and Fig. 8(b), the synthesis and simulation result shows that multiplier with Optimized CSLA takes less time to generate final product than the multiplier with Conventional CSLA and BEC based CSLA because the CS unit that produces output-carry before the FSG calculates the final-sum. The multiplier with Optimized CSLA has less delay than the multiplier of[10].The delay analysis of multipliers with Conventional CSLA, BEC based CSLA and Optimized CSLA are represented in chart shown in fig. 8(b).

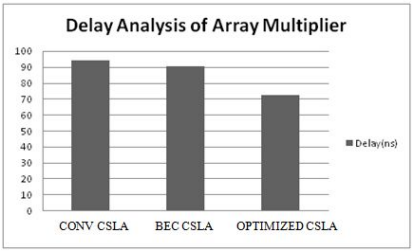

Figure-8(b). Delay Analysis

C. Power Analysis

Table II and Fig. 8(c) shows that there is an improvement in power consumption in case of multiplier using Optimized CSLA. The power analysis of multipliers with Conventional CSLA, BEC based CSLA and optimized CSLA are represented in chart shown in Fig. 8(c).

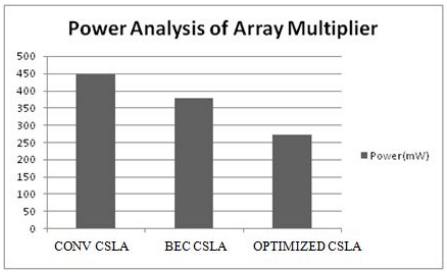

Fig. 8(c). Power Analysis

V. CONCLUSION

This paper presents three different multipliers that are modeled using Verilog. According to the results obtained, implementation of multiplier using Optimized CSLA logic improves overall performance of multiplier unit. Thus when compared to the multiplier using conventional CSLA, the multiplier with Optimized CSLA involves less area (slices reduced by 7.1% and LUTs reduced by 6.8%), less delay (reduced by 23%) and low power consumption (reduced by 39%). when compared to the multiplier using BEC based CSLA, the multiplier with Optimized CSLA involves less area (slices reduced by 2.3% and LUTs reduced by 2.7%), less delay (reduced by 20%) and low power consumption (reduced by 28%).

REFERENCES

[1] K. K. Parhi, *VLSI Digital Signal Processing*. New York, NY, USA:Wiley,1998.
[2] A.P. Chandrakasan, N. Verma, and D. C. Daly, "Ultralow-power electronicsfor biomedical applications," *Annu. Rev. Biomed. Eng.*, vol.10, pp. 247–274, Aug. 2008.
[3] T. Krishna Moorthy, K. Neelima, K.S. Chakradhar "*A Novel Design And Implementation Of Mtcmos Based Low Power D-Latch*" international journal of advanced scientific and technichal research. Issue 2 volume 4, august 2012-11-08.
[4] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*,2nd ed. New York, NY, USA: Oxford Univ. Press, 2010.
[5] B. Ramkumar and H.M. Kittur, "Low-power and area-efficient carry-selectadder," *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.*, vol. 20, no. 2, pp. 371–375, Feb. 2012.
[6] O. J. Bedrij, "Carry-select adder," *IRE Trans. Electron. Comput.*,vol. EC-11, no. 3, pp. 340–344, Jun. 1962.
[7] Y. Kim and L.-S. Kim, "64-bit carry-select adder with reduced area,"*Electron. Lett.*, vol. 37, no. 10, pp. 614–615, May 2001.
[8] Y. He, C. H. Chang, and J. Gu, "An area-efficient 64-bit square root carry select adder for low power application," in *Proc. IEEE Int. Symp. CircuitSyst.*, 2005, vol. 4, pp. 4082–4085.
[9] Basant Kumar Mohanty and Sujit Kumar Patel, "*Area- Delay- Power Efficient Carry Select Adder,"* IEEE Transactions on Circuit And systems, Vol. 61, No. 6, June 2014.
[10] V.Vijayalakshmi, R. Seshadri, DR.S. Ramakrishanan, *"Design and implementation of 32 bit unsigned multiplier using CLAA and CSLA,"*IEEE Proceedings on Cirucits, Devices and Systems, 2013.
[11] Hasan Krad and Aws Yousif Al, "Performance Analysis of a 32-Bit Multiplier with a Carry-Look-Ahead Adder and a 32-bit Multiplier with a Ripple Adder using VHDL," in Journal of Computer Science 4 (4): 305-308, 2008
[12] Wakerly, J.F., 2006. Digital Design-Principles and Practices. 4th Edn. Pearson Prentice Hall, USA.ISBN: 0131733494.

ISSN: 2278 – 7798

[13] William Stallings, 2006. Computer Organization and Architecture Designing for Performance. 7th Edn. Pearson Prentice Hall, USA. ISBN: 0-13-185644-8.

P. Ram Sirisha, received a B. Tech degree in the department of Electronics and Communication Engineering from Swarnandhra College Of Engineering And Technology. Currently pursuing M. Tech in Jawaharlal Nehru Technological University Kakinada in the department of Electronics and Communication Engineering.

Dr. A. M. Prasad, pursued Ph.D., M.E. Currently working as professor and HOD in the department of Electronics and Communication Engineering in Jawaharlal Nehru Technological University Kakinada. His areas of research includes Antennas and Instrumentation. He had published 20 papers in international Journals, 10 papers in National Journals.

ISSN: 2278 – 7798