International Journal of Engineering Research and Reviews ISSN 2348-697X (Online)
Vol. 4, Issue 1, pp: (9-18), Month: January - March 2016, Available at: www.researchpublish.com

Design of Fastest Multiplier Using Area–Delay–Power Efficient Carry-Select Adder

Mandala Sowjanya[1], N.G.N PRASAD[2], G.S.S Prasad[3]

Abstract: **Design of a high performance and high-density multiplier is presented. This multiplier is constructed by using the area, time and power efficient carry select adder. In previous we read about the designing of multipliers using the ripple carry adders. By using the ripple carry adders the propagation delay is high. To overcome this problem we are using the carry select adder in this paper. In the proposed scheme, the carry select (CS) operation is scheduled before the calculation of final-sum, which is different from the conventional approach. Bit patterns of two anticipating carry words (corresponding to c_{in} = 0 and 1) and fixed c_{in} bits are used for logic optimization of CS and generation units. An efficient CSLA design is obtained using optimized logic units. The proposed multiplier design involves significantly less area and delay than the recently proposed multipliers.**

Keywords: **Adder, arithmetic unit, low power design.**

I. INTRODUCTION

Carry Select adder designing involves ripple carry adder pairs which will work for summation either for Cin = 0 or and Cin = 1. Ripple carry adder is one of the adders which added three digits and reflects carry bit into next process. In conventional design two RCA's are used for summation for Cin = 0 and Cin =1 and final selection process will be carried out by multiplexer. Binary to Excess one converter is just the replacement of one RCA block for Cin = 1 in regular design because it gives added sum by considering Cin = 1. This paper describes the designing methodology of various fundamentals logic design simulation which helps in making the process running as in digital adder speed of addition is limited by propagation of carry through adder[1] [2] . Adders are an almost obligatory component of every contemporary integrated circuit. The prerequisite of the adder is that it is primarily fast and secondarily efficient in terms of power consumption and chip area.

This paper presents the choice of selecting the adder topologies. The adder topology used in designing carry select adder work are ripple carry adder, binary to excess one converter and D-latch. Addition is an indispensable operation for any digital system, DSP or control system. Adders are also very significant component in digital systems because of their widespread use in other basic digital operations such as subtraction, multiplication and division. Hence, for improving the performance of the digital adder would extensively advance the execution of binary operations inside a circuit compromised of such blocks[1]. Ripple carry adder is the simplest but slowest adders with n operand size in bits.

The carry-ripple adder is composed of any cascaded single-bit full-adders. As in initial conventional case each part of adder is composed of two carry ripple adders with cin_0 and cin_1, respectively. Through the multiplexer, we can select the correct output result according to the logic state of carry-in signal. The carry-select adder can compute faster because the current adder stage does not need to wait the previous stage's carryout signal. Next we replace this ripple carry adder with BEC (Binary to Excess-1) converter in second terminology. The CLSA is used in many computational systems to alleviate the problem of carry propagation delay by independently generating multiple carries and then select a carry to generate the sum [1]. However, the CSLA is not area efficient because it uses multiple pairs of Ripple Carry Adders (RCA) to generate partial sum and carry by considering carry input and then the final sum and carry are selected by the multiplexers (mux).

2. RELATED WORK

The "Urdhva Tiryagbhyam" Sutra [5-10] is a general multiplication formula applicable to all cases of multiplication. Urdhva" and „Tiryagbhyam" words are derived from Sanskrit literature. „Urdhva" means "Vertically" and „ Tiryagbhyam" means "crosswise". The multiplication of two 2-digit decimal numbers 21 and 32 is shown in Figure 1. The least significant digit 1 of multiplicand is multiplied vertically by least significant digit 2 of the multiplier, get their

Product 2 and set it down as the least significant part of the answer. Then 2 and 2, 1 and 3 are multiplied crosswise, add the two, get 7 as the sum and set it down as the middle part of the answer. Then 2 and 3 is multiplied vertically, get 6 as t heir product and put it down as the last the left hand most part of the answer.

So 21 × 32 = 276

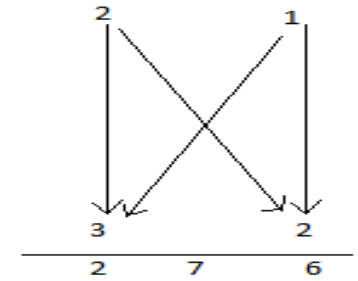


fig:1 multiplication of 21x32
by urdhva tiryalgorithm suitra

The „Urdhva Tiryagbhyam" algorithm can be implemented for binary number system in the same way as decimal number system. Let us consider the multiplication of two 2-bit binary numbers a1a0 and b1b0. Assuming that the result of this multiplication would be 4 bits, we express it as p2 p1 p0. The least significant bit a0 of multiplicand is multiplied vertically by least significant bit b0 of the multiplier, get their product p0 and set it down as the least significant part of the answer (p0). Then a1 and b0, and a0 and b1 are multiplied crosswise, add the two, get sum1 and carry1, the sum bit is the middle part of the answer (p1). Then a1 and b1 is multiplied vertically, and add with the previous carry (carry1) and get p2 (2 bit) as their product and put it downs as the left hand most part of the answer (p2). So a1a0 X b1b0=p2 p1 p0.Similarly The 2×2 Vedic multiplier module is then used to implement higher level multipliers (4×4 multiplier, 8×8 multiplier, 16×16 multiplier)

2.1 16 x 16 Vedic Multiplier Modules:

The architecture of 16x16 Vedic multiplier using „Urdhva Tiryagbhyam" Sutra is shown in Fig.2.The 16x16 Vedic multiplier architecture is implemented using four 8x8 Vedic multiplier modules, one 16 bit carry save adder, and two 17 bit binary adder stages

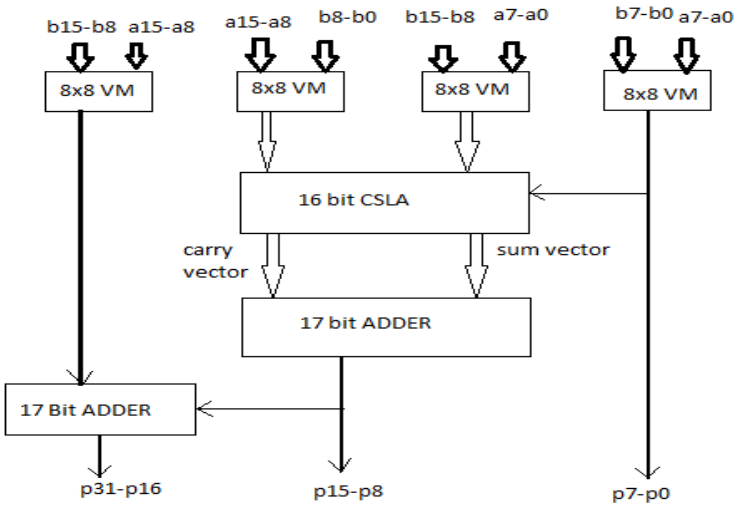


Fig 2: Architecture of 16x16 Vedic Multiplier Module Urdhva Tiryagbhyam

Multiplication Result = (p31-p16) & (p15-p8) & (p7-p0). Where & = concatenate operation. The proposed architecture uses 16-bit carry save adder and 17-bits adder modules to generate the final 32-bits product (p31-p16) & (p15-p8) & (p7-p0).The p7- p0 (8-bits) of the product represents least significant 8-bits of the 16-bit output of the right hand most 8x8 multiplier module. The 16-bit carry save adder adds three input 16-bit operands

i.e. concatenated 16-bit ("00000000" & most significant eight bits output of right hand most 8x8 multiplier module), each 16-bit output of second and third 8x8 multiplier modules. The 16-bit carry save adder produces two 16-bit outputs

Operands, sum vector and carry vector. The outputs of the carry save adder are fed into first 17-bit adder to generate 17-bit sum. The middle part (p15- p8) represents the least significant eight bits of 17-bit sum. The 16-bit output of the left most 8x8 multiplier module and concatenated 16-bits ("0000000"& the most significant nine bits of 17-bits sum) are fed into second 17-bit adder. The p31-p16 represents sixteen bit sum. The 33rd carry bit is omitted while taking the final product.

3. IMPLEMENTATION

The proposed CSLA is based on the logic formulation given in (4a)–(4g), and its structure is shown in Fig. 3(a). It consists of one HSG unit, one FSG unit, one CG unit, and one CS unit. The CG unit is composed of two CGs (CG0 and CG1) corresponding to input-carry '0' and '1'. The HSG receives two n-bit operands (A and B) and generate half-sum word s0 and half-carry word c0 of width n bits each. Both CG0 and CG1

Receive s0 and c0 from the HSG unit and generate two n-bit full-carry words c01 and c11 corresponding to input-carry '0' and '1', respectively. The logic diagram of the HSG unit is shown in Fig. 3(b). The logic circuits of CG0 and CG1 are optimized to take advantage of the fixed input-carry bits[5]. The optimized designs of CG0 and CG1 are shown in Fig. 3(c) and (d), respectively. The CS unit selects one final carry word from the two carry words available at its input line using the control signal cin. It selects c01 when cin = 0; otherwise, it selects c11. The CS unit can be implemented using an n-bit 2-to-l MUX.

However, we find from the truth table of the CS unit that carry words c01 and c11 follow a specific bit pattern. If c01 (i) = '1', then c11 (i) = 1, irrespective of s0(i) and c0(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. 3(e), which is composed of n AND–OR gates. The final carry word c is obtained from the CS unit. The MSB of c is sent to output as count, and (n −1) LSBs are XORed with (n − 1) MSBs of half-sum (s0) in the FSG [shown in Fig. 3(f)] to obtain (n − 1) MSBs of final-sum (s). The LSB of s0 is XORed with cin to obtain the LSB of s.

4. PROPOSED CS ADDER DESIGN

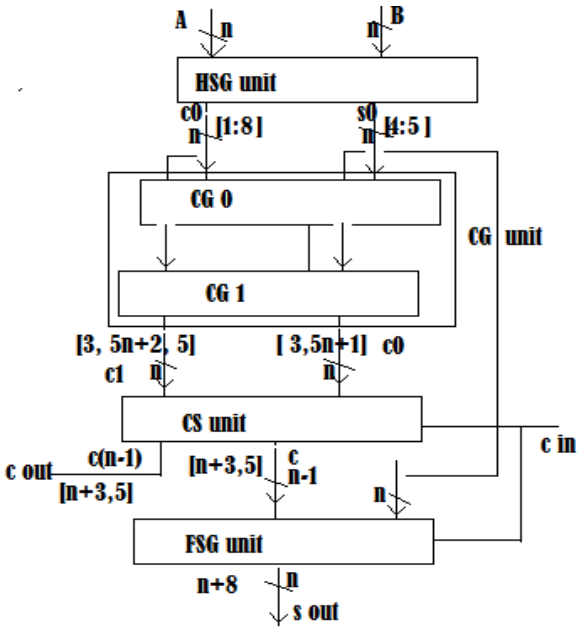
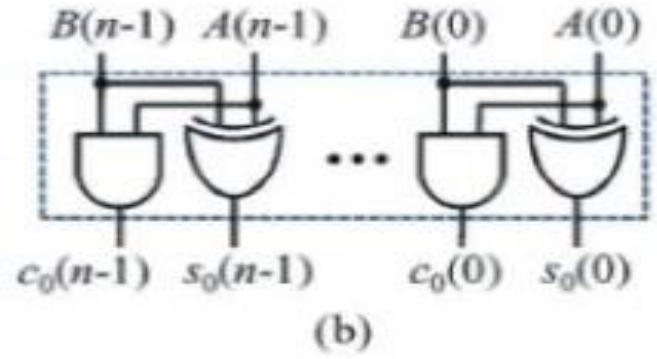
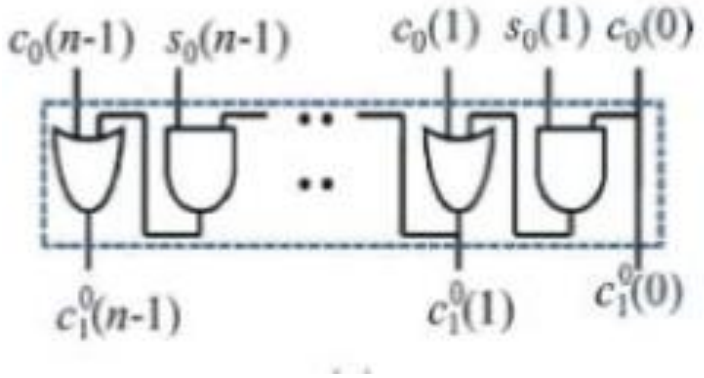


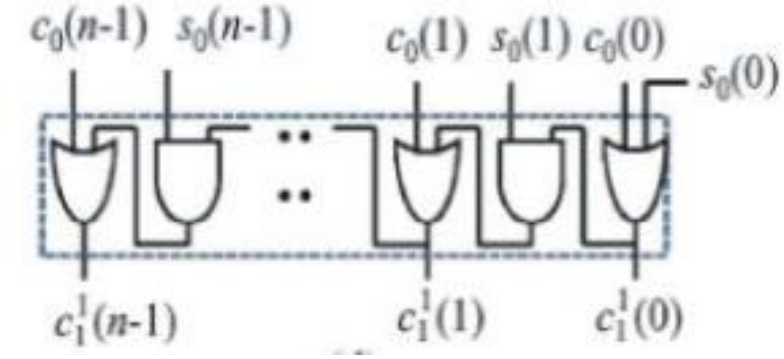
Fig 3 CSLA design



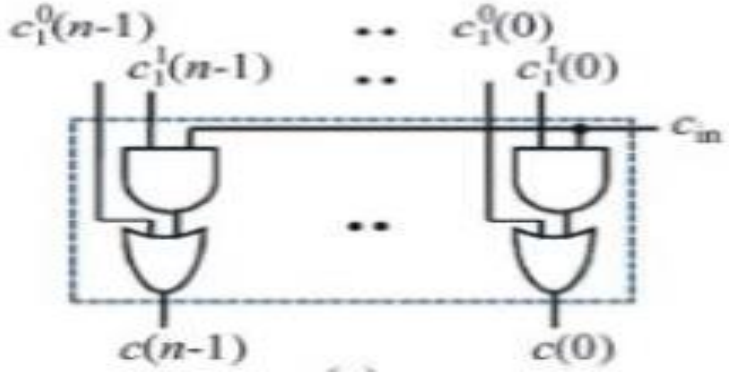
Gate level design of HSG



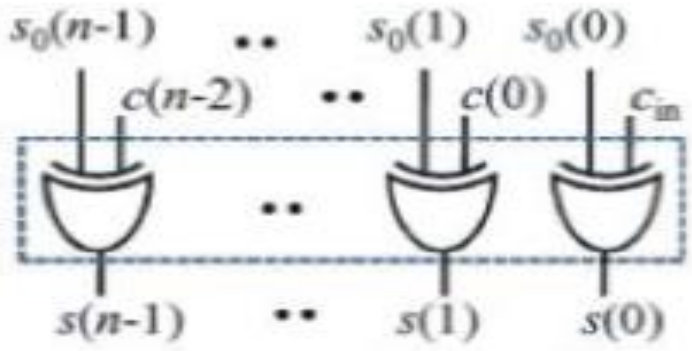
Gate level design of (CG0) input carry=0



Gate level design of (CG0) input carry=1



Gate level design of CS unit



Gate level design of FSG unit

In the proposed scheme, the CS operation is scheduled before the calculation of final-sum, which is different from the conventional approach. Carry words corresponding to input-carry '0' and '1'generated by the CSLA based on the proposed scheme follow a specific bit pattern, which is used for logic optimization of the CS unit. Fixed input bits of the CG unit are also used for logic optimization. Based on this, an optimized design for CS and CG units are obtained. Using these optimized logic units, an efficient design is obtained for the CSLA[6].

The proposed CSLA design involves significantly less area and delay than the recently proposed BEC-based CSLA. Due to the small carry output delay, the proposed CSLA design is a good candidate for the SQRT adder. The ASIC synthesis result shows that the existing BEC-based SQRT-CSLA design involves 48% more ADP and consumes 50% more energy than the proposed SQRTCSLA, on average, for different bit-widths[6].

Vedic multiplier:

A binary multiplier is an electronic circuit used in digital electronics, such as a computer, to multiply two binary numbers. It is built using binary adders. A variety of computer arithmetic techniques can be used to implement a digital multiplier. Most techniques involve computing a set of partial products, and then summing the partial products together. This process is similar to the method taught to primary schoolchildren for conducting long multiplication on base-10 integers, but has been modified here for application to a base-2 (binary) numeral system.

Until the late 1970s, most minicomputers did not have a multiply instruction, and so programmers used a "multiply routine"[1][2] which repeatedly shifts and accumulates partial results, often written using loop unwinding. Mainframe computers had multiply instructions, but they did the same sorts of shifts and adds as a "multiply routine". Early microprocessors also had no multiply instruction. Though the multiply instruction is usually associated with the 16-bit microprocessor generation,[3] at least two "enhanced" 8-bit micro have a multiply instruction: the Motorola 6809, introduced in 1978,[4][5] and the modern Atmel AVR 8bit microprocessors present in the AT Mega, ATTiny and ATX Mega microcontrollers. As more transistors per chip became available due to larger-scale integration, it became possible to put enough adders on a single chip to sum all the partial products at once, rather than reuse a single adder to handle each partial product one at a time. Because some common digital signal processing algorithms spend most of their time multiplying, digital signal processor designers sacrifice a lot of chip area in order to make the multiply as fast as possible; a single-cycle multiply accumulate unit often used up most of the chip area of early DSPs.

Multiplication basics:

The method taught in school for multiplying decimal numbers is based on calculating partial products, shifting them to the left and then adding them together. The most difficult part is to obtain the partial products, as that involves multiplying a long number by one digit (from 0 to 9):

```
    123
X 456
 =====
 738 (this is 123 x 6)
 615 (this is 123 x 5, shifted one position
        to the left)
+ 492 (this is 123 x 4, shifted two
```

Positions to the left)
=====
56088

A binary computer does exactly the same, but with binary numbers. In binary encoding each long number is multiplied by one digit (either 0 or 1), and that is much easier than in decimal, as the product by 0 or 1 is just 0 or the same number. Therefore, the multiplication of two binary numbers comes down to calculating partial products (which are 0 or the first number), shifting them left, and then adding them together (a binary addition, of course):

1011 (this is 11 in decimal)
======
0000 (this is 1011 x 0)
1011 (this is 1011 x 1, shifted one position to the left)
1011 (this is 1011 x 1, shifted two positions to the left)
+ 1011 (this is 1011 x 1, shifted three positions to the left)
=========
10011010 (this is 154 in decimal)

This is much simpler than in the decimal system, as there is no table of multiplication to remember: just shifts and adds. This method is mathematically correct and has the advantage that a small CPU may perform the multiplication by using the shift and add features of its arithmetic logic unit rather than a specialized circuit. The method is slow, however, as it involves many intermediate additions. These additions take a lot of time. Faster multipliers may be engineered in order to do fewer additions[8]; a modern processor can multiply two 64-bit numbers with 6 additions (rather than 64), and can do several steps in parallel. The second problem is that the basic school method handles the sign with a separate rule ("+ with + yields +", "+ with – the sign of the number in the number itself, usually in the two's complement representation. That forces the multiplication process to be adapted to handle two's complement numbers, and that complicates the process a bit more. Similarly, processors that use ones' complement, sign-and-magnitude, IEEE-754 or other binary representations require specific adjustments to the multiplication process.

2 x 2 Multiplier

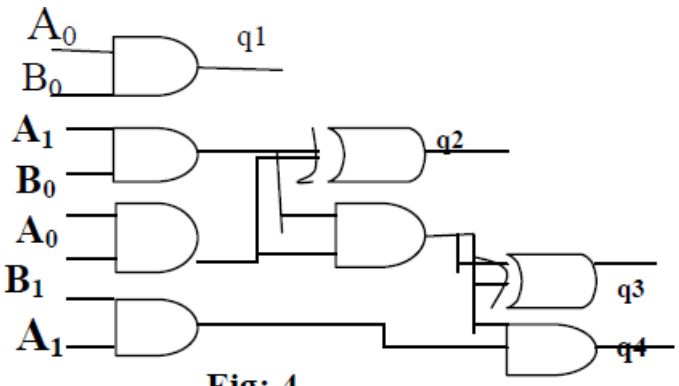


Fig: 4

4 X4 Multiplier

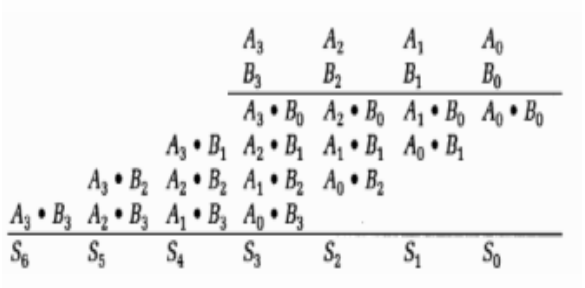
A binary 4×4 multiplier is a device that performs the 4-bit by 4-bit binary multiply algorithm. The following is an example of such algorithm:

Multiplicand 1011
Multiplier X 0110

0000
10110
101100
0000000

1000010

Note that in binary multiplication, the process involves shifting the multiplicand, and adding the shifted multiplicand or zero. Each bit of the multiplier determines whether a 0 is added or a shifter version of the multiplicand (0 implies zero added, 1 implies shifted multiplicand added). Thus, we can infer a basic shift-and-add algorithm to implement unsigned binary multiplication. In general, the logic is described as:

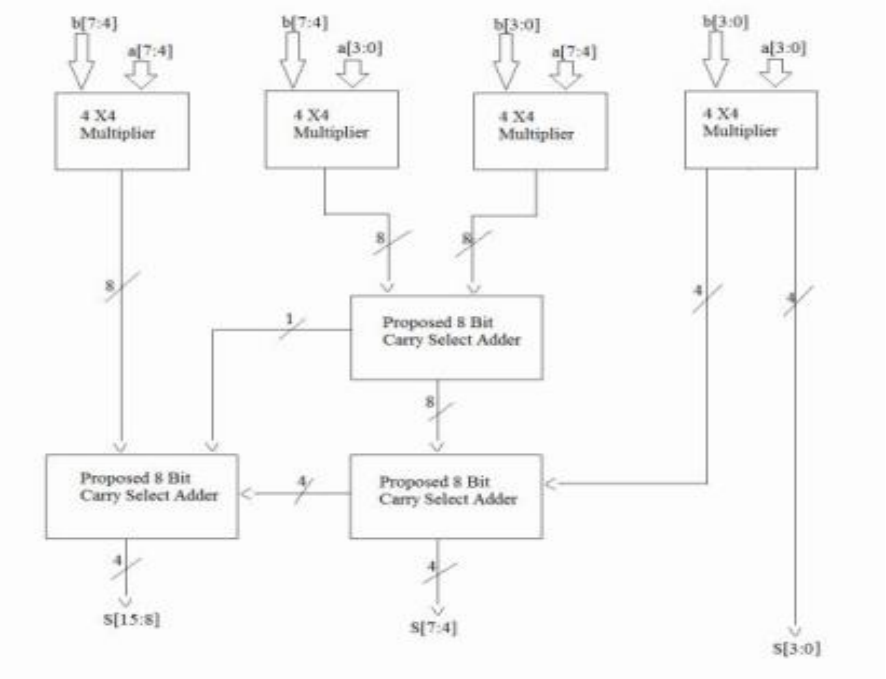


The 4x4 binary multiplier circuit has 8 inputs and 8 outputs.

8X8 Multiplier

The 8-bit multiplier is designed using four 4x4 multiplier. The output of these multipliers is added by modifying the logic levels of carry select adders. The 8-bit input sequence is divided into two 4-bit numbers and given as inputs to the 4-bit multiplier blocks (a[7:4] & b[7:4], a[3:0] & b[7:4], a[7:4] & b[3:0], a[3:0] & b[3:0]). The four multipliers used are similar and give 8-bit intermediate products which are added using overlapping logic with the help of three modified parallel adders (ADDER-1, ADDER-2 and ADDER- 3). The four LSB product bits P[3:0] are directly obtained from one of the multipliers. The output of the second and third multiplier block is added directly using ADDER-1 as the second and third region is overlapping. Then the higher order bit of first multiplier block is added to the overlapping sum using ADDER-2 which

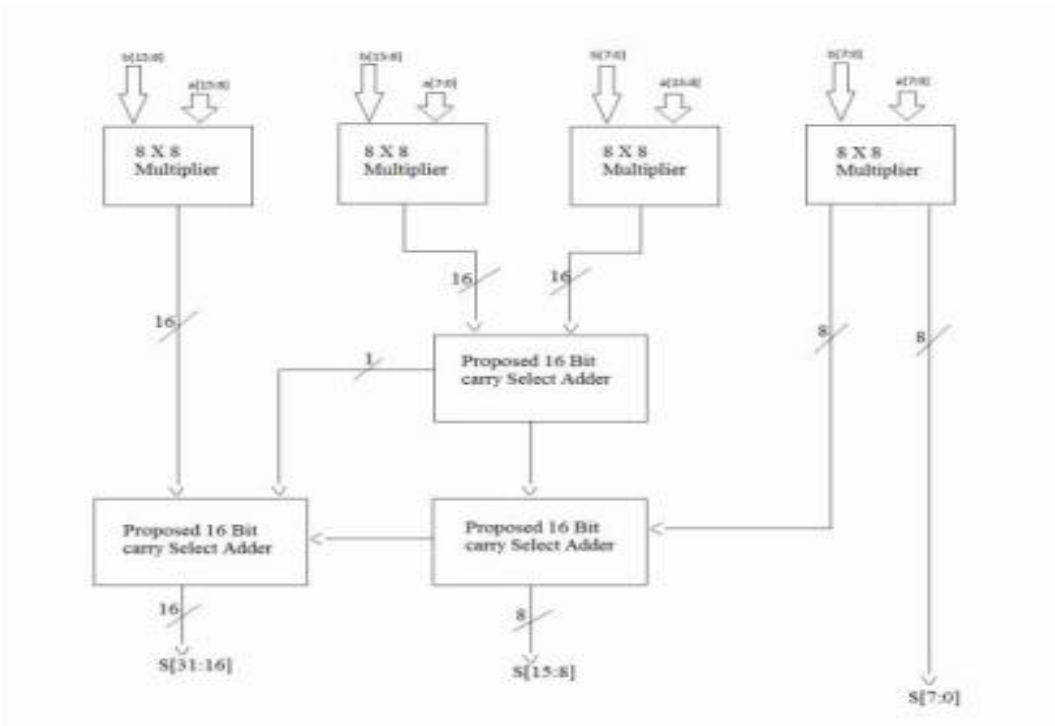
Gives the product P[7:4][10]. Finally, MSB bits P[15:8] are obtained by adding the fourth multiplier output.



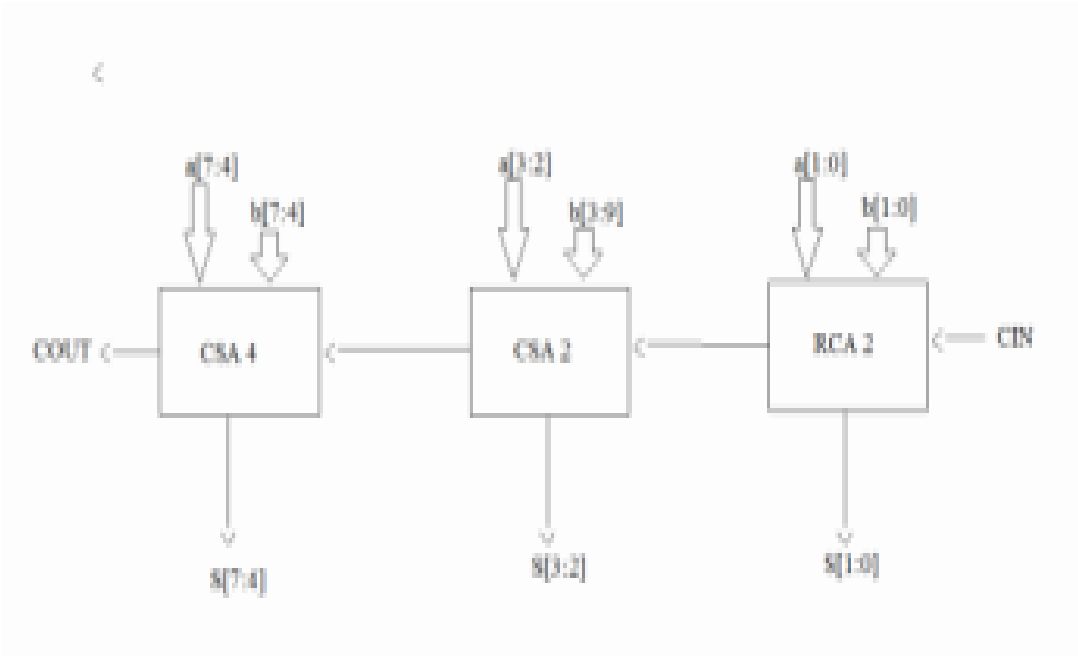
16*16 multiplier

The 16-bit multiplier is designed using four 8x8 multiplier. The output of these multipliers is added by modifying the logic levels of carry select adders. The 16-bit input sequence is divided into two 8-bit numbers and given as inputs to the 8-bit multiplier blocks (a[15:8] & b[15:8], a[7:0] & b[15:8], a[15:8] & b[7:0], a[7:0] & b[7:0]). The four multipliers used are similar and give 16-bit intermediate products which are added using overlapping logic with the help of three modified parallel adders (ADDER-1, ADDER-2 and ADDER-3)[11]. The four LSB product bits P[7:0] are directly obtained from

one of the multipliers. The output of the second and third multiplier block is added directly using ADDER-1 as the second and third region is overlapping. Then the higher order bit of first multiplier block is added to the overlapping sum using ADDER-2 which gives the product P[15:8]. Finally, MSB bits P[31:16] are obtained by adding the fourth multiplier output.



8bit 16 bit carry select adder

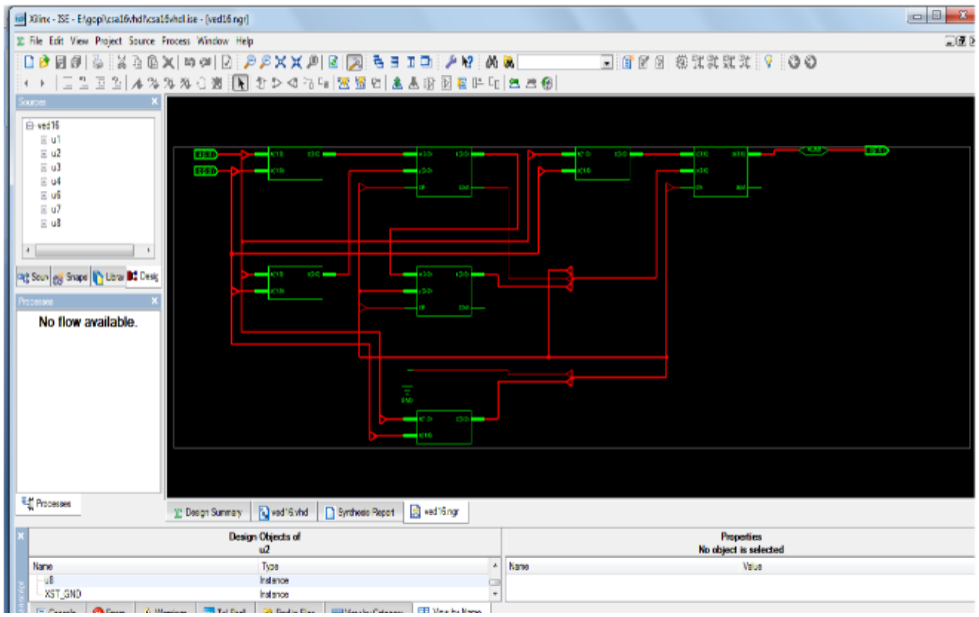


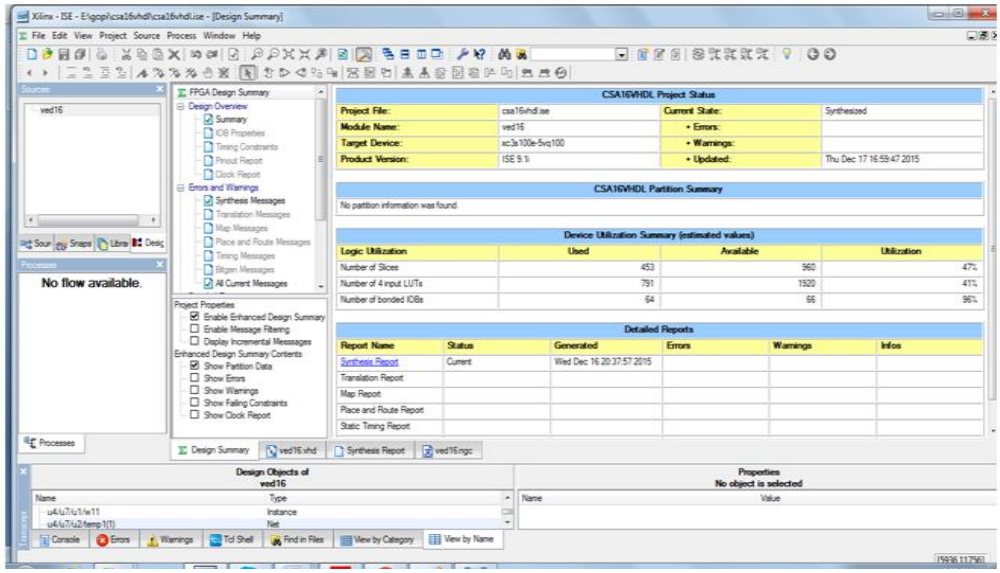
16 Bit SQRT CSLA

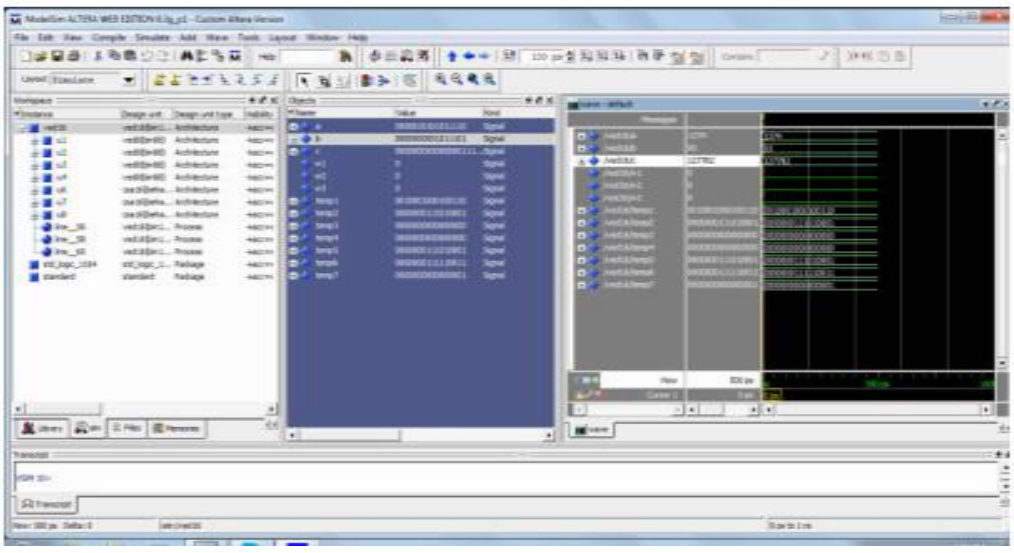
16-bit SQRT-CSLA design using the proposed CSLA is shown in Fig. 4, where the 2-bit RCA, 2-bit CSLA, 3-bit CSLA, 4-bit CSLA, and 5-bit CSLA are used. We have considered the cascaded configuration of (2-bit RCA and 2-, 3-, 4-, 6-, 7-, and 8-bit CSLAs) and (2-bit RCA and 2-, 3-, 4-, 6-, 7-, 8-, 9-, 11-, and 12-bit CSLAs), respectively, for the 32-bit SQRTCSLA and the 64-bit SQRT-CSLA to optimize adder delay.

5. EXPERIMENTAL RESULTS

RTL Schematic Digram







6. CONCLUSION

This paper presents a novel way of realizing a high speed multiplier using carry select adders instead of using ripple carry adders. The 4-bit and 8-bit modified multipliers are designed. The 16-bit multiplier is realized using four 8-bit multipliers and modified carry select adders. Though the number of gates used is fairly high, the increase in speed compensates for the increase in area. The proposed 16-bit multiplier gives a total delay of 39.362ns which is less when compared to the total delay of any other renowned multiplier architecture. Results also indicate a 13.65% increase in the speed when compared to normal multiplier with ripple carry adder. Our design outshines all other designs.

REFERENCES

[1] Swami Bharati Krishna Tirthaji Maharaja, "VedicMathematics", MotilalBanarsidass publishers, 1965.

[2] Rakshith T R and RakshithSaligram,"Design of High Speed Low Power Multiplier using Reversible logic: a Vedic Mathematical Approach",

[3] International Conference on Circuits, Power and Computing Technologies (ICCPCT-2013), ISBN: 978-1-4673-4922-2/13, pp.775-781.

[4] M.E. Paramasivam and Dr. R.S.Sabeenian, "An Efficient Bit Reduction Binary Multiplication Algorithm using Vedic Methods",IEEE 2nd International Advance Computing Conference, 2010, ISBN: 978-1-4244-4791-6/10, pp. 25-28.

[5] 5. Sushma R. Huddar, Sudhir Rao Rupanagudi, Kalpana M and Surabhi Mohan, "Novel High Speed Vedic Mathematics Multiplier using Compressors", International Multi conference on Automation, Computing, Communication, control and Compressed Sensing(iMac4s),22-23 March 2013, Kottayam, ISBN:978-1-4673-5090-7/13, pp.465-469.

[6] L. Sriraman and T. N. Prabakar, "Design and Implementation of Two Variables Multiplier Using KCM and Vedic Mathematics", 1st International Conference on Recent Advances in Information Technology (RAIT -2012), ISBN: 978-1-4577-0697-4/12.

[7] K. K. Parhi, VLSI Digital Signal Processing. New York, NY, USA: Wiley, 1998.

[8] .A. P. Chandrakasan, N. Verma, andD. C. Daly, "Ultralow-power electronics for biomedical applications," Annu. Rev. Biomed.Eng., vol. 10, pp. 247– 274, Aug. 2008.

[9] O. J. Bedrij, "Carry-select adder, IRE Trans. Electron. Compute. Vol.EC-11, no. 3, pp. 340–344, Jun.1962.

[10] 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.

[11] 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. Circuits Syst., 2005, vol. 4, pp. 4082–4085.

[12] B. Ramkumar and H.M. Kittur, "Low-power and area-efficient carry-select adder," IEEE Trans.Very Large Scale Integr. (VLSI) Syst., vol. 20, no. 2, pp. 371–375, Feb. 2012.