Journal of AL-Qadisiyah for computer science and mathematics Vol.10 No.1 Year 2018
ISSN (Print): 2074 – 0204 ISSN (Online): 2521 – 3504

Comp Page 88 - 99 Muthana .Y

Design and Implementation of Efficient and High-Speed Multiplication Circuits Based on Vedic Algorithms

Muthana Yaseen Nawaf Isawi

University of Kirkuk
Department of Computer Science, College of Science
Muthana2085@yahoo.com

Recived : 13\11\2017 **Revised : 18\12\2017** **Accepted : 21\12\2017**

Available online : 26/1/2018

DOI:10.29304/jqcm.2018.10.1.354

Abstract. The increasing speed of computer processors with each passing day has required the design of arithmetic circuits to be verified as high performance. For this reason; by being observed the computer arithmetic, it enabled faster algorithms to come out and verifications of hardware in terms of the facilities that technology provides. The main aim of the computer arithmetic is the design of the circuits and algorithms that will increase the speed of the numerical process. To this end, the design of arithmetic multiplication circuits with a faster and higher bit length is presented through the efficient bit reduction method in this paper. The developed fast and efficient algorithms for arithmetic multiplication process by using the efficient bit reduction method have been observed in this work. By making changes in some multiplication methods that are based on Vedic math's, the higher bit length circuits of multiplication circuits in the literature which are 4 bits have been developed by using some basic properties of multiplication like decomposition and bit shifting. Analysis of arithmetic circuits is implemented by verifying functionally with VHDL simulations, getting output signal waveform and measurements of delay time. All the circuits of hardware that are observed have been described via VHDL and the performances of multiplication circuits that are synthesized have been presented via FPGA.

Key Words. Vedic algorithms, bit reduction, digital multiplier, VHDL, FPGA.

1.Introduction. To speak of today's engineering world, multiplication-based operations are some of the most commonly used functions and have recently been used in many Digital Signal Processing (DSP) applications such as Convolution, Fast Fourier Transform, Filtering and in the Arithmetic Logic Unit (ALU) of microprocessors [1, 2]. The most commonly used process is the multiplication process speed and designing the low-power multiplication circuit has been the focus of attention in recent years [3]. To minimize power consumption and delay in digital systems are required optimization at every stage of the design. This optimization means choosing the best algorithm for the situation, which means the highest level of design, the topology and finally the technology used in the implementation of the digital circuits. Based on these components, different types of available multiplication circuits are designed [4, 5].

The use of multiplication methods has been documented in the civilizations of Egypt, Babylon, India, and China [3]. In the early days of the advent of computers, multiplication was generally applied through a series of operations (addition, subtraction, and shifting). There are many algorithms proposed in the literature to perform the multiplication process, each offering different advantages and performing differently in terms of delay, circuit complexity, chip area and power consumption [5, 6]. The structure of the multipliers is generally divided into three categories. The first is a serial multiplier that focuses on the hardware and uses as minimum chips as possible. Second is parallel multipliers (tree and array) that perform mathematical operations at high speed. But the disadvantage of these multipliers is that they use a larger chip area. The third is the serial-parallel multiplier, which stands as a good alternative between the serial multiplier that takes a long time and the parallel multiplier that takes a large chip area [1, 5]. This paper presents a high-speed efficient multiplier implementation based on Vedic multiplication algorithms (Urdhva Tiryakbhyam Sutra and Nikhilam Sutra). In addition, various algorithms for arithmetic multiplication using efficient bit reduction method have been investigated. The commonly used Vedic multiplication algorithm and classical Booth multiplication algorithm have been chosen as arithmetic multiplication operations. However, in order to understand the working logic of the algorithms, the basic principles of multiplication algorithms, hardware implementation circuits and performance properties are given. However, the proposed Vedic algorithm we have developed based on Vedic mathematics is presented in detail.

2. Vedic Algorithms and Booth Multiplier

2.1 Vedic Mathematics: Vedic mathematics is part of the four Veda “wisdom books”. It makes explanations about some mathematical terms such as geometry, trigonometry, arithmetic, quadratic equations, factorization and even calculus [7].

Vedic mathematics is basically composed of 16 Sutra, which deals with the branches of mathematics such as arithmetic, algebra, geometry.

These methods can be applied directly to geometry, trigonometry, differential, integral, conics, and applied mathematics of various types. Since Vedic formulas (Sutra) are claimed to be based on the natural principles working conditions of the human mind, they offer a very interesting field and some efficient algorithms that can be applied to various branches of engineering such as programming and digital signal processing [7, 8].

2.2 Urdhva Tiryakbhyam Sutra: The multiplier is based on the Urdhva Tiryakbhyam algorithm of the ancient Indian Vedic mathematics. Urdhva Tiryakbhyam Sutra is a general form that can be applied to all cases of multiplication such as binary, hex, decimal and octal. The word means "Vertical & Crosswise" [7]. It is based on a new idea that helps to produce all the partial products and then to make the simultaneous additions of these partial results. Thus, the partial products and the parallelism in the production of their summaries can be achieved using Urdhava Tiryakbhyam. Since the partial results and their summations are calculated in parallel, the multiplier is independent of the clock frequency of the processor. On this count, the multiplier will need the same time to calculate the result, so it will be independent of the clock frequency [7, 9].
The main advantage is the reduction of the need for microprocessors to manage increasingly rising clock times. While a higher clock frequency usually results in an increased operating power, the disadvantage is that it increases the power dissipation which causes the device management to increase in temperature. The advantage of the multiplier is that as the number of bits increases, the gate delay and area increase more slowly than the other multipliers. Therefore, it is efficient in terms of time, space and power [8, 10].

Now we will see how this algorithm is used with binary numbers. An example (1101 * 1010) is given in Table 1.

Journal of AL-Qadisiyah for computer science and mathematics Vol.10 No.1 Year 2018
ISSN (Print): 2074 – 0204 ISSN (Online): 2521 – 3504

Table 1: Using Urdhva Tiryakbham for binary numbers

X = 1101, Y = 1010 $x_3\ x_2\ x_1\ x_0$ **Multiplicand** $y_3\ y_2\ y_1\ y_0$ **Multiplier** P6 P5 P4 P3 P2 P1 P0 $p_0 = x_0y_0$; $c_1p_1 = x_1y_0 + x_0y_1$; $c_2p_2 = c_1 + x_2y_0 + x_1y_1 + x_0y_2$; $c_3p_3 = c_2 + x_3y_0 + x_2y_1 + x_1y_2 + x_0y_3$; $c_4p_4 = c_3 + x_3y_1 + x_2y_2 + x_1y_3$; $c_5p_5 = c_4 + x_3y_2 + x_2y_3$; $c_6p_6 = c_5 + x_3y_3$	**METHODOLOGY** 1 1 0 1 / 1 0 1 0 — P0 1 1 0 1 / 1 0 1 0 — P1 1 1 0 1 / 1 0 1 0 — P2 1 1 0 1 / 1 0 1 0 — P3 1 1 0 1 / 1 0 1 0 — P4 1 1 0 1 / 1 0 1 0 — P5 1 1 0 1 / 1 0 1 0 — P6

2.3 Nikhilam Sutra: It means "all from 9 and last from 10". Basically, starting from the leftmost digits, each number is subtracted from 9 and the last number is subtracted from 10 [2, 7]. The Sutra algorithm is based on two different methods of multiplying numbers. The first is to find the nearest base of two numbers in multiplication., and the second is the subtraction method. Although the Nikhilam Sutra is applicable to all multiplication operations, it is essentially effective when the numbers are large and the complexity of the multiplication process is less [11]. We will illustrate Sutra by taking the multiplication of two decimal numbers (89 * 92) is show in Figure 1.

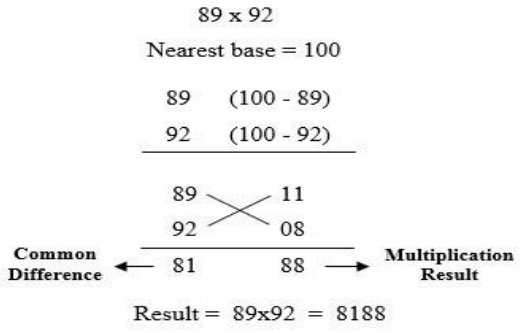


Figure 1: Multiplication using Nikhilam Sutra

2.4 Proposed Vedic Algorithm: In the binary arithmetic, a new reduced bit multiplication algorithm has proposed by modified the Nikhilam Sutra algorithm using some basic features such as decomposition and bit shifting. Based on the proposed algorithm, a 4x4-bit multiplication operation can be reduced to a single 2x2-bit multiplication operation. As a result, this algorithm reduces the delay for carry propagation more than any 4x4 bit multiplication. In the 4-bit proposed multiplication algorithm [4], it can be extended for larger numbers with some changes depending on the steps in the algorithm. The algorithm of proposed Vedic multiplier for the multiplication of two 8-bit numbers is given below.

(a) Initialization

Initialize: flag1 = flag2 = flag3 = flag4 = flag5 = flag6 = flag7 = flag8 = flag9 = flag10 = flag11 = flag12 = flag13 = 0

(b) Preprocessing

Input 8-bit binary numbers a and b

n1 = Number of least significant consecutive zeros in a

n2 = Number of least significant consecutive zeros in b

n = n1 + n2

a'= Right shift a by n1

b'= Right shift b by n2

(c) Processing

1. IF (a' > 128 & b' > 128) THEN
   a'=255-a'; b=255-b'; flag1=1;
2. IF (a' > 64 & b' > 128) THEN
   b'=b'-128; flag2=1;
   [IF (b' > 64 & a' > 128) THEN a'=a'-128;]
3. IF (a' > 64 & b' > 64) THEN
   a'=128-a'; b=128-b'; flag3=1;
4. IF (a' > 32 & b' > 64) THEN
   b'=b'-64; flag4=1;
   [IF (b' > 32 & a' > 64) THEN a'=a'-64;]
5. IF (a' > 32 & b' > 32) THEN
   a'=64-a'; b'=64-b'; flag5=1;
6. IF (a' > 16 & b' > 32) THEN
   b'=b'-32; flag6=1;
   [IF (b' > 16 & a' > 32) THEN a'=a'-32;]
7. IF (a' > 16 & b' > 16) THEN
   a'=32-a'; b=32-b'; flag7=1;
8. IF (a' > 08 & b' > 16) THEN
   b'=b'-16; flag8=1;
   [IF (b' > 08 & a' > 16) THEN a'=a'-16;]
9. IF (a' > 08 & b' > 08) THEN
   a'=16-a'; b=16-b'; flag9=1;
10. IF (a' > 04 & b' > 08) THEN
    b'=b'-08; flag10=1;
    [IF (b' > 04 & a' > 08) THEN a'=a'-08;]
11. IF (a' > 04 & b' > 04) THEN
    a'=08-a'; b=08-b'; flag11=1;
12. IF (a' > 02 & b' > 04) THEN
    b'=b'-04; flag12=1;
    [IF (b' > 02 & a' > 04) THEN a'=a'-04;]
13. IF (a' > 02 & b' > 02) THEN
    a'=04-a'; b=04-b'; flag13=1;
14. IF (a'=01) THEN p'=b' | IF (b'=01) THEN
    p'=a'
    GOTO Step 16
15. Perform 4bit multiplication: p'=a'*b';
16. IF (flag13= 1) THEN
    p'= [LHS=04-(a'+b')+ Carry of RHS] | [RHS=(2 bit)p'];
17. IF (flag12= 1) THEN p'=a'*04+(a'*b');
18. IF (flag11= 1) THEN
    p'= [LHS=08-(a'+b')+ Carry of RHS] | [RHS=(3 bit)p'];
19. IF (flag10= 1) THEN p'=a'*08+(a'*b');
20. IF (flag9= 1) THEN
    p'= [LHS=16-(a'+b')+ Carry of RHS] | [RHS=(4 bit)p'];
21. IF (flag8= 1) THEN p'=a'*16+(a'*b');
22. IF (flag7= 1) THEN
    p'= [LHS=32-(a'+b')+ Carry of RHS] | [RHS=(5 bit)p'];
23. IF (flag6= 1) THEN p'=a'*32+(a'*b');
24. IF (flag5= 1) THEN p'= [LHS=64-(a'+b')+ Carry of RHS] | [RHS=(6 bit)p'];
25. IF (flag4= 1) THEN p'=a'*64+(a'*b');
26. IF (flag3= 1) THEN p'= [LHS=128-(a'+b')+ Carry of RHS] | [RHS=(7 bit)p'];
27. IF (flag2= 1) THEN p'=a'*128+(a'*b');
28. IF (flag1= 1) THEN p'= [LHS=255-(a'+b')+ Carry of RHS] | [RHS=(8 bit)p'];
29. p = Left shift p' by n bits
30. Return the product p
31. **End**

In the preprocessing stage, the multiplier and the multiplied binary numbers are shifted directly to the right to remove the least significant consecutive zero bits. This reduces the calculation time by reducing the number of multiplier and multiplicand bits. The effect of the raised zero bits is combined more efficiently by shifting the last output to the left with the bits in the equal number.

To illustrate this multiplication table, consider the multiplication of two binary numbers (11000000 * 11000000).

a = $(1100\ 0000)_2$ = $(C0)_{16}$ = $(192)_{10}$

b = $(1100\ 0000)_2$ = $(C0)_{16}$ = $(192)_{10}$

Preprocessing:

a' = 11 n1 = 6; shift a to the right by 6 bit because the number zero is six;
b' = 11 n2 = 6; shift a to the right by 6 bit because the number zero is six;
n = n1 + n2 = 12

Processing:

Since number a 'and b' is greater than $(10)_2$, step 13 is taking place.

IF (a' > 02 & b' > 02) THEN

a' = 100 – a'; b' = 100 – b';
a' = 0100 - 0011; a' = 0001;
b' = 0100 - 0011; b' = 0001;

p' = a' * b';
p' = 0001 * 0001; p'(RHS) = 0001;

IF (flag13 = 1) THEN p'= [LHS=04-(a'+b')+ Carry of RHS] | [RHS=(2 bit)p'];

(a'+b') = 0001+0001 = 0010
Co of RHS= 0
p' = [0100 – 0010 + 0] | [01]
p' = [10] | [01]

p' = 1001 Now we will shift the final result to the left according to n.
n = 12
p = 1001 0000 0000 0000 = $(36864)_{10}$

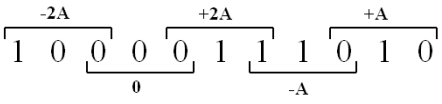
2.5 Booth Multiplication Algorithm: The Booth multiplication algorithm is a very efficient multiplication in signed numbers. The Booth algorithm is a method that reduces the numbers of generated partial products [1, 3]. The Booth algorithm is based on the fact that the multiplicative number presented in a certain range is converted to a higher base number and the number of digits is reduced [12]. In the Booth algorithm, the three-bit parts of the multiplier are scanned and the operations corresponding to the values of these parts are performed. This reduces the summation time and accelerates the multiplication process [3, 12]. Table 2 shows the 4-base (3 bit scans) Booth Recording process. The multiplier number encoded by the Booth Recording process is formulated below.

$$Z_i = -2X_{i+1} + X_i + X_{i-1}$$

Table 2: BOOTH-4 Recording Process

X_{i+1}	X_i	X_{i-1}	Z_i
0	0	0	+0A
0	0	1	+A
0	1	0	+A
0	1	1	+2A
1	0	0	-2A
1	0	1	-A
1	1	0	-A
1	1	1	+0A

As an example, 1000011101 is taken as a multiplier. The coding obtained by selecting 3 digits is as follows. where the start bit $X_{-1} = 0$ is taken.



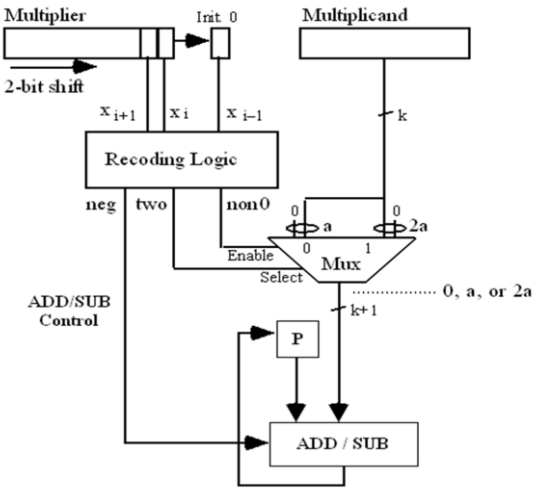


Figure 2: Hardware structure of the 4-base Booth algorithm

3. The Multipliers Implementation

3.1 (8x8) bit Urdhva Tiryakbhyam Multiplier: The 8x8 bit multiplication is generated by using four 4x4 bit multiplier blocks. Just as in the case of a 4x4 multiplication block, the numbers a and b are divided into smaller pieces at n / 2 = 4 bits long. These newly formed 4-bit pieces are inserted as input into the 4x4 multiplier block, where again these new pieces are divided into smaller pieces of n / 4 = 2 bits long and added to the 2x2 multiplication block. The result that produced from the output of the 4x4 multiplication block is sent to an addition tree for addition as shown in Figure 3 [6, 11].

3.2 (8x8) bit Proposed Vedic Multiplier:
The general architecture structure of the proposed Vedic multiplication circuit for the 8-bit multiplication is shown in Figure 4.

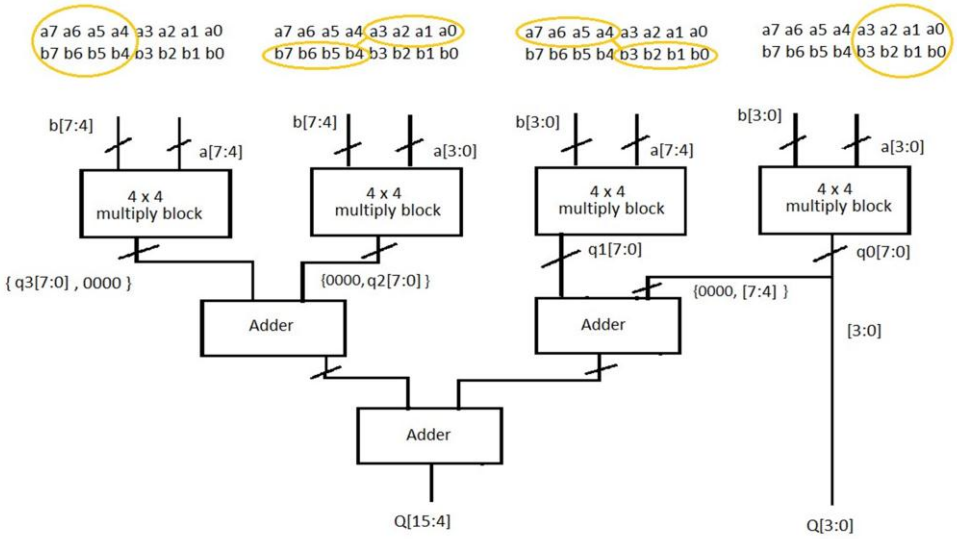


Figure 3: Block Diagram of 8x8 Urdhva Multiply block

Journal of AL-Qadisiyah for computer science and mathematics Vol.10 No.1 Year 2018
ISSN (Print): 2074 – 0204 ISSN (Online): 2521 – 3504

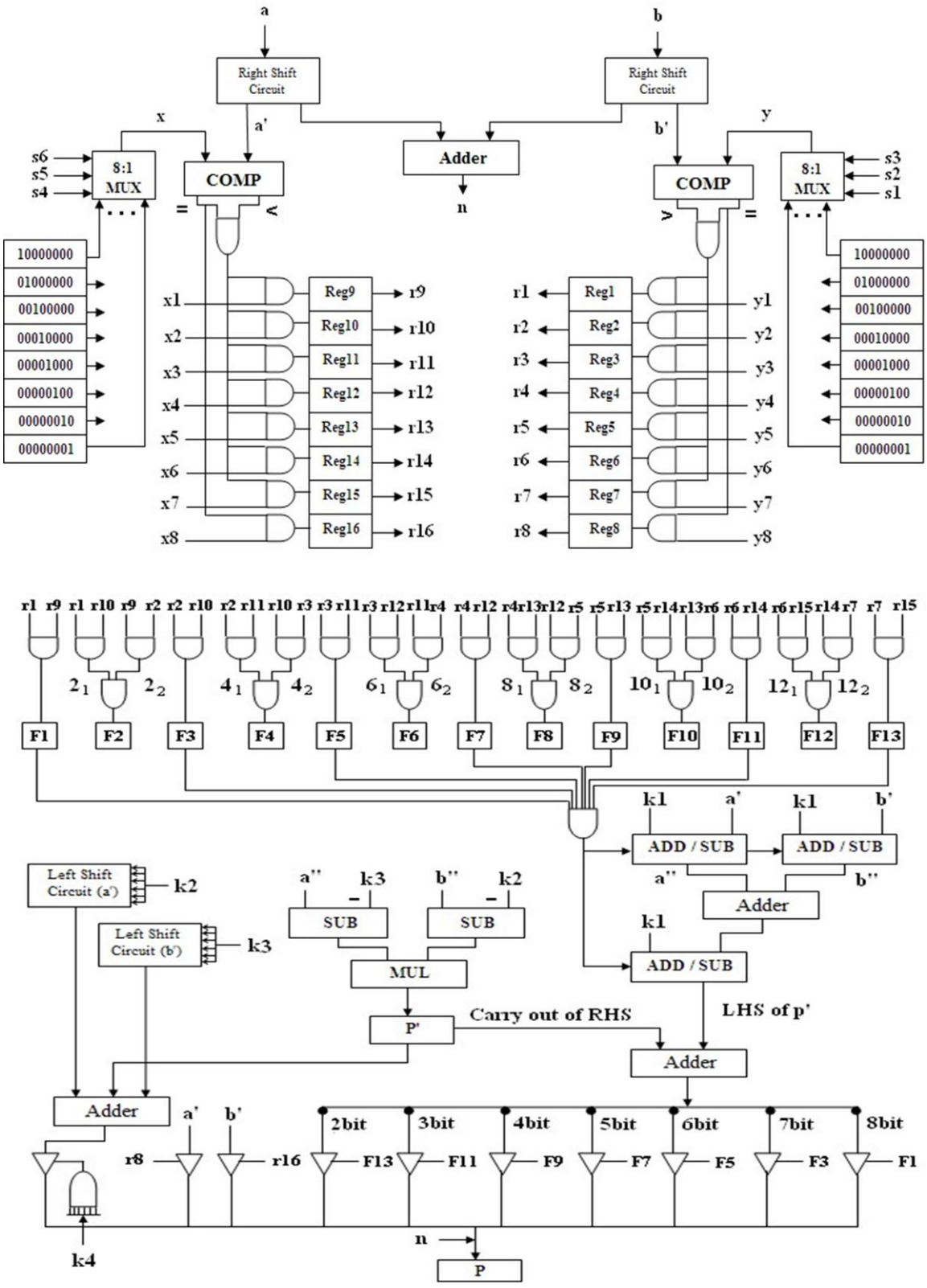


Figure 4: Hardware architecture of the proposed Vedic multiplier

In the Figure 4 k1 represent [F1, F3, F5, F7, F9, F11, F13, 0, 0], k2 = [22, 42, 62, 82, 102, 122, 0, 0], k3 = [21, 41, 61, 81, 101, 121, 0, 0] and k4 = [f2, f4, f6, f8, f10, f12].

4. Performance Comparison

All multiplier algorithms are tested and simulated by using VHDL and MAX + plus II environment (3s100evq100-5 configuration). And Performance analysis is performed using the Xilinx FPGA Spartan 3E (XC3S100E, Package VQ100, Speed -5) device. The VHDL and MAX + plus simulations of Urdhva, Booth and proposed Vedic multiplication algorithms are shown in Figures 5, 6 and 7 for 8 bit operands, respectively. Here, it is seen that arithmetic multiplication circuits are functionally verified.

The multiplier circuits are synthesized on the FPGA kit and their performance was obtained. As a performance criterions, from input to output the longest delay time and the total unit gate count (chip area) has been taken as a criterion. The delay here represents the delay on the FPGA kit. The Table 3 shown below is FPGA hardware performance of multiplication methods.

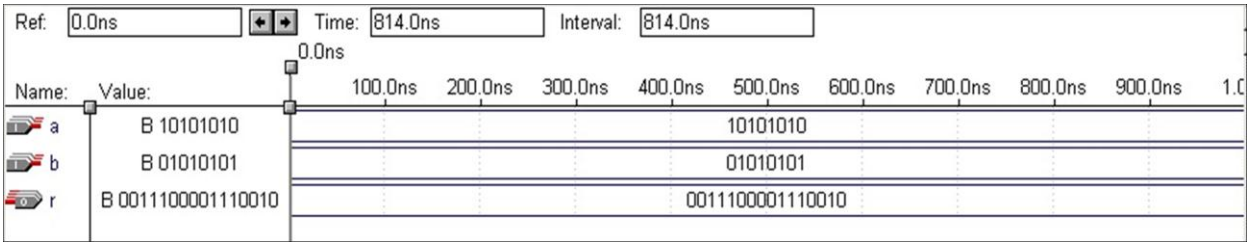


Figure 5: Timing diagram of 8x8 Urdhva multiplier

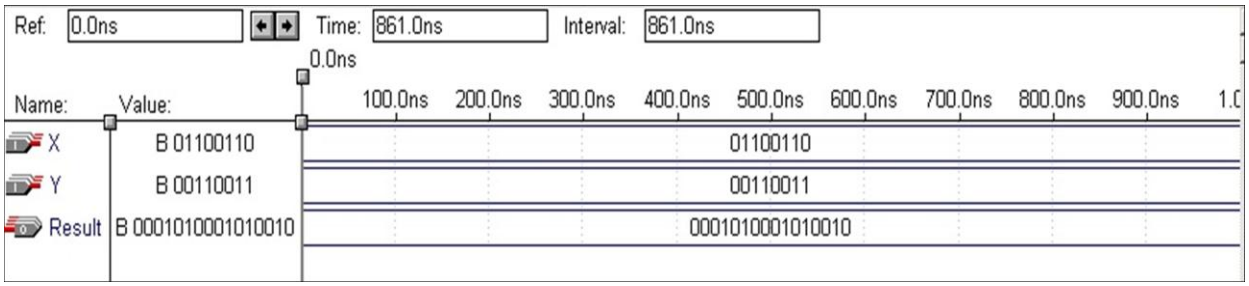


Figure 6: Timing diagram of 8x8 Booth multiplier

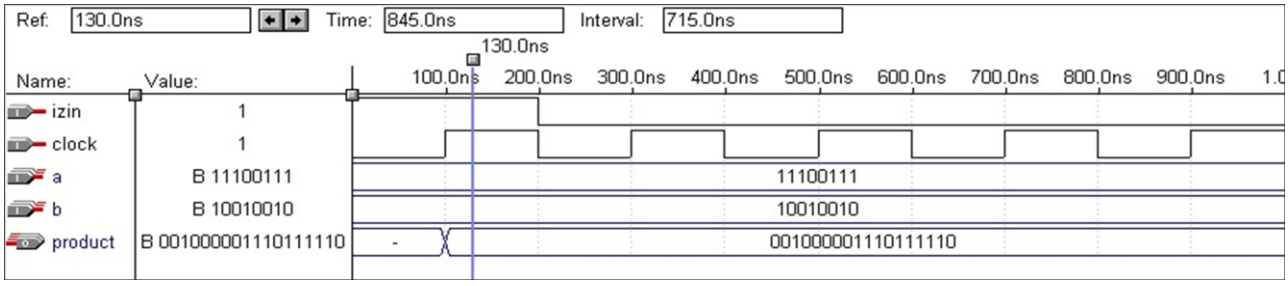


Figure 7: Timing diagram of 8x8 proposed Vedic multiplier

Table 3: FPGA implementation results

Device Utilization Summary (FPGA: Spartan 3E XC3S100E, Package VQ100, Speed -5)							
Type of Multiplier		**Urdhva**		**Booth**		**Proposed**	
Number of bits	**Available**	**4bit**	**8bit**	**4bit**	**8bit**	**4bit**	**8bit**
Number of Slices	960	16	75	18	93	13	47
Number of Slice Flip Flops	1920	0	0	0	0	8	15
Number of 4 input LUTs	1920	28	133	32	164	27	89
Number of bonded IOBs	66	16	32	16	32	15	27
Number of GCLKs	24	0	0	0	0	2	2
Number of IOs		16	32	16	32	18	34
Delay (ns)		11.005	21.544	12.767	23.934	6.947	9.887
Chip Area		228	634	217	592	260	1164

Figure 8 shows the delay time (T) depending on the bit length of the multiplication algorithms, Figure 9 shows the required chip area (A) based on the bit length, and the productivity AxT (power consumption) graph obtained by multiplying these two values. Here, when calculating the delay time T, the iteration counts of the algorithms, the number of shifting, and the re-coding times are taken into account.

According to the results obtained, the fastest of the multiplication algorithms is the proposed Vedic multiplication algorithm, but the excess chip area that is used with this speed increment is emerging. As the slowest algorithm, the Booth multiplication algorithm works slower. However, this algorithm requires minimum chip area. Urdhva multiplication algorithm is an algorithm that requires medium delay and medium chip area. Figure 10 shows that on the AxT graph used as the basic performance criterion, the minimal power consumption circuit belongs to the circuit implemented by the proposed Vedic method.

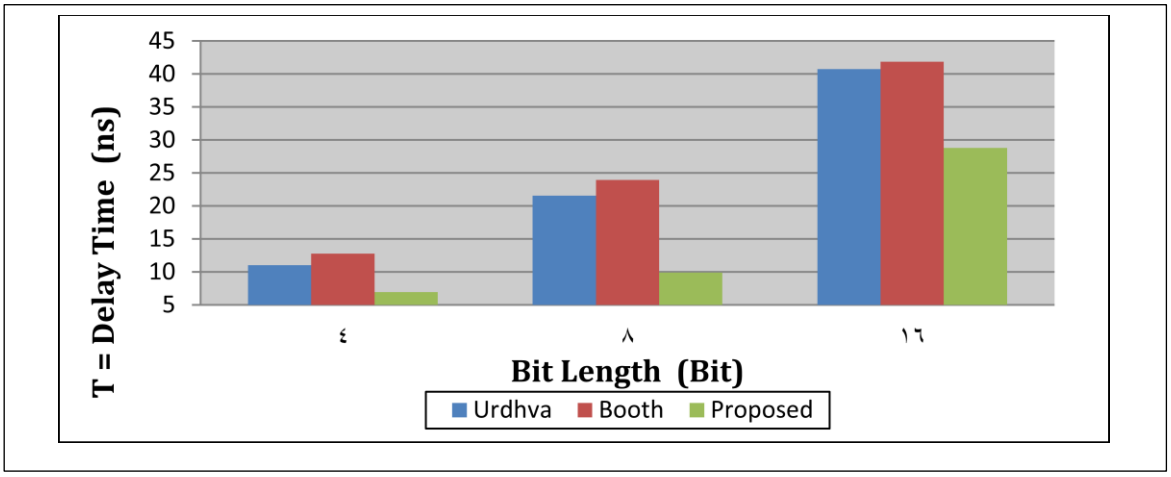


Figure 8: Delay (T) graph of multiplication algorithms

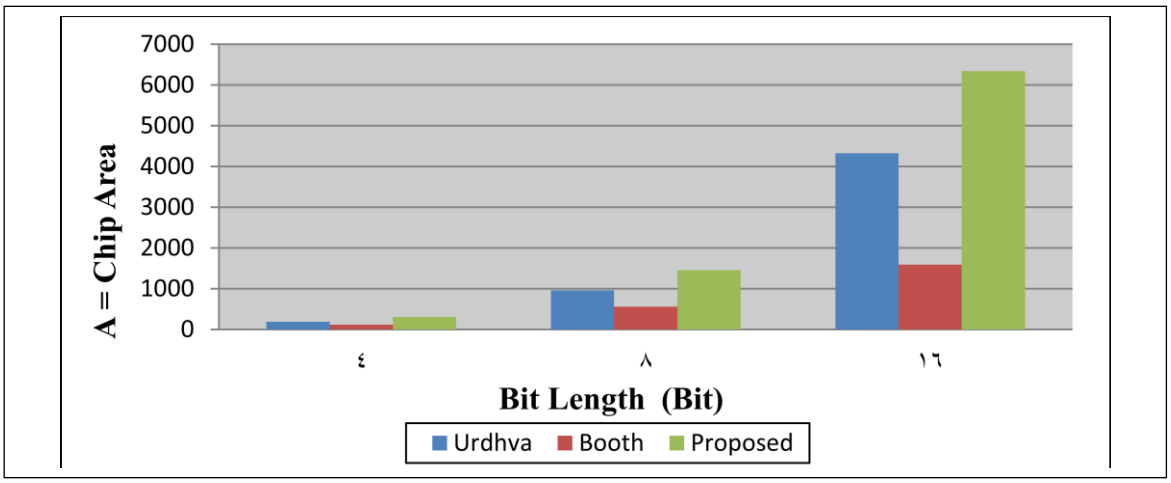


Figure 9: Chip Area (A) graph of multiplication algorithms

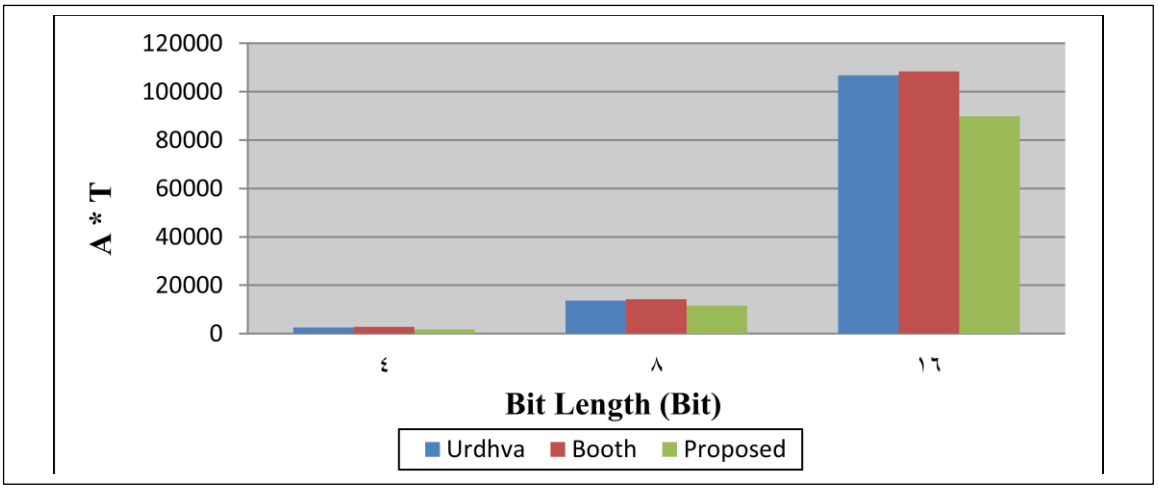


Figure 10: Power consumption (A*T) graph of multiplication algorithms

5. CONCLUSION

In this study, multiplication algorithms that based on Vedic Mathematics and based on the principle of efficient bit reduction are examined. Performance analysis of the algorithms was performed by simulating all the multiplication circuits in VHDL language. In addition, all hardware multiplication circuits have been synthesized by using FPGA kit to determine the performance of the circuits. In the Multiplier circuits, it can be concluded that the fastest one (i.e. the lowest delay time) is the proposed Vedic multiplier circuit, while the slowest one is the Booth multiplier circuit. On the other hand, it has been seen that the Booth multiplier uses the least number of the unit gates (i.e. it can be produced with the least cost). If the amount of power consumed in the chip is taken into consideration, the proposed Vedic multiplier circuit is best; The Booth multiplier circuit has been determined to have the worst performance. The Urdhva multiplier circuit exhibits a medium delay and a medium cost performance at the same time.

7. REFERENCES

[1] G.-K. Ma, F. J. Taylor, "**Multiplier Policies for Digital Signal Processing**", IEEE ASSP Mag., Vol. 7, no. 1, pp. 6–20, (1990).

[2] P. D. Chidgupkar and M. T. Karad, "**The Implementation of Vedic Algorithms in Digital Signal Processing**", Global J. of Engg. Edu., Vol. 8, no. 2, pp. 153-157, (2004).

[3] A.D. Booth, "**A Signed Binary Multiplication Technique**", Qrt. J. Mech. App. Math.,, Vol. 4, pp. 236–240, (1951).

[4] Harpreet Singh Dhillon and Abhijit Mitra, "**A Reduced-Bit Multiplication Algorithm for Digital Arithmetic**", International Journal of Computational and Mathematical Sciences 2, pp. 64-69, (2008).

[5] Wallace, C.S., "**A suggestion for a fast multiplier**", IEEE Trans. Elec. Comput., Vol. EC-13, no. 1, pp. 14–17, (1964).

[6] H. Thapliyal and M. B. Srinivas, "**High Speed Efficient N×N Bit Parallel Hierarchical Overlay Multiplier Architecture Based on Ancient Indian Vedic Mathematics**", Enformatika Trans., Vol. 2, pp. 225–228, (2004).

[7] Swami Bharati Krsna Tirtha, "**Vedic Mathematics**", Motilal Banarsidass, Varanasi, India, (1965).

[8] Asmita Haveliya, "**A Novel Design for High Speed Multiplier for Digital Signal Processing Applications (Ancient Indian Vedic mathematics approach)**", International Journal of Technology and Engineering System (IJTES), Vol. 2, no.1, pp. 27-31, (2011).

[9] Pushpalata Verma, K. K. Mehta, "**Implementation of an Efficient Multiplier based on Vedic Mathematics Using EDA Tool**", International Journal of Engineering and Advanced Technology, Vol. 1, pp. 75-79, (2012).

[10] M.C. Hanumantharaju, H. Jayalaxmi, R.K. Renuka, M. Ravishankar, "**A High Speed Block Convolution Using Ancient Indian Vedic Mathematics**", International Conference on Computational Intelligence and Multimedia Applications, Vol. 2, pp. 169-173, (2007).

[11] G.Ganesh Kumar and V.Charishma, "**Design of High Speed Vedic Multiplier using Vedic Mathematics Techniques**", International Journal of Scientific and Research Publications, Vol. 2, pp. 1-5, (2012).

[12] Rubinfield L.P., "**A proof of the Modified Booth's Algorithm for Multiplication**", IEEE Trans. Computers, Vol.25, no.10, pp. 1014-1015, (1975).

Journal of AL-Qadisiyah for computer science and mathematics Vol.10 No.1 Year 2018
ISSN (Print): 2074 – 0204 ISSN (Online): 2521 – 3504

تصميم وتنفيذ دوائر ضرب الفعالة وعالية السرعة بالاعتماد على خوارزميات فيديك

مثنى ياسين نواف
جامعة كركوك / كلية العلوم / قسم علوم الحاسوب
Muthana2085@yahoo.com

المستخلص :

مع مرور الايام تطلبت السرعة المتزايدة لمعالجات الكومبيوتر الى تصميم الدوائر الحسابية ذات الاداء العالي. وقد أتاح هذا الشرط إعادة النظر في الحساب الحاسوبي، ومكن من ظهور خوارزميات سريعة، وبعض تطبيقات الاجهزة الذي توفره التكنولوجيا .والغرض الرئيسي من الحساب الحاسوبي هو تصميم الدوائر والخوارزميات التي من شأنها زيادة سرعة المعالجة الرقمية. تحقيقا لهذه الغاية، هذا البحث يقدم تصميم وتنفيذ دوائر الضرب الحسابية ذات السرعة العالية جداً وبطول بت أعلى من خلال استخدام طريقة خفض بت فعالة وذلك من خلال إجراء تغييرات في بعض أساليب عملية الضرب التي تقوم على الرياضيات الفيدية، فقد تم تطوير دوائر الضرب ذات ٤ بت الى بت أعلى من الدوائر الضرب باستخدام بعض الخصائص الأساسية في عملية الضرب مثل التحليل وازاحة البتات. تم كتابة وتنفيذ كودات خوارزميات الضرب في لغة توصيف العتاد للدارات المتكاملة عالية السرعة المعروفة (VHDL) وتنفيذ تحليلات الأداء لدوائر الضرب الحسابية عن طريق التحقق الوظيفي من خلال المحاكاة XILINX و الميدان المتقشف لمصفوفات البوابات القابلة للبرمجة (FPGA)، والحصول على إشارة الخرج الموجي وقياسات أوقات التأخير.