RESEARCH ARTICLE OPEN ACCESS

Design And Analysis of Booth Multiplier Using FPGA

N.V.N.Prasanna Kumar*
*(Senior Assistant Professor, Department of ECE, Aditya Engineering College, Surampalem
Corresponding Author:N.V.N.Prasanna Kumar

ABSTRACT
Multipliers play an important role in today's digital signal processing and many other applications. Multiplication can be either signed multiplication or unsigned multiplication. In the case of unsigned multiplication, two binary numbers only with their magnitudes are involved in multiplication. In signed multiplication both the sign and magnitude of the multiplier and multiplicand are multiplied. Braun multipliers are used to perform unsigned multiplication. The signed multiplication was done by Baugh Wooley multiplier and Booth multipliers. In this paper, the structural (gate level) implementation of booth multiplier is carried out using Xilinx Spartan 3E FPGA board. It provides future scope in layout level or in back end level for post layout simulation.

Keywords*:* Baugh Wooley multiplier, Booth multiplier, Braun multiplier, Modified Booth Encoding, Modified Booth multiplier

---

Date of Submission: 15-12-2017 Date of acceptance: 28-12-2017

I. INTRODUCTION

1.1. Binary Multiplier

A Binary multiplier is an electronic hardware device used in digital electronics or a computer or other electronic device. It is built using binary adders. The rules for binary multiplication can be stated as follows

1. If the multiplier digit is a 1, the multiplicand is simply copied down and represents the product.
2. If the multiplier digit is a 0 the product is also 0.

1.2. Types Of Multipliers

1.2.1. Array Multiplier

The composition of an array multiplier is shown in figure. There is a one-to-one topological correspondence between this hardware structure and the manual multiplication shown in fig 1.1.The generation of partial products requires N x M two-bit AND gates most of the area of the multiplier is devoted to the adding of the N partial products, which requires N - 1 M-bit adders. The shifting of the partial products for their proper alignment is performed by simple routing and does not require any logic. The overall structure can easily be compacted into a rectangle, resulting in a very efficient layout. Due to the array organization, determining the propagation delay of this circuit is not straightforward. Consider the implementation of the partial sum adders are implemented as ripple-carry structures. Performance optimization requires that the critical timing path be identified first. This turns out to be non-trivial. In fact, a large number of paths of almost identical length can be identified.

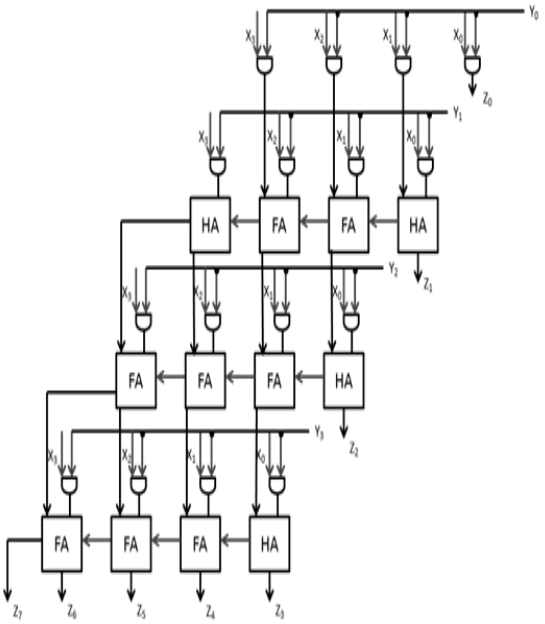

Fig.1.1: 4 × 4 bit-array multiplier

Two of those are highlighted in fig 2.2.

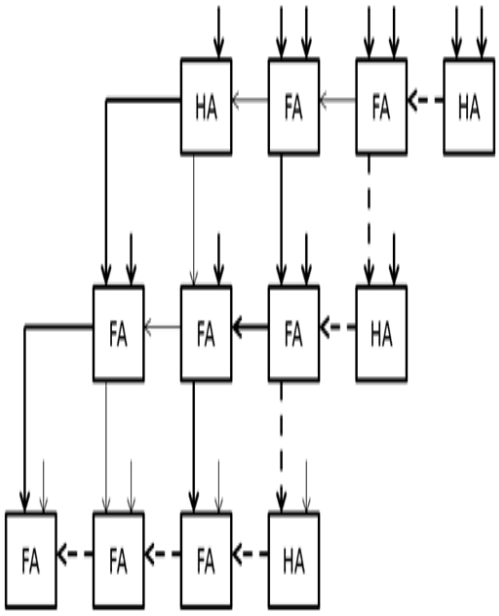

Fig.1.2: Ripple- carry based 4 × 4 multiplier

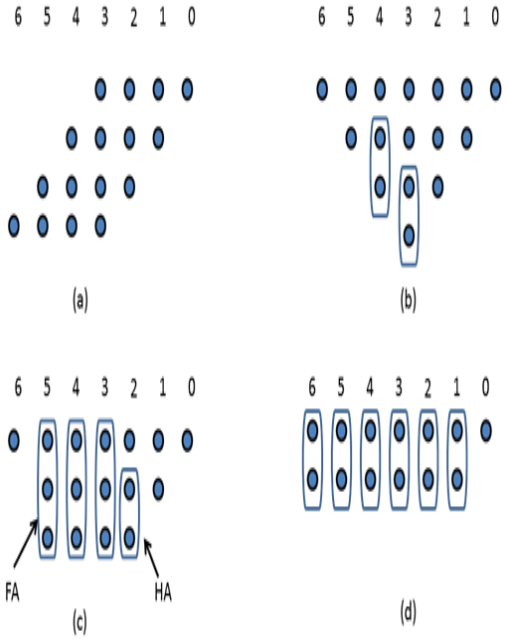

Fig.1.3: Wallace tree multiplier

1.2.2 Wallace Tree Multiplier:

In Tree multipliers the partial-sum adders can also be rearranged in a treelike fashion, reducing both the critical path and the number of adder cells needed. Wallace multiplier is summing the partial product bits in parallel using a tree of Carry Save Adders which became generally known as the "Wallace Tree", With Wallace method, a three step process is used to multiply two numbers: One is to form the bit products. The other is that the bit product matrix is "reduced" to a two row matrix by using Carry-save adders (Known as Wallace Tree). Lastly the remaining two rows are summed using a fast carry-propagate adder to produce the product. Although this may seem to be a complex process, it yields multipliers with delay proportional to the logarithm of the operand size 'n'.

The partial-sum adders can also be rearranged in a treelike fashion, reducing both the critical path and the number of adder cells needed. With this process, the number of adder cells required can be reduced. This is illustrated in fig 1.3, where the original matrix of partial products is reorganized into a tree shape to visually illustrate its varying depth. The first type of operator that can be used to cover the array is a full adder, which takes three inputs and produces two outputs: the sum, located in the same column and the carry, located in the next one. For this reason, the FA is called a 3-2 compressor. It is denoted by a circle covering three bits. The other operator is the half-adder, which takes two input bits in a column and produces two outputs.

To arrive at the minimal implementation, iteratively cover the tree with FAs and HAs, starting from its densest part. In the first step, we introduce HAs in columns 4 and 3. The reduced tree is shown in Figure1.3.(b). A second round of reductions creates a tree of depth 2 (Fig 1.3 c). Only three FAs and three HAs are used for the reduction process. The final stage consists of two-input adders, for which any type of adder can be used. The presented structure is called the Wallace tree multiplier, and its implementation is shown in fig 1.3. The tree multiplier realizes substantial hardware savings for larger multipliers. The propagation delay is reduced as well. There is numerous other ways to accumulate the partial-product tree.

1.2.3. braun multiplier:

The simplest parallel multiplier is the Braun array. All the partial products are computed in parallel, and then collected through a cascade of Carry Save Adders. The completion time is limited by the depth of the carry save array, and by the carry propagation in the adder. Note that this multiplier is only suited for positive operands. The structure of the Braun algorithm for the unsigned binary multiplication is shown in figure 1.4.A and B are four bit inputs, S_{ij} represent the intermediate values. Here 16 AND gate's and 12 Full adders are employed to obtain the final product.

The logic diagram of the Braun Multiplier is as follows:

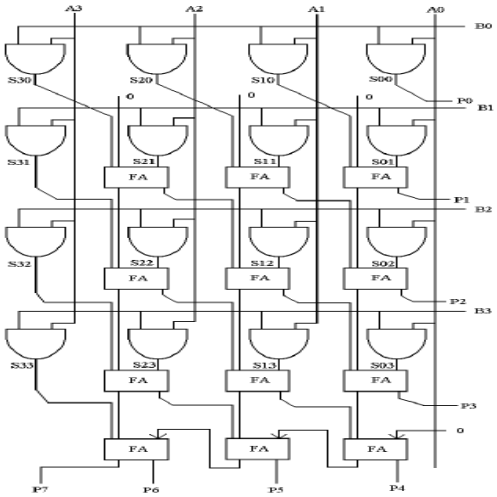

Fig. 1.4 Braun Multiplier

1.2.4 Baugh-Wooley Multiplier:

The Baugh-Wooley multiplication algorithm is an efficient way to handle the sign bits. This technique has been developed in order to design regular multipliers, suited for 2's complement numbers. Let us consider two n bit numbers, A and B, to be multiplied. A and B can be represented as

$$A = -a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i \quad (1)$$

$$B = -b_{n-1}2^{n-1} + \sum_{i=0}^{n-2} b_i 2^i \quad (2)$$

Where a_i and b_i represents i^{th} bits of A and B, respectively, and a_{n-1} and b_{n-1} are the sign bits. The product, P=A*B, is then given by the following equation:

$$P = A * B = (-a_{n-1}2^{n-1} + \sum_{i=0}^{n-2} a_i 2^i) * (-b_{n-1}2^{n-1} + \sum_{j=0}^{n-2} b_j 2^j)$$

$$= a_{n-1}b_{n-1}2^{2n-2} + \sum_{i=0}^{n-2}\sum_{j=0}^{n-2} a_i b_j 2^{i+j} - 2^{n-1}\sum_{i=0}^{n-2} a_i b_{n-1} 2^i - 2^{n-1}\sum_{j=0}^{n-2} a_{n-1} b_j 2^j \text{------- (3)}$$

Equation (3) indicates that the final product is obtained by subtracting the last two positive terms from the first two terms.

Rather than to do subtraction we can obtain the 2's complement of the last two terms and add the all terms to get the final product. The last two terms are n-1bits each that extend from the binary weight from the position 2^{n-1} up to 2^{2n-3}. On the other hand, the final product is 2n bits and extends in binary weight 2^0 up to 2^{2n-1}.

Assuming X I one of the last two terms we can represent it with zero padding's as

$$X = -0 \times 2^{2n-1} + 0 \times 2^{2n-2} + 2^{n-1}\sum_{i=0}^{n-2} x_i 2^i + \sum_{j=0}^{n-2} 0 \times 2^j$$

The above equation gives the value of X due to the fact that a negative value is associated with the MSB. The multiplication process for the Baugh-Wooley is shown in figure 1.5.

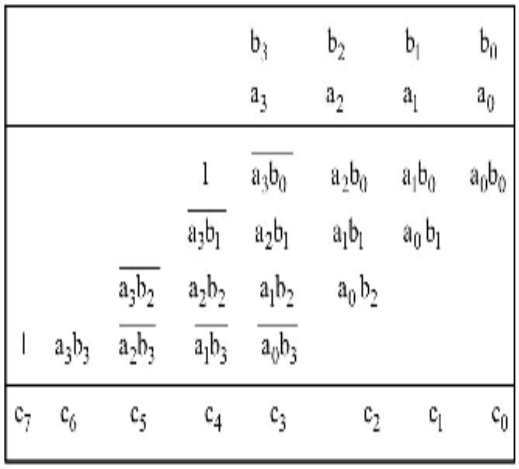

Fig. 1.5 Multiplication process for the Baugh-Wooley

When two 8 bit numbers are multiplied the resultant product size will be the sum of the number of bits in the given inputs. Then the product will be maximum of size 16 bits. a and b are two 8 bit signed inputs and P is the product obtained after multiplication. The block diagram representation of the 4-bit Baugh Wooley multiplier is shown in fig 1.6.

1.2.5 Booth's Multiplier:

Booth's multiplier works on two's complement. It is similar to paper-pencil method, except that it looks for the current as well as previous bit in order to decide what to do.

It is done in the following steps

1. If the current multiplier digit is 1 and earlier digit is 0 (i.e. a 10 pair) shift and sign extend the multiplicand, subtract with previous result.
2. If it is a 01 pair, add to the previous result.
3. If it is a 00 pair, or 11 pair, do nothing.

Note that the multiplicand and multiplier are 8-bit two's complement number, but the result is a 16-bit two's complement number. "10" pair causes a subtraction, aligned with 1, "01" pair causes an

addition, aligned with 0. In both cases, it aligns with the one on the left. The algorithm starts with the 0^{th} bit. Assume that there is a $(-1)^{th}$ bit, having value 0.

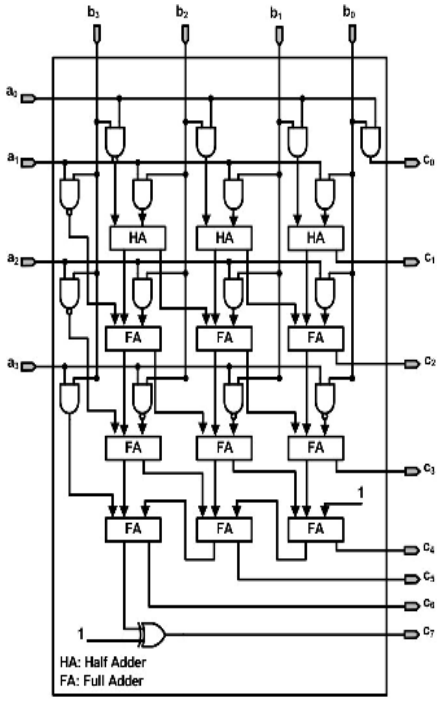

Fig.1.6: Block Diagram of Baugh Wooley

1.2.6 Modified Booth Multiplier:

Modified Booth Encoding (MBE) is a technique that has been introduced to reduce the number of PP rows, still keeping the generation process of each row both simple and fast enough. In this method, the bits can be encoded by considering three bits at a time. The most commonly used technique is radix-4 MBE, since it allows for the reduction of the size of the partial product array by almost half, and it is very simple to generate the multiples of the multiplicand. More specifically, the classic two's complement n * n bit multiplier using the radix-4 MBE scheme, generates a PP array with a maximum height of [n/2]+1 rows, each row before the last one being one of the following possible values: all zeros, +/-X; +/-2X. By using Booth Recoding table of Modified Booth technique, we can generate partial products of minimum width.

i+1	I	i-1	Partial product
0	0	0	0*M
0	0	1	1*M
0	1	0	1*M
0	1	1	2*M
1	0	0	–2*M
1	0	1	–1*M
1	1	0	–1*M
1	1	1	0*M

Table 2.1 Modified booth encoding table

The steps involved in the generation of partial products using Modified Booth algorithm are given below:

1. Pad the LSB with one zero.
2. Pad the MSB with 2 zeros if n is even and 1 zero if n is odd.
3. Divide the multiplier into overlapping groups of 3-bits.
4. Determine partial products from modified booth encoding table.
5. Compute the Multiplicand Multiples
6. Sum Partial Products

II. IMPLEMENTATION

4.1 Booth Multiplication

The inputs to be multiplied called multiplier and multiplicand are given. The MSB denotes the sign of the number. If the multiplier(x) or the multiplicand(y) or both are negative the 2's complement block is enabled and the 2's complemented multiplier is named x2, the 2's complemented multiplicand is called y2. The multiplier and its 2's complement are given to 8 bit 2*1 multiplexer the selection input is the sign bit of the multiplier. The output of the multiplexer is positive if the selection input is zero else it is negative. The multiplexer output of the multiplier is M1. Similarly the multiplicand and the 2's complemented multiplicand are given to the 8 bit 2*1 multiplexer; the selection input is the sign bit of the multiplicand. The output of the multiplicand is positive if the selection input is zero else it is negative.

The multiplexer output of the multiplicand is called M2. The M2 is concatenated with 9 bits after the LSB. The resultant number is called R2. The value of the M2 is 2's complemented by using a 8 bit 2's complement block and then concatenated with 9 zeros after LSB and this number is named ad y_12.

The number M1 is concatenated with 8 zero's before MSB and a single bit zero after LSB. Taking a buffer named K2 whose current value is zero and of size 17 bits. The M1 and K2 are given as inputs to the 17 bit 2*1 multiplexer. The selection line input of the Multiplexer is generated as follows: A 3 bit down counter output is given to the input of the 3 bit and gate the output of the and gate is zero only for the first count and for remaining counts the output is zero. In this way the 17bit 2*1 multiplexer output is M1 for the first count and K2 for the remaining count. The output of the multiplexer is K3. K3 is a 17 bit register the K3(0) and K3(1) bits are given as inputs to the 2 to 4 decoder The outputs of the decoder are the enable inputs for 17 bit adders and shifter. According to Booth algorithm if the K3(1) and k3(0) are "00" or "11"then only shifting operation have to be performed. If K3(1) and K3(0)

are "01" then addition operation accompanied with shifting operation have to be performed, else subtraction operation have to be performed. The y0 and y3 the outputs of the decoder are given to an OR gate because if any of the outputs is HIGH then we have to perform the shifting operation basing on algorithm. The output is given as input to the Shifter. The shifter performs the right shift arithmetic operation on the given 17 bit input. The output of the shifter is zero if the shifter is disabled. The output of the decoder y1 is given as the enable input to the 17 bit adder the inputs of this adder are K3 and R1, since addition of the R1 and to the left most part of the K3.The result after the addition is given to the shifter to perform the right arithmetic shift operation. The output y2 of the decoder is given as the enable input to the 17 bit adder whose inputs are K3 and y_12. After the addition the number is given to shifter for right arithmetic shift operation. The output of the both adders is zero if they are disabled. In the end after the addition the output of the adders are right shifted by using the shifter. The results from the shifters are added by using 17 bit adder and the final result is placed in the K2 register. All this process occurs for single count of clock in the counter. The K2 value updated for every clock cycle. The number of clock cycles depends upon the number of bit does the input will have. As here we are interested in 8 bit multiplier, the numbers of clock cycles are eight. After completion of the 8 cycles the result or the product will stored in K2. If only one input is negative then the result value is the 2's complement of the actual result to obtain the result we have to 2's complement the result. Here we obtain a 17 bit output, basing on the last bit must be discarded to obtain the output.

Block Diagram Of 8 Bit Booth Multiplier Is Shown In Figure 4.1:

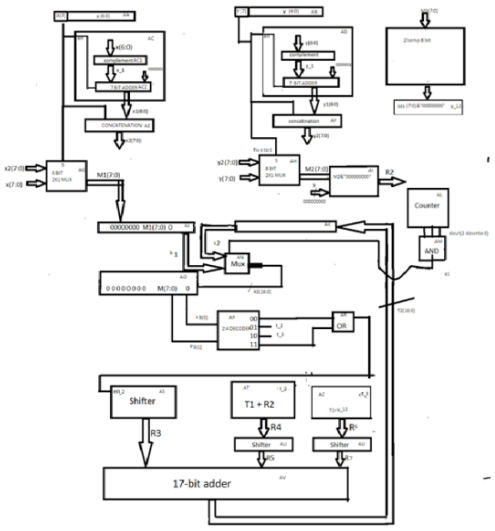

Fig. 4.1 Block diagram of 8 Bit Booth Multiplier.

III. SIMULATION RESULTS:

The simulation is carried out using Xilinx software for different combinations of inputs.

Case1: for two positive inputs:

The simulated output for two positive numbers is shown in figure 5.1. The inputs of the design are as shown below:

Multiplier x=00100000$_2$ (32)$_{10}$,
Multiplicand y=00101000$_2$ (40)$_{10}$.
The result final product is
K2=00000101000000000$_2$ (1280)$_{10}$.

Fig.5.1: Simulation result for Two Positive inputs

Case2: for one positive input and one negative input:

The simulated output for one positive number and one negative number is shown in figure 5.2. The inputs of the design are as shown below

Multiplier x=0001100$_2$ (-24)$_{10}$,
Multiplicand y=10011000$_2$ (24)$_{10}$.

The result final product is
K2=11111101110000000$_2$ (-576)$_{10}$

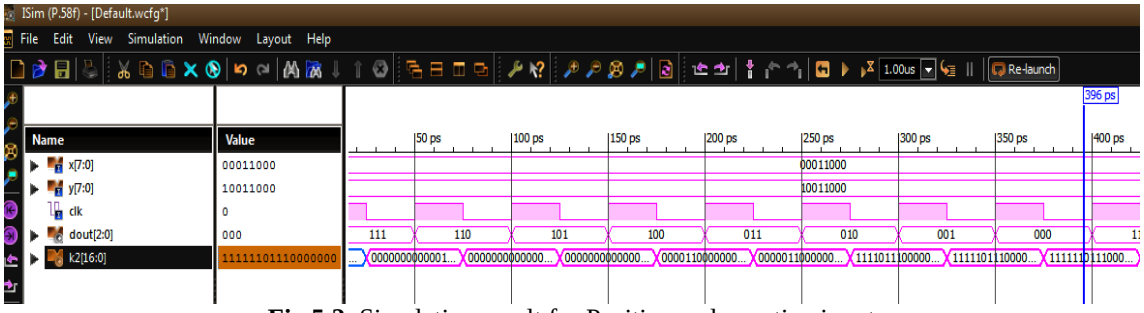

Fig.5.2: Simulation result for Positive and negative input

Case3: for two negative inputs:
The simulated output for two negative numbers is shown in figure 5.3. The inputs of the design are as shown below.

Multiplier **x= 1001001_2 $(-17)_{10}$,**
Multiplicand **y= 1001001_2 $(-17)_{10}$.**
The result final product is
K2=0000000100100010_2 $(289)_{10}$.

Fig.5.3: Simulation result for two negative inputs

IV. CONCLUSION AND FUTURE SCOPE

6.1 Conclusion:

The simulation results indicate that the proposed method can perform the multiplication of given inputs. The device is designed using Booth multiplier algorithm. The design is implemented for 8 bit multiplication of two inputs and the design is synthesized. It is observed that the power dissipation is reduced to 26.01mW. The area is reduced with the reduction in number of partial products compared with conventional booth multiplier technique.

6.2 Future Scope:

The design is implemented in front end environment. It can be further processed to back end environment for fabricating the ASIC's. The design can be optimized for better performance in terms of its cost including power dissipation, delay and area. Further more in order to reach the current technological requirement, higher order multiplier like 16bit, 32 bit, 64 bit multipliers can be designed.

REFERENCES

[1]. Computer System Architecture(3rd edition), by M.Morris Mano
[2]. CMOS VLSI Design by Neil.H.E.Weste and
[3]. David Money Harris.
[4]. http://staff.ustc.edu.cn/~han/CS152CD/Cont ent/C
[5]. OD3e/InMoreDepth/IMD3-Booths-
[6]. Algorithm.pdf
[7]. https://www.youtube.com/watch?v=1aTR9 WQF FtM
[8]. http://shodhganga.inflibnet.ac.in:8080/jspui/ bits
[9]. tream/10603/6521/10/10_chapter%205.pdf

N.V.N.Prasanna Kumar "Design And Analysis of Booth Multiplier Using FPGA. "International Journal of Engineering Research and Applications (IJERA) , vol. 7, no. 12, 2017, pp. 81-86.