DESIGN AND IMPLEMENTATION OF HIGH SPEED LOW POWER RADIX-16 BOOTH MULTIPLIER

[1]T Y Lakshmi, [2]Dr. P Sreenivasulu
[1]M.Tech., Student, [2]Professor
[1]ECE Department,
[1]Narayana Engineering College, Gudur, India

Abstract: The IC Technology focuses on the planning of ICs considering additional space improvement and low power techniques. In Scientific applications multiplication may be a heavily used for operation that figures conspicuously in signal process. Multiplication may be a hardware intensive subject and thus we as users area unit largely involved with obtaining low-power, smaller space and better speed. The foremost necessary concern in classic multiplication largely accomplished by K-cycles of shifting and adding, is to hurry up underlying multi-operand addition of partial product. During this paper we will design and implement the Booth multiplier using Ripple Carry Adder architecture. Multipliers are designed for each radix-2, radix-4 and radix-8. The Results can show that the multiplier is able to multiply two 32 bit signed numbers and how this technique reduces the number of partial products, which is an important factor to be achieved in this paper. Ripple Carry adder have simplification of addition operation and power reduction property we can propose a low power radix-4 modified booth multiplier. compared with the radix 4 modified booth multiplier using carry look ahead adder (CLA),the experimental result shows that our propose design has reduce the power dissipation to 25.27 % using Ripple Carry Adder, power has estimated as 8.22mW which was 11mW when designed with CLA Adder.

***IndexTerms* - Booth Multiplier, Radix8, Ripple carry adder (RCA), Booth Decoder, Partial Product.**

I. INTRODUCTION

Now a Days Analogue systems are replacing by digital systems because of its properties have high speed performance, takes less area and less power dissipation. In Modern IC Technology Multiplication is one of the most important and basic arithmetic operation that constitute programs [1]. Multiplication Operation has 8.72%of all instructions in typical Scientific Program. Many multipliers have been proposed in the past with consideration of small area, low power and high performance. Multiplication is achieved by the addition of a certain number of partial products rows. Each partial product row is generated by multiply the multiplier bit one by one to multiplicand. In a simple multiplier, the generated partial products rows are equal to the number of bits in multiplier. For example, in 8×8 [2] bit multiplication, it will produce 8 partial product rows. It will take more adders and more time.

One of the simplest and mostly used multiplier is booth multiplier. It improves the performance of multiplier. The number of partial products rows that must be added to give the multiplication's result can be reduced by using Booth decoding. In Booth multiplier [3], the numbers of reduced partial products rows are depend on the grouping done at multiplier bits. Booth algorithm is scan and skips the chains and it can reduce the number of additions and it produces the result. Compare to the conventional multiplication algorithm, where each bit of the multiplier is multiplied with the multiplicand and the partial products are aligned and added together. This multiplication process is completed in 3 steps. First step: multiplier bits are divided in groups then and these groups are fed to decoder at where it will indicate that which operation is to perform on multiplicand. Second step: Then here, the indicated operation performs on the multiplicand and it will generate the partial products. Third step: Now the generated partial products are adding with adders [4].

After generation of partial products, there are many techniques used for adding the partial products with difference performances .In this proposed paper one of new techniques used that is Ripple Carry Adder [7]. Ripple carry adder have the power reduction property as well as simplification of addition operation. It will increases the performance of the multiplier and reduces the power dissipation, which is mostly required in Digital Systems.

II. RIPPLE CARRY ADDER

This well-known adder design, ripple carry adder consists of cascaded full adders as shown in Figure1. It is shaped by cascading full adder blocks non parallel with each other. The output carry of 1 stage is fed on to the input carry of following stage. AN N-bit parallel adder needs N full adders.

The ripple carry adder design isn't terribly economical once sizable amount of bits square measure used. The gate delay will simply be calculated by inspecting the total adder circuit. It was known that every full adder needs 3 levels of logic. Considering a 64- bit ripple-carry adder, we all know that it's sixty four full adders, therefore the crucial path (worst case) delay is three (from input to hold just in case of the primary adder) + sixty three * two (for carry propagation within the later adders) = 127 gate delays.

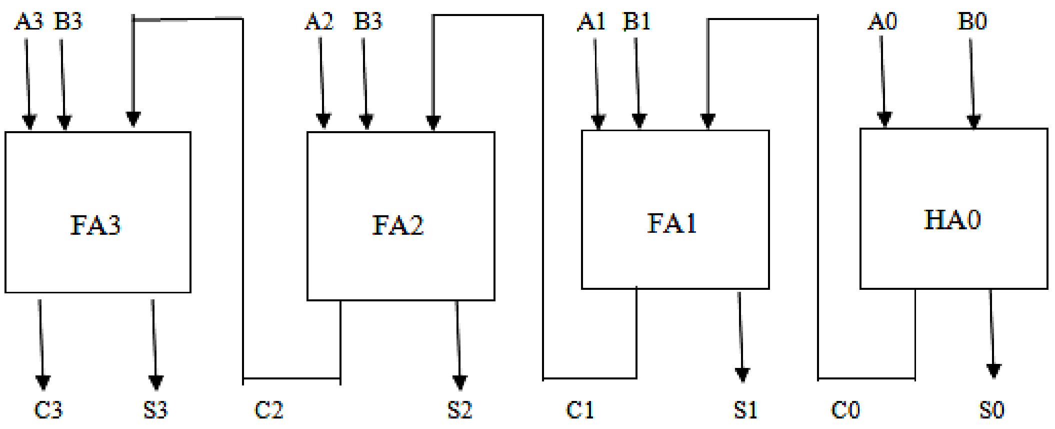


Figure 1: Ripple Carry Adder

2.1 Analysis of Ripple Carry Adder

It is known that combinational logic circuits cannot cipher the outputs instantly. There's some delay between the inputs square measure sent to the circuit, and therefore the time the output is computed.

Let's say the delay is T units of your time. Suppose we have to implement associate degree n-bit ripple carry adder. Since associate degree n-bit ripple carry adder consists of n adders, there'll be a delay of non-governmental organization. This is often O (n) delay. While the adders square measure operating in parallel, the carry's should "ripple" their means from the smallest amount vital bit and work their thanks to the foremost vital bit. It takes T units for the performance of the right column to create it as input to the adder within the next to right column.

Thus, the carries abate the circuit; it creates the addition linear quantity bits within the adder. This is not a giant 14 drawback, usually, as a result the hardware adder's square measure mounted in size. They add, say, thirty two bits at a time. There is not associate degree thanks to create an adder add an impulsive variety of bits. It will be exhausted code, not in hardware. In effect, this is often what makes hardware "hard". Even though there square measure a set variety of bits to feature in an adder, there square measure ways that to create the adder to add a lot of quickly.

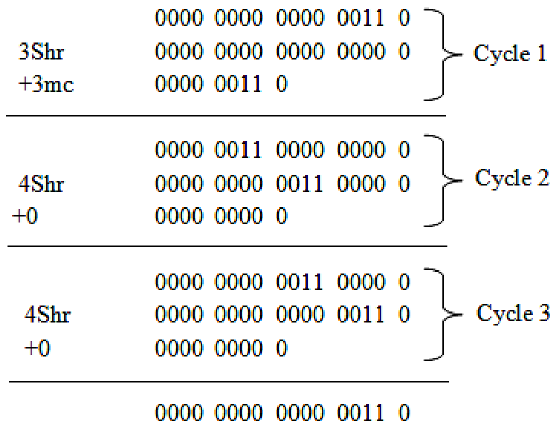
III. Radix-16 Booth Encoder Multiplier

Booth Encoding Algorithm is the one of the most well known technique used to reduce the number of partial products. That will be added while multiplying the multiplicand higher radix.Radix-16 generally Booth algorithm simplifies and scan strings of five bits. This Radix-16 Booth Encoding multiplier uses 5-bit encoding scheme to produce one fourth number of partial products. The combinations of one pair of Ck which consist of 5-bits and Sk to encode Radix-16 Booth Encoding multiplier is shown in below Table 1.

Radix-16 Booth algorithm which scan and strings of five bits with the algorithm given below: Extend the sign bit 1 position if necessary to ensure that n is even append a 0 to the right of the LSB of the multiplier. According to the value of each vector, each Partial Product will be 0, +y, –y, +2y,–2y,+3y,- 3Y,+4y,-4y,+5y,-5y,+6y,-6y,+7y,-7y,+8y,-8y .The negative values of y are made by taking the 2's complement and Carry-look-ahead (CLA) fast adders are used [5]-[7]. By shifting y by one bit left multiplication is carried out. Thus, in any case, Designing n-bit parallel multipliers, n/4 partial products are generated. Grouping starts from the LSB, and the first block only uses five bits of the multiplier. First of all we will make group of five bits for Multiplier Encoding for Radix-16 Booth Multiplier will be done according to the table given below:

Example:

Multiplication of 2*3
Multiplicand = 0000 0001
Multiplier = 0000 0011



```
            0000 0000 0000 0011 0 ┐
3Shr        0000 0000 0000 0000 0 ├ Cycle 1
+3mc        0000 0011 0           ┘
-------------------------------------
            0000 0011 0000 0000 0 ┐
4Shr        0000 0000 0011 0000 0 ├ Cycle 2
+0          0000 0000 0           ┘
-------------------------------------
            0000 0000 0011 0000 0 ┐
4Shr        0000 0000 0000 0011 0 ├ Cycle 3
+0          0000 0000 0           ┘
-------------------------------------
            0000 0000 0000 0011 0
```

BLOCK	PARTIAL PRODUCTS
00000,11111	0
00001,00010	+1XMULTIPLICAND
00011,00100	+2XMULTIPLICAND
00101,00110	+3XMULTIPLICAND
00111,01000	+4XMULTIPLICAND
01001,01010	+5XMULTIPLICAND
01011,01100	+6XMULTIPLICAND
01101,01110	+7XMULTIPLICAND
01111	+8XMULTIPLICAND
10000	-8XMULTIPLICAND
10001,10010	-7XMULTIPLICAND
10011,10100	-6XMULTIPLICAND
10101,10110	-5XMULTIPLICAND
10111,11000	-4XMULTIPLICAND
11001,11010	-3XMULTIPLICAND
11011,11100	-2XMULTIPLICAND
11101,11110	-1XMULTIPLICAND

Table 1: Combinations of block and partial products for Radix-16 Booth Encoding Multiplier

IV. BOOTH'S ENCODING

Booth Multiplier can reduce the number of iteration step to perform multiplication as compare to conventional steps. Flow chart of booth encode multiplier is as shown in below Figure 2.Booth algorithm 'scans' the multiplier operand and skips chains of this algorithm can reduce the number of additions required to produce the result compared to Conventional Multiplication algorithm, where each bit of the multiplier is multiplied with the multiplicand and the partial products are aligned and added together.

Booth's multiplication rule could be a multiplication algorithm which might multiply 2 signed binary numbers during a two's complement notation. Booth's rule has the power to perform fewer additions and subtractions as compared to traditional multiplication rule. It's an encryption method which might be accustomed notation. While compared to the traditional multiplication rule, booth's rule has the power to perform the subtractions and additions. The flow chart for booth encoder multiplier is shown in below.

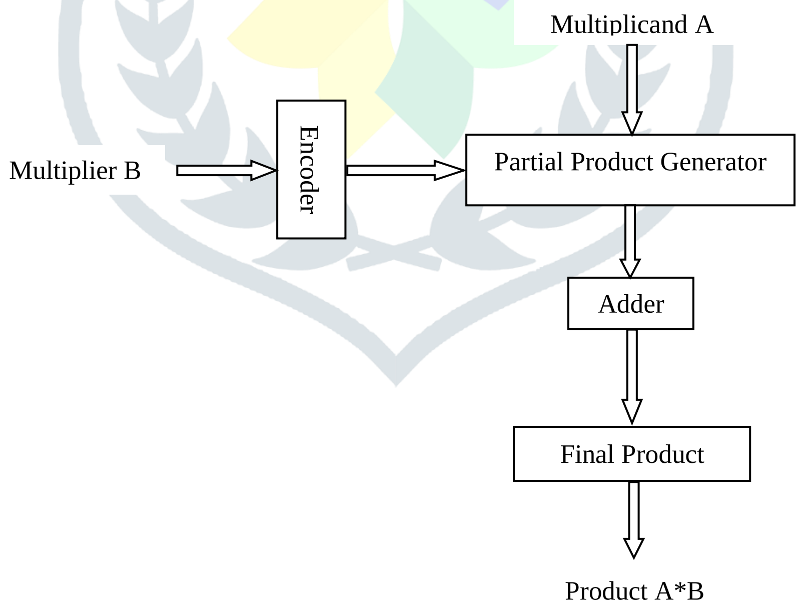


Figure 2: Flow chart for Booth Encoder Multiplier

Example:

0 0 1 1 1 1 1 1 0 0
+1 -1
+1 -1
+1 -1
+1 -1

+1 -1
+1 -1
0 +1 0 0 0 0 0 -1 0 0

Booth's algorithmic program examines consecutive bits of the N-bit number Y in signed two's complement illustration, which has Associate in implicit bit below the smallest amount important bit, y-1 = 0. For every bit Lolo, as i runs from zero to N-1, the bits Lolo and yi-1 square measure thought of. Once these 2 bits square measure equal, the accumulator P stays unchanged. Wherever Lolo = zero and yi-1 = one, the number times 2i is additional to P; and wherever Lolo = one and yi-1 = zero, the number times 2i gets subtracted from P. The ultimate price of P is the signed product.

The illustration of the number and products don't seem to be specified; usually, these also are in two's complement illustration, sort of a number, however any system of numeration that supports addition and subtraction can work furthermore. The specific order of the steps isn't determined. Generally, it issue from LSB to mutual savings bank, beginning at i = 0; the multiplication by 2i is then replaced by shifting of the P accumulator to the proper between steps; low bits are shifted out, and resultant additions or subtractions will then be done simply on the best N bits of P. There square measure several variations and optimizations on these details. The algorithmic program is commonly delineate as changing strings of 1's within the number to a high-order +1 and a low-order –1 at the ends of the string. Once the string runs through the mutual savings bank, there's no high-order +1, and also the web result is interpretation as a negative of the acceptable price.

V. Implementation And Results

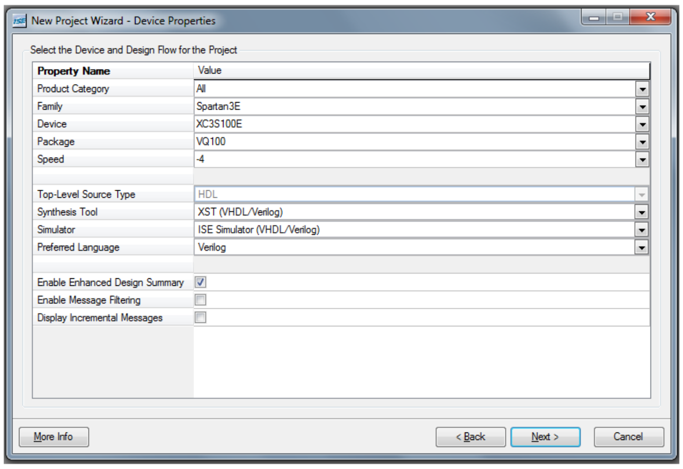
After partial product generated, there are many techniques to add them to generate final product like using half adder and full adder, wallace tree, carry save adder, Carry look adder etc. During this paper we will design the Booth multiplier [3] using Ripple Carry Adder architecture. Additionally multipliers are designed for each radix-2, radix-4 and Radix-8. Based on the simplification of addition operation and power reduction property in ripple carry adder (RCA), a low power radix 4 modified booth multiplier is proposed, compared with the radix 4 modified booth multiplier using carry look ahead adder (CLA), the experimental result shows that our propose design has reduce the power dissipation to 25.27 % using RCA, power has estimated as 8.22mW which was 11mW when designed with CLA Adder [1]-[3].

5.1 Comparison of Radix 2, Radix 4 and Radix 8 Algorithm

The shortcoming of Radix 2 Booth’s algorithm is that it becomes inefficient when there are isolated 1's. For example, 001010101 (decimal 85) gets reduced to 01-11- 11-11-1 (decimal 85), requiring eight instead of four operations. 001010101(0) recoded as 011111111, requiring 8 instead of 4 operations. This problem can be overcome by using high radix Booth’s algorithms [3].

As we move towards Radix 8 less number of partial products are generated but more number of operations are required to generate {+1,+2, +3, +4, -1, -2, -3, -4}. In Radix 4 we need to save {+2, -2, +1, -1}. Speed of Radix 8 is highest among Radix 2, 4 and 8 but the complexity increases.

Xilinx 12.3i ISE Simulator (VHDL/Verilog) has been used to simulate the proposed methodology of multiplication of two 16 bit numbers using Radix-16 modified booths algorithm. The device used by the simulator is XC3S100E of Spartan3E family with the speed of -4 which is shown in Figure 3.



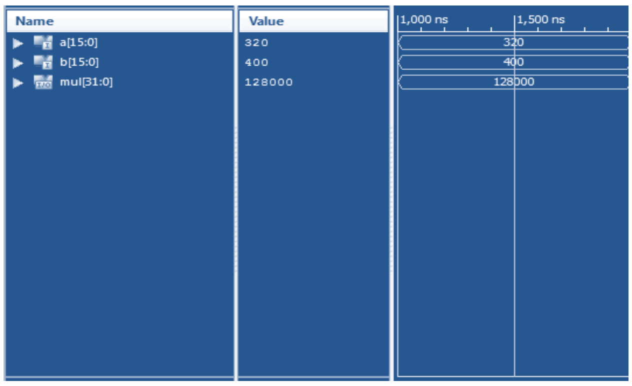
Select the Device and Design Flow for the Project

Property Name	Value
Product Category	All
Family	Spartan3E
Device	XC3S100E
Package	VQ100
Speed	-4
Top-Level Source Type	HDL
Synthesis Tool	XST (VHDL/Verilog)
Simulator	ISE Simulator (VHDL/Verilog)
Preferred Language	Verilog
Enable Enhanced Design Summary	✓
Enable Message Filtering	☐
Display Incremental Messages	☐

Figure 3: Properties of Xilinx 12.3i module

Simulation And Results:

The simulation result for multiplication of two16 bit numbers (320 and 400) using radix-16 booth encoder multiplier is shown in the Figure 4.



```
Name        Value     1,000 ns   1,500 ns
a[15:0]     320       320
b[15:0]     400       400
mul[31:0]   128000    128000
```

Figure 4: Simulated Results of radix-16 booth encoder multiplier

Here we take multiplier as 400 and multiplicand as 320 finally we got the product term as 128000.Here we perform the multiplication of two16 bit numbers using radix-16 booth encoder multiplier. This simulation results as shown in Figure 4.

Synthesis Report:

Comparison of the radix-8 and radix-16 multiplier is carried out and results are shown in the table given below table. Here we compared Number of 4 input LUTs. The Result can be analyzed by following Table 2.

Logic utilization	**Radix 8**	**Radix 16**
Delay	52.252ns	50.053ns
Number of 4 input LUTs	1525	1115

Table 2: Comparison results of radix-8 and radix-16 booth encoder multiplier

VI. CONCLUSION

After browsing all the toil and when facing plenty of issues, we have a tendency to manage to complete the objectives of the paper that square measure to implement booth's formula for the look of a binary multiplier factor mistreatment ripple carry adder design. While using ripple carry adder we can avoid number of iteration steps and number of cycles also decreased. In any case we have a tendency to come to a conclusion that ripples Carry Adders square measure best suited to our Applications. Then we have a tendency to turn our focus into the look of Multipliers. Initial of all we have a tendency to designed a booth's radix-4 multiplier factor. If we have a tendency to comparison information between radix-8 and radix-16 booth multipliers we have a tendency to note that radix-16 consumes less power than radix-8, as a result of radix-16 uses virtually a 0.5 variety of iterations than radix-8. As radix-8 appeared a lot of appropriate for the look we have a tendency to dispensed additional analysis on radix-8 multiplier factor by mistreatment ripple carry adder design.

REFERENCES

[1] B. Millar, P.E. Madrid and E.E. Swartzlander, Jr., "A fast hybrid multiplier combining Booth and Wallace/Dadda algorithms," Proc. of the 35th IEEE Midwest Symposium on Circuits and Systems, pp.158-165, Aug. 1992.

[2] C.S. Wallace, "A suggestion for fast multipliers," IEEE Trans. Electron. Comput.,Feb. 1964.

[3] A.D. Booth, "A signed binary multiplication technique," Quarterly J. Mechan. Appl. Math., vol IV. Part 2, 1951.

[4] O.L. MacSorley, "High speed arithmetic in binary computers," Proc. IRE, Jan. 1961.

[5] H. Sam y A. Gupta, "A generalized multi bit recoding of two's complement Binary numbers and its proof with application in multiplier implementations," IEEE Trans. Comput., Aug. 1990,pp.1006-1015.

[6] T.G. Noll, "Carry-save architectures for high-speed digital signal processing."Journal of VLSI Signal Processing, 1991, pp. 121-140.

[7] M.R. Santoro, "A pipelined 64´64 iterative array multiplier," Proc. Dig. Tech. Paper Int. Solid- State Circ. Conf., Feb. 1988.