Design Of High Performance Configurable Radix-4 Booth Multiplier Using Cadence Tools

Dr.T. Esther Rani
CVR College of Engineering, Dept of ECE, Hyderabad, India
estherlawrenc@gmail.com

***Abstract* — Fast multipliers are crucial in digital signal processing systems. The speed of multiply operation is of great importance in digital signal processors and general purpose processors especially since the media processing took off. As the need for efficient design is increasing without compromising the performance, industry has to concentrate on the tradeoffs. Here, a modified Booth multiplier is implemented using an algorithm that reduces the number of partial Products to be generated using the fastest multiplication algorithm. In this work, 8X8 multipliers with maximum range of input from -128 to +127 and negative numbers represented in 2's complement form can be used. Booth Encoder i.e., Partial Product Generator and Hybrid adder are used for the design of modified booth multiplier to achieve minimum delay and less area.**

***Index Terms*— Modified Booth multiplier, digital signal processors, Booth Encoder, multiplication algorithm, Hybrid adder.**

I. INTRODUCTION

Multiplier is a key component of any high performance system such as FIR filters, microprocessors, digital signal processors, etc. Performance of a system is generally determined by the performance of the multiplier as the multiplier is the element with slow operation and consumes more area in any system. Therefore, optimizing the speed and area of a multiplier is a major design issue. As the area and speed are usually conflicting constraints, whole spectrum of multipliers with different area-speed constraints will be designed in parallel. The multipliers with such design constraints have moderate performance in both speed and area. Radix 2^n multipliers which operate on digits in a parallel fashion instead of bits bring the pipelining to the digit level and avoid most of' the problems was introduced by M. K. Ibrahim in 1993. These structures are iterative and modular and the pipelining done at the digit level brings the benefit of constant operation speed irrespective of the size of the multiplier.

A high speed low-power multiplier is proposed adopting the new modified Booth implementing approach. The modified booth encoder will reduce the number of partial products generated by a factor of 2. Power dissipation is recognized as a critical parameter in modern VLSI design. Dynamic power dissipation which is the major part of total power dissipation is due to the charging and discharging capacitance in the circuit. Dynamic power dissipation is calculated by $P_d = \alpha C_L V^2 f$. Power reduction can be achieved by reduction of output Capacitance $C_{L,}$ power supply voltage V, switching activity α and clock frequency f.

In most computers, the operand usually contains the same number of bits. When the operands are interpreted as integers, the product is generally twice the length of operands in order to preserve the information content. This repeated addition method that is suggested by the arithmetic definition is slow that it is almost always replaced by an algorithm that makes use of positional representation. It is possible to decompose multipliers into two parts. The first part is dedicated to the generation of partial products, and the second one collects and adds them.

Booth multiplication allows for smaller, faster multiplication circuits through encoding the signed numbers to 2's complement, which is also a standard technique used in chip design, and provides significant improvements by reducing the number of partial product to half compared to conventional multiplication techniques. Multipliers are categorized relative to their applications, architecture and the way the partial products are produced and summed up. Based on all these, a designer might decide the type of multiplier.

A. Basic Binary Multiplier

A multiplier circuit is able to perform a multiplication of n-bitsXn-bits at a high speed by increasing the speed of the forming process of the partial products so that the delay time may be inhibited from increasing for a large n, and which can inhibit the chip size becoming large. Multiplication is more complicated than addition, being implemented by shifting as well as addition. Because of the partial products involved in most multiplication algorithms, more time and more circuit area is required to compute, allocate, and sum the partial products to obtain the multiplication result. Pencil-and-paper algorithm for multiplication is explained below.

10.32377/cvrjst0611

Multiplicand: 0010 -- Stored in register r1

Multiplier: x 1101 -- Stored in register r2

Partial Prod 0010 -- No shift for LSB of Multiplier
" " 0000 -- 1-bit shift of zeroes (can omit)
" " 0010 -- 2-bit shift for bit 2 of Multiplier
" " 0010 -- 3-bit shift for bit 3 of Multiplier
---------- Zero-fill the partial products and ad
PRODUCT 0011010 -- Sum of all partial products stored in r3

B. Partial Product Generation

Since the amount of hardware and the delay depends on the number of partial products to be added, this may reduce the hardware cost and improve performance by considering different methods. Straightforward extensions of the Booth recoding scheme can further reduce the number of partial products, but require a time consuming N-bit carry propagate addition before any partial product generation can take place. Figure 1shows the basic unsigned multiplication using dot notation.

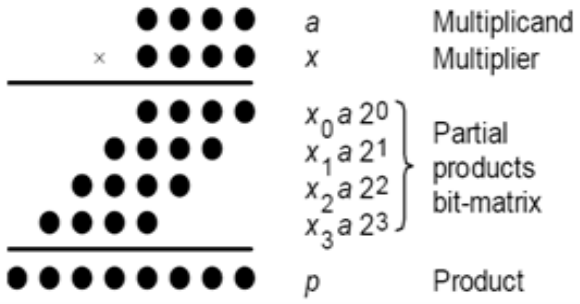

Figure 1: Basic unsigned Multiplication using Dot notation

The number of dots in the partial product section of the dot diagram proportional to the amount of hardware required to sum the partial products and form the final product. Time multiplexing can reduce the hardware requirement, at the cost of slower operation. The latency of an implementation of a particular algorithm is also related to the height of the partial product section (i.e., the maximum number of dots in any vertical column) of the dot diagram. This relationship can vary from logarithmic to linear or tree implementations where interconnect delays are significant. However, in a real implementation there will more be interconnect delay due to the physical separation of the common inputs of each AND gate, and distribution of the multiplicand to the selection elements.

C. Fast Adders

Fast carry propagate adders are important to high performance multiplier design in two ways. First, an efficient and fast adder is needed to make any "hard" multiples that are needed in partial product generation. Second, after the partial products have been summed in a redundant form, a carry propagate adder is needed to produce the final non redundant product. Half adder and full adder are two types of single bit adders. The full adder takes into account a carry input such that multiple adders can be used to add larger numbers. Many researchers reported on the multiplier architectures including array, parallel and pipelined multipliers that have been pursued and the pipelining is the most widely used technique to reduce the propagation delays of digital circuits.

i. Carry Look-Ahead Adder (CLA)

The concept behind the CLA is to get rid of the rippling carry present in a conventional adder design. The rippling of carry produces unnecessary delay in the circuit. For a conventional adder the expressions for sum and carry signal can be written as follows.

S =A xor B xor C…….....*(1)*
$C0 = AB + BC + AC$…….....*(2)*

It is useful from an implementation perspective to define S and *C0* as functions of some intermediate signals G (generate), D (delete) and P (propagate). G =1 implies that a carry bit will be generated, P=1 implies that an incoming carry will be propagated to C0 as shown in figure 2. These signals are computed as

$G = AB$…….....*(3)*
$P = A$ xor B……..*(4)*

Also S and C0 in terms of G and P can be expressed as

$C\,0(G,P) = G + PC$…….*(5)*
$S(G,P) = P$ *xor* C……*(6)*

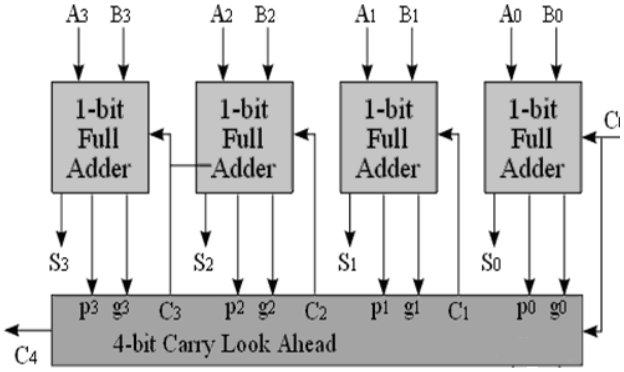

Figure 2: 4-bit Carry Look Ahead Adder

ii. Hybrid Adder

Hybrid Adder is a combination of any two adders and is used in high speed applications. The proposed hybrid adder consists of two carry look ahead adders and a multiplexer as shown in figure 3. Adding two n-bit

numbers with a hybrid adder is done with two adders (therefore two carry look ahead adders) in order to perform the calculation twice, one time with the assumption of the carry being zero and the other assuming one. After the two results are calculated, the correct sum, as well as the correct carry, is then selected with the multiplexer once the correct carry is known. The propagation delay is less for hybrid adder and at the same time it occupies larger area compared to the other adders.

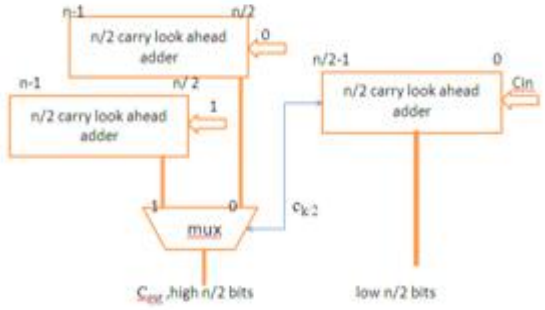

Figure 3: n-bit Hybrid Adder

II. DESIGN OF RADIX-2 AND RADIX-4 BOOTH MULTIPLIER

This section describes an 8x8 signed parallel Booth multiplier which reduces the number of the partial products. Modified booth algorithm is used and for adding partial products efficiently. A 16-bit hybrid adder has been implemented for generating the final result. After deciding on the multiplier architecture, different logic styles for multiplier implementation have been compared and concluded that Booth Multiplier is the most efficient multiplier in terms of power and delay.

One of the solutions for realization of high-speed multipliers is to enhance parallelism and to decrease the number of subsequent calculation stages. It is well known that both Modified Booth algorithm and the hybrid adder are effective in decreasing number of stages.

A. Booth Multiplication Algorithm for Radix-2

Booth algorithm gives a procedure for multiplying binary integers in signed 2's complement representation.
The booth algorithm is explained with the following example:
Example, 2 x - 4
0010 (2) * 1100 (-4)

Step 1: Making the Booth table
I. From the two numbers, pick the number with the smallest difference between a series of consecutive numbers, and make it a multiplier.
i.e., 0010 -- From 0 to 0 no change, 0 to 1 one change, 1 to 0 another change, and so there are two changes on this one
1100 -- From 1 to 1 no change, 1 to 0 one change, 0 to 0 no change, so there is only one change on this one.
Therefore, multiplication of 2 x (– 4), where 2 (0010_2) is the multiplicand and (– 4)

(1100_2) is the multiplier.

II. Let X = 1100 (multiplier)
Let Y = 0010 (multiplicand)
Take the 2's complement of Y and call it –Y
–Y = 1110

III. Load the X value in the table.
IV. Load 0 for X-1 value it should be the previous first least significant bit of X
V. Load 0 in U and V rows which will have the product of X and Y at the end of operation.
VI. Make four rows for each cycle; this is because four bits numbers are multiplied.

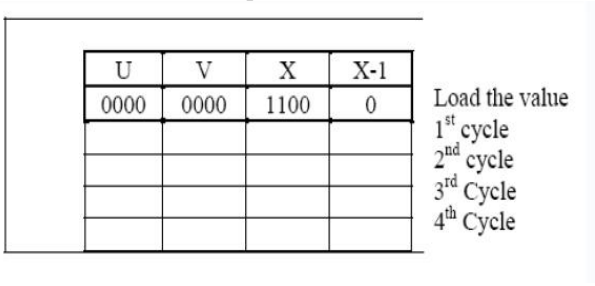

Figure 4: 4-bit Radix-2 Booth multiplication example

Step 2: Booth Algorithm

Booth algorithm requires examination of the multiplier bits, and shifting of the partial product. Prior to the shifting, the multiplicand may be added to partial product, subtracted from the partial product, or left unchanged according to the following rules.

The first least significant bits of the multiplier "X" is observed, and the previous least significant bits of the multiplier "X - 1". Table 1shows the Radix-2 Booth Encoding.

TABLE I
RADIX-2 BOOTH ENCODING TABLE

X	X-1	Partial product
0	0	Shift only
0	1	Add Y to U, and shift
1	0	Subtract Y from U, and shift or add (-Y) to U and shift
1	1	Shift only

II Take U & V together and shift arithmetic right shift which preserves the sign bit of 2's complement number. Thus a positive number remains positive, and a negative number remains negative.

10.32377/cvrjst0611

III Shift X circular right shifts because this will prevent us from using two registers for the X value.

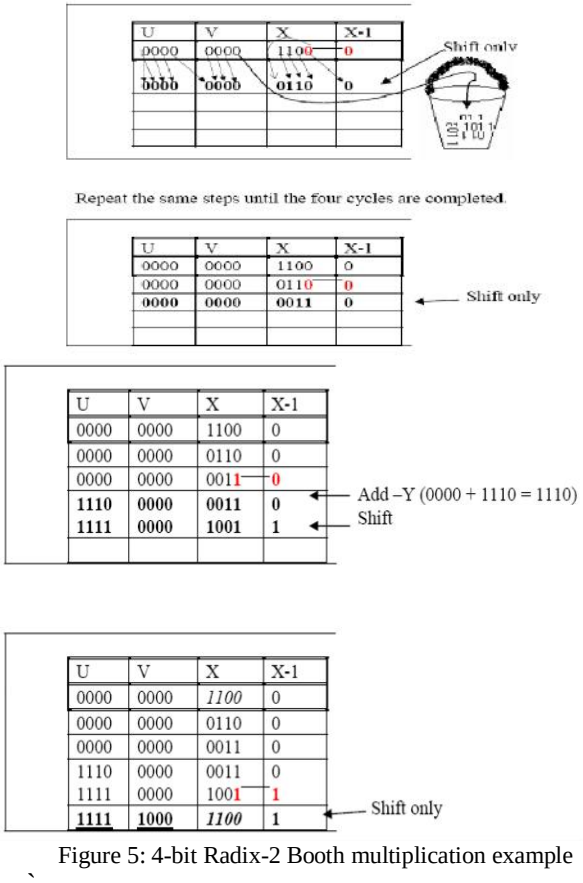

Figure 5: 4-bit Radix-2 Booth multiplication example

`

After finishing four cycles, the answer is shown in the last rows of U and V which is: 11111000_2

By the fourth cycle, the two algorithms have the same values in the product register.

B. Booth Multiplication Algorithm for Radix-4

One of the solutions of realizing high speed multipliers is to enhance parallelism which helps to decrease the number of subsequent calculation stages. The original version of the Booth algorithm (Radix-2) had two drawbacks that, the number of add subtract operations and the number of shift operations becomes variable and becomes inconvenient in designing parallel multipliers. Also the algorithm becomes inefficient when there are isolated 1's. These are overcome by using modified Radix-4 Booth algorithm which scans strings of three bits with the algorithm given below:

1) Extend the sign bit 1 position if necessary to ensure that n is even.

2) Append a 0 to the right of the LSB of the multiplier.

3) According to the value of each vector, each Partial Product will he 0, +y , -y, +2y or -2y.

The negative values of y are made by taking the 2's complement .The multiplication of y is done by shifting y by one bit to the left. Thus, in any case, in designing a n-bit parallel multipliers, only n/2 partial products are generated. The block diagram of radix-4 booth multiplier is as shown in figure 6.

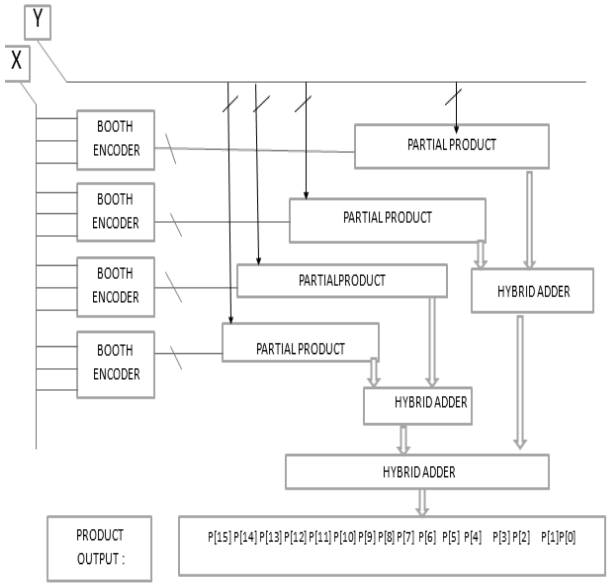

Figure 6: Block Diagram of Radix-4 Booth Multiplier

C. Booth Encoder

It will encode the multiplicand based on multiplier bits. In Radix -4 compare 3 bits at a time with overlapping technique. Grouping starts from the LSB, and the first block only uses two bits of the multiplier and assumes a zero for the third bit. There are two inputs for booth encoder one is multiplicand and the other is 3 bits from multiplier, based on these two inputs it will encode the multiplicand. For 8 bit multiplier the no of blocks and partial products will be 4. The function of booth encoder is as shown in the table2 below.

TABLE II
RADIX-4 BOOTH ENCODING TABLE

BLOCK	PARTIAL PRODUCT
000	0
001	1*multiplicand
010	1*multiplicand
011	2*multiplicand
100	-2*multiplicand
101	-1*multiplicand
110	-1*multiplicand
111	0

D. Hybrid Adder

Multiplication of two numbers is carried out in two steps.

1. Generating partial products: Generation of partial products is done by booth encoder.
2. Adding the partial products: To produce output, all the partial products must be added. This is done by high speed hybrid adder. Hybrid adder is a combination of carry look ahead adder and carry select adder.

III. IMPLEMENTATION OF BOOTH MULTIPLIERS

This design flow would start with a behavioral-style description of the system, written in Verilog, and would take the following steps. Simulation at the high level behavioral model used to confirm that the conceptualized design does function correctly. Logic synthesis describes the design as an interconnection or netlist of logic gates and flip-flops. Technology mapping maps the gates from logic synthesis into the standard cells in the library. At the end of the technology mapping step, physically what each component in our system looks like and what interconnections need are known, but physically what the interconnections will look like were not specified. Placement and Routing takes the standard cell netlist as an input, and produces a full layout.

CAD tools, like Cadence, are used to automate these steps as much as possible. The Cadence Incisive® NC Simulator has been used to simulate the design at the behavioral level. The Cadence Encoutner® RTL Compiler global synthesis has been used to produce the logic synthesis of the design and map it to the required technology. The Cadence SoC Encounter™ RTL-to-GDSII system places the design and routes it to produce the final layout.

IV. RESULTS AND ANALYSIS

The results of the booth multiplier are verified using simulation process. As the main aim of the work is to decrease the propagation delay and multiplier performance depends on the performance of adders, Multiplication of two numbers is carried out in two steps: Generation of partial products has been performed by booth encoder and addition of the partial products. The simulation diagram of Radix-2 Booth multiplier is as shown in the figure 7. The synthesized netlist of the Radix-2 Booth multiplier has one main module in which carry select adder is present.the synthesized netlist of main module is as shown in figure 8. The synthesized netlist of the carry select adder for radix-2 Booth multiplier is as shown in figure 9.

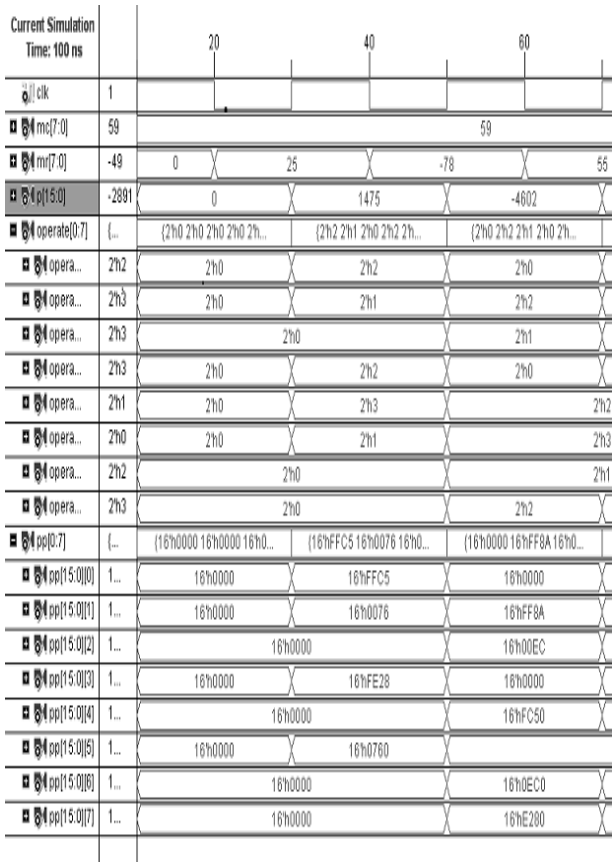

Figure 7: Simulation Diagram of Radix-2 Booth multiplier

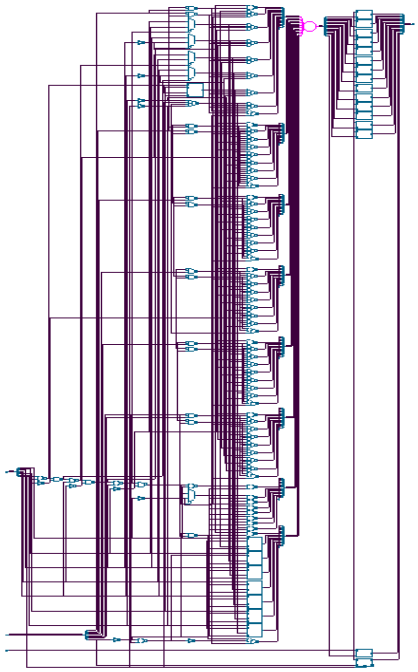

Figure 8: synthesized netlist of Radix-2 Booth multiplier

10.32377/cvrjst0611

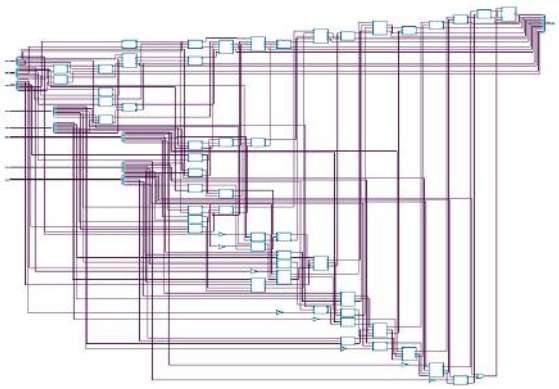

Figure 9: synthesized netlist of carry select adder for radix-2 Booth multiplier

By changing the timing delay values in the script the power and area of the radix-2 Booth multiplier are observed. For different values of timing delays the corresponding slack, power and area are as shown the table3.

TABLE III
DELAY, POWER and AREA for RADIX-2 BOOTH MULTIPLIER

Delay (ps)	Slack (ps)	Power (mW)	Area (um^2)
3000	6	1.42	7504
3500	1	1.33	7548
4000	9	1.18	6686
4200	29	1.17	6603
4500	110	1.15	6470

The output of the SoC Encounter that is final layout design of Radix-2 Booth multiplier and is shown below in figure 10.

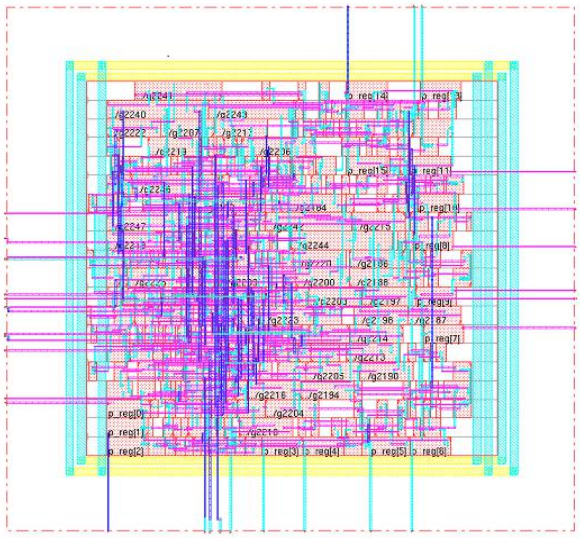

Figure 10:Final layout design of Radix-2 Booth multiplier.

The simulation diagram of modified Booth multiplier is as shown in the figure11. The synthesized netlist of the Radix-4 Booth multiplier has one main module in which carry select adder is present.the synthesized netlist of main module is as shown in figure 12. The synthesized netlist of the carry select adder for modified Booth multiplier is as shown in figure 13.

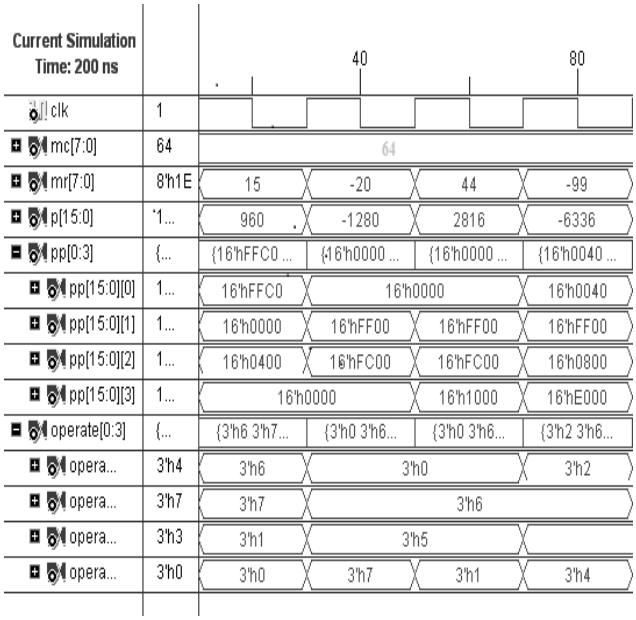

Figure 11: Simulation Diagram of modified Booth multiplier.

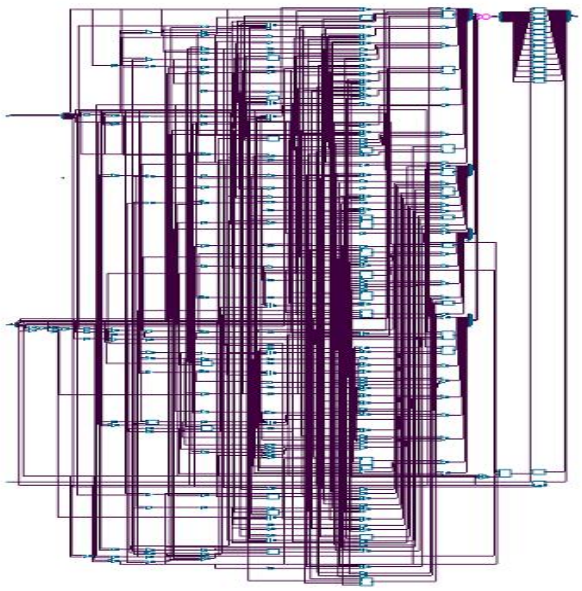

Figure 12:synthesized netlist of modified Booth multiplier

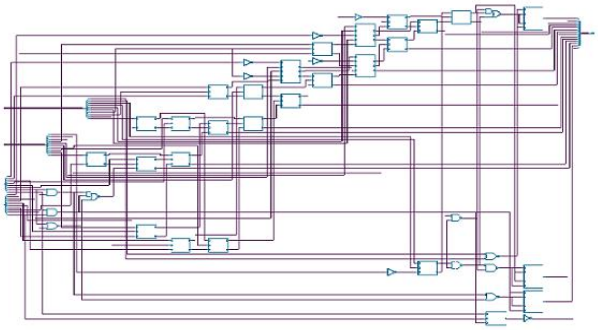

Figure 13: Synthesized netlist of carry select adder for modified Booth multiplier.

By changing the timing delay values in the script the power and area of the modified Booth multiplier are observed. For different values of timing delays the corresponding slack,power and area are as shown the table 4.

TABLE IV
DELAY, POWER and AREA for MODIFIED BOOTH MULTIPLIER

Delay (ps)	Slack (ps)	Power (mW)	Area (um^2)
3000	2	1.12	6749
3500	2	0.98	5911
4000	17	0.955	5582
4200	20	0.944	5565
4500	23	0.9	5545

The final layout design of the modified Booth multiplier for length to width ratio equal to one is shown in figure 14.

Figure 14: Final layout design of modified Booth multiplier.

For a fixed delay in timing , the power and area of the radix-2 Booth multiplier and Modified Booth multiplier are as shown in the table 5.

TABLE V
COMPARISION OF POWER AND AREA OF BOTH THE MULTIPLIERS FOR FIXED TIMING DELAY

	Radix-2 Booth multiplier	Modified Booth multiplier	% change in the value
Delay (ps)	3000	3000	0
Power(mW)	1.42	1.12	21.126
Area (um^2)	7504	6749	10.06

From the above table it is observed that for same delay the power and area of the modified Booth multiplier are less compared to Radix-2 Booth multiplier.

For a fixed area, the power and delay of the radix-2 Booth multiplier and Modified Booth multiplier are as shown in the table 6.

TABLE VI
COMPARISION OF POWER AND DELAY OF 2 MULTIPLIERS FOR FIXED AREA

	Radix-2 Booth multiplier	Modified Booth multiplier	% change in the value
Delay (ps)	4000	3000	25
Power(mW)	1.2	1.12	6.67
Area (um^2)	6700	6700	0

The table 7 is valid for 8 bit x 8 bit multiplier. The table 7 shows advantage of radix-4 compared to radix-2. It has less propagation delay and at the same time it occupies lesser area.

TABLE VII
PERFORMANCE OF THE BOOTH MULTIPLIERS

	Radix-2 Booth multiplier	Modified (Radix-4) Booth multiplier	% change in the value
Number of slices	88	68	22.7
Number of LUTs	156	126	19.23
Path Delay (ns)	24.22	17.1	29.39

The above table is valid for 8 bit x 8 bit multiplier. The above table shows why to move from radix-2 to radix-4 . The main advantage of using radix-4 is it has less propagation delay, i.e speed and at the same time it occupies lesser area.

10.32377/cvrjst0611

Using NC Simulator tool, 8 bit modified Booth multiplier at the behavioral level is coded in a behavioral description using Verilog. It is compiled, checked for syntax errors, elaborated and simulated using a Verilog test bench using the following commands to get the output as shown in figure 15.

Figure 15: Output waveform in NC simulator.

In RTL compiler tool, the mapping of a design into a netlist of standard cells is done. The library containing all these standard cells is usually built according to the fabrication process limitations. The workshop uses a 0.5 um Silicon-on-insulator process manufactured by Peregrine semi conductors. The output is a verilog file describing the netlist. At this point, the number of gates used and the propagation delay of the critical path of the system is reported.

Once the script file is ready, the graphical user interface of the Encoutner RTL Compiler is used to source the script file and synthesize the design of the 8 bit modified Booth Multiplier. The following commands are used to open the graphical user interface of the Encoutner RTL Compiler. Then the final synthesized design is obtained as shown in figure 16 below.

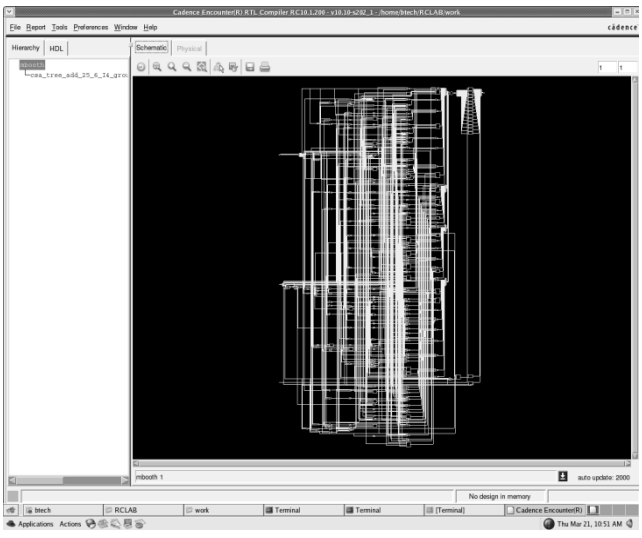

Figure 16: Encounter RTL compiler output after synthesis.

SoC Encounter is used for the placement and routing of the design. In this tool first import the design, specify the floor plan, add the power rails, place the standard cells, and route the design. SoC Encounter output after placing standard cells and after routing are shown in figures 17 and 18.

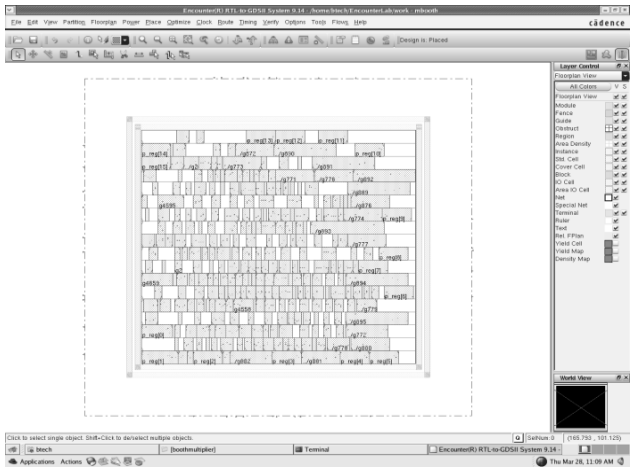

Figure 17: SoC Encounter window after placing standard cells

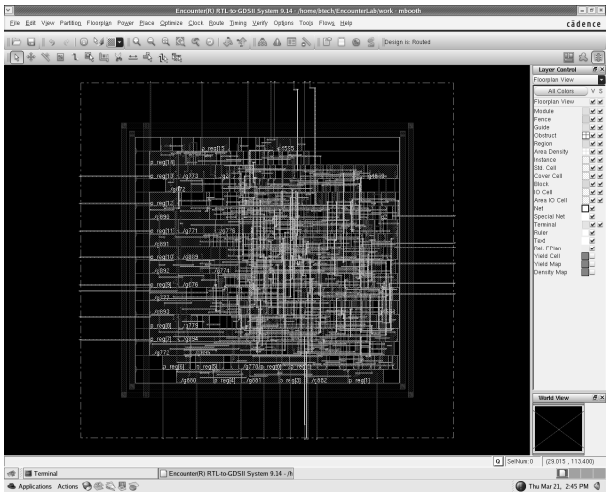

Figure 18: SoC Encounter window after routing

V Conclusion

In this work, 8x8 Radix-4 modified Booth Multiplier has been presented. Modest improvements in area and power over more conventional algorithms have been shown using this algorithm. Algorithms based upon the Booth partial product method are distinctly superior in power and area when compared to non-Booth encoded methods. This result must be used carefully if applied to other technologies, since different trade-offs may apply. The main advantage of using Radix-4 is that it has less propagation delay, and it occupies lesser area. This work is also implemented by using the cadence tools of NC simulator, Encounter RTL compiler and SoC Encounter for simulation, logical synthesis and placing & routing respectively.

REFERENCES

[1] Digital Design Principles and Practices – John F. Wakerly, PHI/ Pearson Education Asia 3rd Ed.,2005.

[2] Computer System Architecture – M. Morris Mano, 3rd Edition, PHI/ Pearson., 2006.

[3] K.H.Chen and Y.S.Chu , "A low power multiplier with spurious power suppression technique" ,IEEE Trans. Very Large Scale Integr .(VLSI)Syst., Vol.15, no-7, pp846-850, July 2007.

[4] Essentials of VLSI circuits and systems – Kamran Eshraghian, Eshraghian Dougles and A.pucknell, PHI, 2005 Edition.

[5] CMOS VLSI Design: A Circuits and Systems Perspective, Third Edition, Neil H.E. Weste, David Harris.

[6] John Rabaey **"**Digital Integrated Circuits", PHI, 1st edition, 1999

[7] http://www.xilinx.com/

[8] Jack Horgan, "Low Power Soc Design", EDAWeekly Review May 17 - 21, 2004

[9] Cadence, "Low Power in EncounterTM RTL Compiler", Product Version 5.2, December 2005

[10] Cadence, "Cadence Low Power Design Flow"

[11] Cadence, "Low Power Application Note for RC 4.1 and SoCE 4.1 USR3", Version 1.0,1/14/2005

10.32377/cvrjst0611