ISSN: 2278 – 1323
International Journal of Advanced Research in Computer Engineering & Technology
Volume 1, Issue 5, July 2012

Implementation of Radix-4 Multiplier with a Parallel MAC unit using MBE Algorithm

Bodasingi Vijay Bhaskar bhaskar748@gmail.com
Valiveti Ravi Tejesvi raviteja433@yahoo.com
Reddi Surya Prakash Rao prakashece_48@reddifmail.com

***Abstract*— A radix-4/-8 multiplier is implemented using modified booth multiplier encoder that demand high speed and low energy operation. Depending on the input pattern, the multiplier operates in the radix-8 mode in 56% of the input cases for low power, but reverts to the radix-4 mode in 44% of the slower input cases for high speed. The performance of the radix-8 multiplier is bottlenecked due to the occurrence of the 3B term in computing the partial products. So for computing the partial product in this case we select the radix-4 mode. It is a good approach if we implement the multiplier as a hybrid architecture of the radix-4/-8 because the radix-8 mode has low power consumption capability, occupying less area and the main advantage is that the number of partial products obtained in this mode are less(N/3) compared to the partial products of the radix-4 mode(N/2). But the detection of the 3B term while computing the partial products is very difficult and it is difficult to implement it on the FPGA board. So by comparing the performances of the two multipliers we suggest to go with the radix-4 multiplier and it is implemented here. In this paper we are implementing the radix-4 multiplier along with the MAC(Multiplier and Accumulator) unit for computing the signal values in real time applications. The carry save adder block is selectively activated to reduce power consumption. This is implemented on XC3S200 FPGA. It consists of 3840 LUT's of 4-input out of which only 550 LUT's are used and also it consists of 173 IOB's out of which only 66 are used. This project is implemented on Xilinx XC3S200 FPGA and is simulated using VHDL and synthesized using Xilinx 12.1.**

***Index Terms*— FPGA, Modified Booth Algorithm, Radix multiplier, VHDL.**

I. INTRODUCTION

Multipliers are the essential components in real time signal processing and also for all the multimedia applications. Many previous works were done in implementing high-speed multiplier to reduce power consumption [2]. This is due to the increased demand for portable multimedia applications which require low power consumption as well as high speed operation.

However low-power multipliers without any consideration for high-speed are not the appropriate solutions of low-energy embedded signal processing for multimedia applications [1] [4]. Previously, a hybrid radix-4/-8 modified Booth encoded (MBE) multiplier was proposed for low-power and high-speed operation. This multiplier architecture had separate radix-4 and radix-8 Booth encoders. Both encoders were operated regardless of the input patterns [2], resulting in power and area overhead. Therefore although its power consumption was reduced compared to the radix-4 architecture, its critical path delay was considerably increased. As a result, its energy efficiency was not improved over conventional radix-4 or radix-8 architectures [2].

We propose a multiplier that computes the partial product and parallely performs the accumulation of the partial products. This multiplier operates on the Modified Booth Encoder algorithm and the Wallace tree in radix-4 mode . In the majority of the input cases, the radix-8 architecture is as fast as radix-4 architecture while consuming less power. However, in the remaining input cases, the radix-8 architecture is bottlenecked by the generation of the ±3B partial product term, which requires an additional carry propagation adding stage. Therefore, we use radix-4 multiplier, which is faster at its operation and the cost of power increases. The structure of the proposed multiplier is illustrated in figure 1.[8]

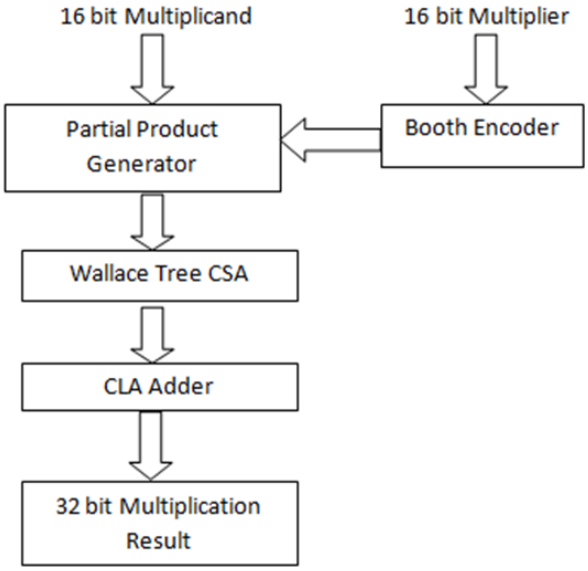

Fig1: Architecture of Radix-4 multiplier

II. CONVENTIONAL RADIX-4/RADIX-8 MULTIPLIER

The modified Booth encoded multiplier has two types, the radix-4 multiplier architecture and the radix-8 multiplier architecture. The difference between the two architectures, summarized in Table I, results from the difference in the number of bits of the input operand that are encoded. A radix-4 multiplier encodes 2 bit-segments of the input operand while the radix-8 encodes 3 bit-segments of the input operand. As a result, the radix-8 architecture has a smaller number of partial products and thus lower power consumption. However, it is slower than the radix-4 architecture due to the higher complexity of the partial product generation stage.

N x N multiplier	Radix-4 multiplier	Radix-8 multiplier
Speed	**Fast**	Slow
Number of Partial product	N/2	**N/3**
Power comsumption	High	**Low**
Utility	**Wide**	Limited
Area	Large	**Small**

TABLE I. COMPARISON BETWEEN RADIX-4 AND RADIX-8

In a radix-8 multiplier, the performance bottleneck

occurs only when the encoding result contains ±3B terms. Table II shows all possible encoding results. In Table II**,** the probability that radix-8 Booth encoding multiplier generates ±3B terms is only four out of sixteen cases. If it is possible to encode partial product in a different way when ±3B term occurs, power consumption can be reduced remarkably with only one third of partial products compared to radix-4's partial product generation method. Moreover an adding step to produce ±3B term is unnecessary therefore the propagation delay of proposed hybrid multiplier is same as a radix-4 multiplier. So by considering this statement we implement the multiplier in radix-4 mode.

Inputs				Partial Products
Xi+2	Xi+1	Xi	Xi-1	Ppi
0	0	0	0	0
0	0	0	1	B
0	0	1	0	B
0	0	1	1	2B
0	1	0	0	2B
0	**1**	**0**	**1**	**3B**
0	**1**	**1**	**0**	**3B**
0	1	1	1	4B
1	0	0	0	-4B
1	**0**	**0**	**1**	**-3B**
1	**0**	**1**	**0**	**-3B**
1	0	1	1	-2B
1	1	0	0	-2B
1	1	0	1	-B
1	1	1	0	-B
1	1	1	1	0

TABLE II. RADIX-8 PARTIAL PRODUCT GENERATION TABLE (AXB)

III. PROPOSED RADIX-4 MULTIPLIER WITH MAC UNIT

There are three techniques are existing in multiplication.

(i) Binary Multiplication

(ii) Array Multiplication

(iii) Multiplier and Accumulator Unit

In this paper we mainly concentrate on Multiplier and Accumulator Unit.

A. Multiplier and Accumulator Unit

The inputs for the MAC are to be fetched from memory location and fed to the multiplier block of the MAC, which will perform multiplication and give the result to adder which will accumulate the result and then will store the result into a memory location. This entire process is to be achieved in a single clock cycle. The architecture of the MAC unit which had been designed in this work consists of one 16 bit register, one 16-bit Modified Booth Multiplier, 32- bit accumulator. To multiply the values of A and B, Modified Booth multiplier is used instead of conventional multiplier because Modified Booth multiplier can increase the MAC unit design speed and reduce multiplication complexity.

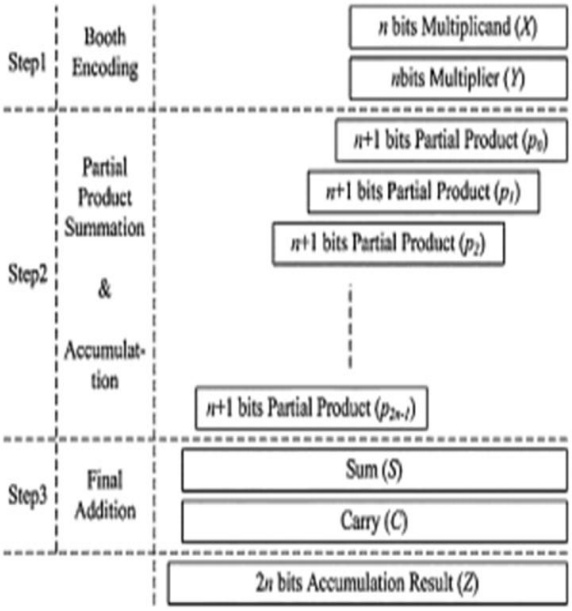

Fig 2**:** Multiplication And Accumulation operation

B. Proposed MAC Architecture

If an operation to multiply two N-bit numbers and accumulate into a 2N-bit number is considered, the critical path is determined by the 2N-bit accumulation operation. if a pipeline scheme is applied for each step in the standard design of Fig ,the delay of last accumulator must be reduced in order to improve the performance of the MAC. The overall performance of the proposed MAC is improved by eliminating the accumulator itself by combing it with CSA function.if the accumulator has been eliminated, the critical path is then determined by the final adder in the multiplier.The basic method to improve the performance of final adder is to decrease the no of input bits.Inorder to reduce these no input bits,the multiple partial products are compressed into a sum and carry by CSA.the number of bits of sums and carries to be transferred to the final adder is reduced by adding lower bits of sums and carries inadvance within the range in which the overall performance will not be degraded.

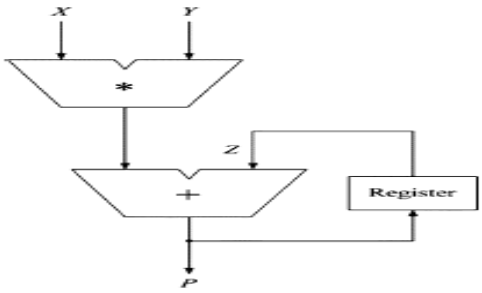

Fig 3: MAC Architecture in hardware

C. Booth Encoder

x_i	x_{i-1}	Operation	Comments	y_i
0	0	shift only	string of zeros	0
1	1	shift only	string of ones	0
1	0	subtract and shift	beginning of a string of ones	$\bar{1}$
0	1	add and shift	end of a string of ones	1

Properties:

- Multiplication starts from least significant bit
- If started from most significant bit - longer adder/subtractor to allow for carry propagation
- No need to generate recoded SD multiplier (requiring 2 bits per digit)
- Bits of original multiplier scanned - control signals for adder/subtractor generated
- Booth's algorithm can handle two's complement multipliers
- If unsigned numbers multiplied - 0 added to left of multiplier (xn=0) to ensure correctness

Booth's Recoding Algorithm:

Booth multiplication [7] is a technique that allows for smaller, faster multiplication circuits, by recoding the numbers that are multiplied. It is the standard technique used in chip design, and provides significant improvements over the "long multiplication" technique.

Shift and Add

A standard approach that might be taken by a novice to perform multiplication is to "shift and add", or normal "long multiplication". That is, for each column in the multiplier, shift the multiplicand the appropriate number of columns and multiply it by the value of the digit in that column of the multiplier, to obtain a partial product. The partial products are then added to obtain the final result:

```
      0 0 1 0 1 1
      0 1 0 0 1 1
      0 0 1 0 1 1
    0 0 1 0 1 1
  0 0 0 0 0 0
0 0 0 0 0 0
0 0 1 0 1 1
0 0 1 1 0 1 0 0 0 1
```

With this system, the number of partial products is exactly the number of columns in the multiplier.

Reducing the Number of Partial Products

It is possible to reduce the number of partial products by half, by using the technique of radix 4 Booth recoding. The basic idea is that, instead of shifting and adding for every column of the multiplier term and multiplying by 1 or 0, we only take every second column, and multiply by ±1, ±2, or 0, to obtain the same results. So, to multiply by 7, we can multiply the partial product aligned against the least significant bit by -1, and multiply the partial product aligned with the third column by 2:

Partial Product 0 = Multiplicand * -1, shifted left 0 bits (x -1)

Partial Product 1 = Multiplicand * 2, shifted left 2 bits (x 8)

This is the same result as the equivalent shift and add method:

Partial Product 0 = Multiplicand * 1, shifted left 0 bits (x 1)

Partial Product 1 = Multiplicand * 1, shifted left 1 bits (x 2)

Partial Product 2 = Multiplicand * 1, shifted left 2 bits (x 4)

Partial Product 3 = Multiplicand * 0, shifted left 3 bits (x 0)

The advantage of this method is the halving of the number of partial products. This is important in circuit design as it relates to the propagation delay in the running of the circuit, and the complexity and power consumption of its implementation. It is also important to note that there is comparatively little complexity penalty in multiplying by 0, 1 or 2. All that is needed is a multiplexer or equivalent, which has a delay time that is independent of the size of the inputs. Negating 2's complement numbers has the added complication of needing to add a "1" to the LSB, but this can be overcome by adding a single correction term with the necessary "1"s in the correct positions.

Radix-4 Booth Recoding

To Booth recode the multiplier term, we consider the bits in blocks of three, such that each block overlaps the previous block by one bit. [4] Grouping starts from the LSB, and the first block only uses two bits of the multiplier (since there is no previous block to overlap):

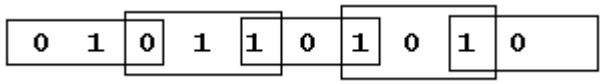

Fig: Grouping of bits from the multiplier term, for use in Booth recoding.

The least significant block uses only two bits of the multiplier, and assumes a zero for the third bit. The overlap is necessary so that we know what happened in the last block, as the MSB of the block acts like a sign bit. We then consult the table 2-3 to decide what the encoding will be.

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

Table III: Booth recoding strategy for each of the possible block values.

Since we use the LSB of each block to know what the sign bit was in the previous block, and there are never any negative products before the least significant block, the LSB of the first block is always assumed to be 0. Hence, we would recode our example of 7 (binary 0111) as :

0 1 1 1
block 0 : 1 1 0 Encoding : * (-1)
block 1 : 0 1 1 Encoding : * (2)

In the case where there are not enough bits to obtain a MSB of the last block, as below, we sign extend the multiplier by one bit.

0 0 1 1 1
block 0 : 1 1 0 Encoding : * (-1)
block 1 : 0 1 1 Encoding : * (2)
block 2 : 0 0 0 Encoding : * (0)

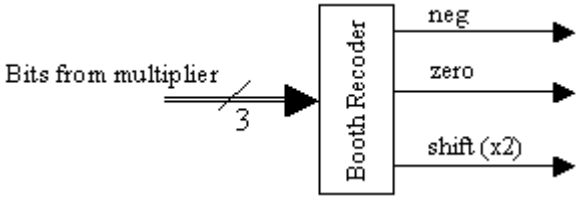

Fig 4: Booth Recoder and its inputs and outputs.

In figure 4,

- The zero signal indicates whether the multiplicand is zeroed before being used as a partial product
- The shift signal is used as the control to a 2:1 multiplexer, to select whether or not the partial product bits are shifted left one position.
- Finally, the neg signal indicates whether or not to invert all of the bits to create a negative product (which must be corrected by adding "1" at some later stage)

The described operations for booth recoding and partial product generation can be expressed in terms of logical operations if desired but, for synthesis, it was found to be better to implement the truth tables in terms of VHDL case and if/then/else statements.

- Bits xi and xi-1 recoded into yi and yi-1 - xi-2 serves as reference bit
- Separately - xi-2 and xi-3 recoded into yi-2 and yi-3 - xi-4 serves as reference bit
- Groups of 3 bits each overlap - rightmost being x1 x0 (x-1), next x3 x2 (x1), and so on

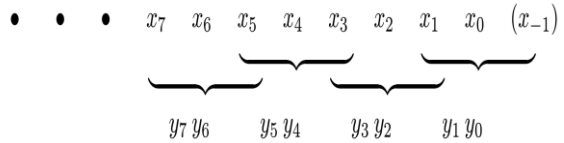

Fig: Grouping of bits from the multiplier term

- To Booth recode the multiplier term, we consider the bits in blocks of three, such that each block overlaps the previous block by one bit. Grouping starts from the LSB, and the first block only uses two bits of the multiplier (since there is no previous block to overlap):

Steps for computing the multiplication of 8 and 20

- Pad LSB with 1 zero

Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	0

- n is even then pad the MSB with two zeros

0	0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	0

- Form 3-bit overlapping groups for n=8 we have 5 groups

0	0	0	0	0	1	0	1	0	0	0

0	0	Y7	Y6	Y5	Y4	Y3	Y2	Y1	Y0	0

- Determine partial product scale factor from modified booth 2 encoding table.

0	0	0	0	0	1	0	1	0	0	0

Groups			Coding
0	0	0	0xY
0	1	0	1xY
0	1	0	1xY
0	0	0	0xY
0	0	0	0xY

- Compute the multiplicand multiples

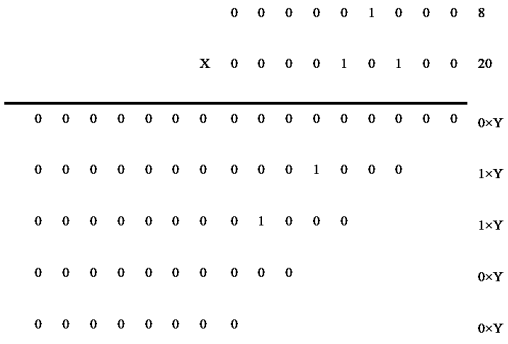

Fig 5: Multiplication using modified Booth encoding

D. Carry Save Adder

The architecture of the hybrid-type CSA that complies with the operation of the proposed MAC is shown in Fig. 5, which performs 8X 8-bit operation.

- A + B => S
- Save carries A + B => S, C_{out}
- Use C_{in} A + B + C => S1, S2 (3# to 2# in parallel)

Used in combinational multipliers by building a Wallace Tree It was formed based on (12).

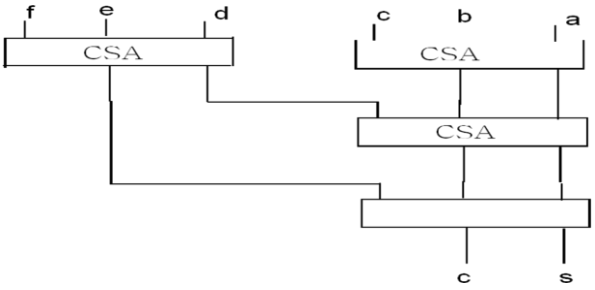

Fig 6: Carry Save Adder

ADVANTAGES

- Booth multipliers save costs (time and area) for adding partial products
- With the higher radix the number of additions is reduced and the redundant Booth code reduces costs for generating partial products in a higher radix system.
- Low power consumption is there in case of radix 4 booth multiplier because it is a high speed parallel multiplier.

APPLICATIONS

- Multimedia and communication systems,
- Real-time signal processing like audio signal processing, video/image processing, or large-capacity data processing.
- The multiplier and multiplier-and-accumulator (MAC) are the essential elements of the digital signal processing such as filtering, convolution, and inner products.

IV. Experimental Result

In this project we are evaluating the performance of the high speed low power MAC using Radix-4 algorithm in VHDL coding. The power consumption details are obtained by synthesizing in XILINX 12.1 version.

When the reset value is active high then the multiplicand multiplies the multiplier in radix-4 mode.

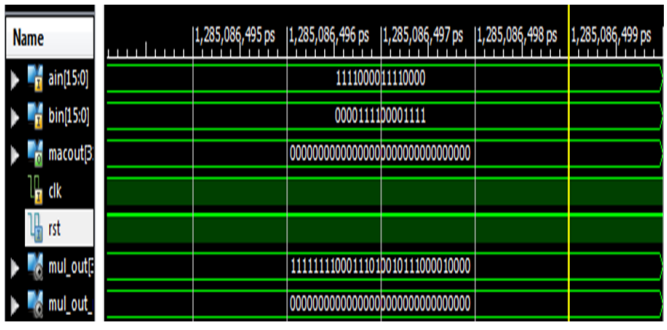

Fig 7: Multiplier output without MAC operation.

When the reset value is active low then the MAC operation is started in the accumulator unit in radix-4 mode.

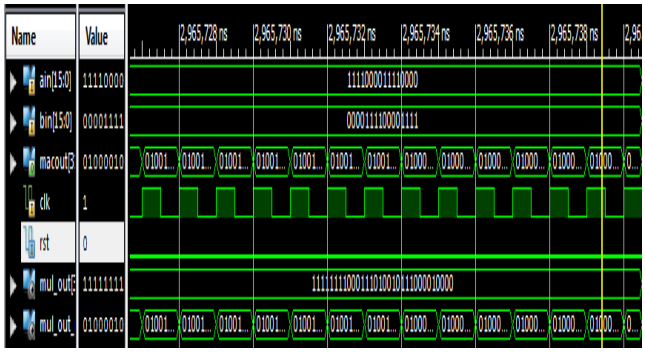

Fig 8: Multiplier output with MAC operation

V. Conclusion

Our project gives a clear concept of different multiplier. We found that the parallel multipliers are much option than the serial multiplier. We concluded this from the result of power consumption and the total area. In case of parallel multipliers, the total area is much less than that of serial multipliers. Hence the power consumption is also less. This is clearly depicted in our results. This speeds up the calculation and makes the system faster.

Multipliers are one the most important component of many systems. So we always need to find a better solution in case of multipliers. Our multipliers should always consume less power and cover less power. So through our project we try to determine which of the three algorithms works the best. In the end we determine that Radix-4 multiplier architecture which is proposed in this paper is an appropriate solution for portable multimedia applications because it is both low-power and high-speed.

References

[1] B.S. Chekauer and E.G. Friedman, "A Hybrid Radix-4/Radix-8 Low 75Power Signed Multiplier Architecture", IEEE Trans. vol. 44, no.8, pp656-659, Aug. 1997.

[2] A.P. Chandrakasan, S.Sheng, and R.W. Broderson, "Low-power CMOS digital design", IEEE J. Solid-State Circuits, vol.27, pp. 473-483, Apr. 1992.

[3] G. N. Sung, Y. J. Ciou, and C. C. Wang, "A power-aware 2-dimensional bypassing multiplier using cell-based design flow," IEEE International Symposium on Circuits and Systems, May 2008

[4] Shiann-Rong Kuang, and Jiun-Ping Wang, "Modified Booth Multipliers With a Regular Partial Product Array", IEEE Trans. vol. 56, no.5, pp404-408, May. 2009.

[5] B. Millar, P.E. Madrid, and E.E. Swartzlander, Jr., "A fast hybrid multiplier combining Booth and Wallace/Dadda algorithms," in Proc.35th IEEE Midwest Symp. Circuits Syst., Aug. 1992, pp. 158–165.D

[6] J. Fadavi-Ardekani, "M × N booth encoded multiplier generator using optimized Wallace trees," IEEE Trans. Very Large Scale Integr. (VLSI) Syst., vol. 1, no. 2, pp. 120–125, Jun. 1993.

[7] www.wikipedia.com for Booth multiplication algorithm

[8] www.googleimages.com for the block diagram of radix-4

Bodasingi Vijay Bhaskar has received his B.Tech degree in Electronics and Communication Engineering at GMRIT, JNTU Kakinada. He has received M.Tech degree in VLSI System Design from St. Theressa Institute of Engineering and Technology JNTU Kakinada. He had an experience of 11 years in total in which he serviced as HOD of ECE dept., for 5 years. Currently working as I/C Principal at Avanthi's St. Theressa Institute of Engineering and Technology JNTU Kakinada. Fields of interest is VLSI, DSP, and Communication Systems.

Valiveti Ravi Tejesvi has received B.Tech degree in Electronics and Communication Engineering at St. Theressa Institute of Engineering and Technology, JNTU Kakinada during 2010. Currently working on M.Tech degree in VLSI System Design from Avanthi's St. Theressa Institute of Engineering and Technology JNTU Kakinada. Fields of interest is VLSI, Communication Systems, and DSP.

Reddi Surya Prakash Rao has received B.Tech degree in Electronics and Communication Engineering at St. Theressa Instituteof Engineering and Technology, JNTU Kakinada during 2005 and M.Tech degree in DECS from HINDU College, JNTU Hyderabad. He had an experience of 5 years in total. Fields of interest is Communication Systems, DSP.