Advance in Electronic and Electric Engineering.
ISSN 2231-1297, Volume 3, Number 6 (2013), pp. 683-690
© Research India Publications
http://www.ripublication.com/aeee.htm

Implementation of Modified Booth Algorithm (Radix 4) and its Comparison with Booth Algorithm (Radix-2)

Sukhmeet Kaur[1], Suman[2] and Manpreet Signh Manna[3]

[1]*ECE, SSIET (P.T.U), Derabassi, Punjab, India.*
[2]*ECE, SSIET, Derabassi, Punjab, India.*
[3]*EIE, SLIET (Deemed University), Longowal, Sangrur, India.*

Abstract

This paper describes implementation of radix-4 Modified Booth Multiplier and this implementation is compared with Radix-2 Booth Multiplier. Modified Booth's algorithm employs both addition and subtraction and also treats positive and negative operands uniformly. No special actions are required for negative numbers. In this Paper, we investigate the method of implementing the Parallel MAC with the smallest possible delay. Parallel MAC is frequently used in digital signal processing and video/graphics applications. A new architecture of multiplier - and accumulator (MAC) for high speed arithmetic by combining multiplication with accumulation and devising a carry-look-ahead adder (CLA), the performance is improved. Modified Booth multiplication algorithm is designed using high speed adder. High speed adder is used to speed up the operation of Multiplication. Designing of this algorithm is done by using VHDL and simulated using Xilinx ISE 9.1i software has been used and implemented on FPGA xc3s50-5pq208. (Abstract)

Keywords: Booth multiplier, Modified Booth Multiplier, CLA, VHDL.

1. Introduction

Multipliers are key components of many high performance systems such as FIR filters, Microprocessor, digital signal processors, etc.(Hsin-Lei Lin , 2004). Signed multiplication is a careful process. With unsigned multiplication there is no need to take the sign of the number into consideration. However in signed multiplication the same process cannot be applied because the signed number is in a 2's compliment form which would yield an incorrect result if multiplied in a similar fashion to unsigned multiplication. That's where Booth's algorithm comes in. Booth's algorithm preserves the sign of the result. Booth algorithm is also used for high speed multiplication as high speed digital multipliers find application in many digital signal processing. Digital multipliers are also utilized in other digital signal processors, such as radar and sonar. In many applications, processing of analog signals in digital form in real time would not be practical without high speed digital multipliers. Other applications include fast Fourier transform processors and high speed floating point arithmetic logic units (Robert C. Ghest, 1979).

Booth Multiplier reduces number of iteration step to perform multiplication as compare to Conventional steps. 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 (Y.N. Ching, 2005). More interestingly the number of additions is data dependent, which makes this algorithm.

In this paper, we are aiming to build up a Booth Encoding 8 bits Multiplier. Booth Encoding is an effective method which greatly increases the speed of our algebra. We also attempts to reduce the number of partial products generated in a multiplication process by using the modified Booth algorithm. The multiplier takes in 2 8-bits operands: the multiplier(MR) and the multiplicand (MD), then produces 16-bit multiplication result of the two as its output. The architecture comprises four parts: Complement Generator, Booth Encoder, Partial Product and Carry Look-ahead Adder as shown in Fig 1.

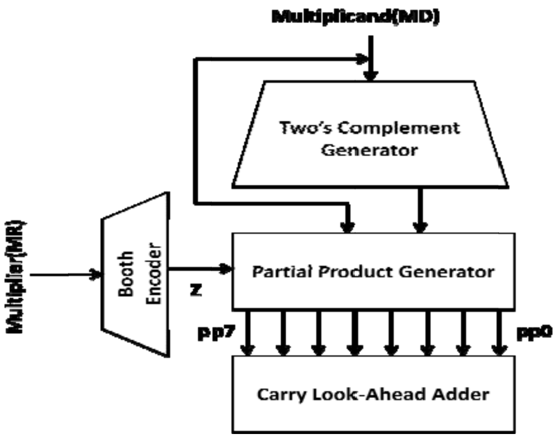

Figure 1: Architecture of Booth Multiplier.

2. Booth Multiplier(Radix-2)

The Booth algorithm was invented by A. D. Booth, forms the base of Signed number multiplication algorithms that are simple to implement at the hardware level, and that have the potential to speed up signed multiplication Considerably. Booth's algorithm is based upon recoding the multiplier, y, to a recoded, value, z, leaving the multiplicand, x, unchanged. In Booth recoding, each digit of the multiplier can assume negative as well as positive and zero values. There is a special notation, called signed digit (SD) encoding, to express these signed digits. In SD encoding +1 and 0 are expressed as 1 and 0, but -1 is expressed as 1(Vincent P. Heuring, 2003).

The value of a 2s complement integer was defined a by equation 1.

$$y = -y_{m-1}2^{m-1} + \sum_{i=0}^{m-2} y_i 2^i \tag{1}$$

This equation says that in order to get the value of a signed 2's complement number, multiply the m – ith digit by -2`-1, and multiply each remaining digit i by +2g.

For example, -7, which is 1001 in 2's complement notation, would be, in SD notation, 1001 = -8 + 0 + 0 + 1 = -7.

For implementing booth algorithm most important step is booth recoding. By booth recoding we can replace string of 1s by 0s. For example the value of strings of five 1s, 11111 = 2^5 - 1 = 1000$\bar{1}$ = 32 – 1 = 31. Hence if this number were to be used as the multiplier in a multiplication, we could replace five additions by one addition and one subtraction.

The Booth recoding procedure, then, is as follows:

1. Working from lsb to msb, replace each 0 digit of the original number with a 0 in the recoded number until a 1 is encountered.
2. When a 1 is encountered, insert a 1 at that position in the recoded number, and skip over any succeeding I's until a 0 is encountered.
3. Replace that 0 with a 1 and continue.

This algorithm is expressed in tabular form in Table 1, considering pairs of numbers, $\mathrm{y_{i-1}}$ and $\mathrm{y_i}$, and the recoded digit, $\mathrm{z_i}$, as shown in Table 1.

Table 1: Booth recoding table for radix-2.

y_i	y_{i-1}	z_{i-i}	**Multiplier Value**	**Situation**
0	0	0	0	String of 0s
0	1	1	+1	End of string of l s
1	0	1	-1	Begin string of 1s
1	1	0	0	String of 1s

Booth Algorithm:

1. Add 0 to right of LSB of multiplier and look at rightmost of multiplier to make pairing of 2 bits from right to left and mark corresponding multiplier value as shown in Fig. 1
2. 00 or 11: do nothing
3. 01: Marks the end of a string of 1s and add multiplicand to partial product (running sum).
4. 10: Marks the beginning of a string of 1s Subtract multiplicand from partial product.

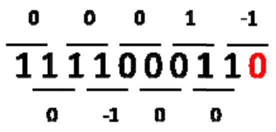

Figure 2: 2 Bit Pairing as per booth recoding.

One of the solutions realizing high speed multipliers is to enhance parallelism which helps in decreasing the number of subsequent calculation stages. The Original version of Booth's multiplier (Radix – 2) had two drawbacks.

1. The number of add / subtract operations became variable and hence became inconvenient while designing Parallel multipliers.
2. The Algorithm becomes inefficient when there are isolated 1s.

These problems are overcome by using Radix 4 Booth's Algorithm which can scan strings of three. This Booth multiplier technique is to increase speed by reducing the number of partial products by half.

3. Modified Booth Multiplier (Radix-4)

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. Radix-4 booth encoder performs the process of encoding the multiplicand based on multiplier bits. It will 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 as shown by figure 3.

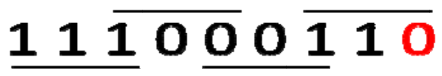

Figure 3: 3 Bit Pairing as per booth recoding.

The functional operation of Radix-4 booth encoder is shown in the Table.2.It consists of eight different types of states and during these states we can obtain the outcomes, which are multiplication of multiplicand with 0,-1 and -2 consecutively.

Table 2: Booth recoding table for radix-4.

Multiplier Bits Block			Recoded 1-bit pair		2 bit booth	
i+1	i	i-1	i+1	i	Multiplier Value	Partial Product
0	0	0	0	0	0	Mx0
0	0	1	0	1	1	Mx1
0	1	0	1	-1	1	Mx1
0	1	0	1	0	2	Mx2
1	0	0	-1	0	-2	Mx-2
1	0	1	-1	1	-1	Mx-1
1	1	0	0	-1	-1	Mx-1
1	1	0	0	0	0	Mx0

Radix-4 Booth algorithm is 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 be 0, +y, –y, +2y or –2y.

The negative values of y are made by taking the 2’s complement. The negative values of y are made by taking the 2’s complement and Carry-look-ahead (CLA) fast adders are used for addition in this paper. The multiplication of y is done by shifting y by one bit to the left. Thus, in any case, in designing n-bit parallel multipliers, only n/2 partial products are generated (Hsin-Lei Lin 2004, H.Lee 2002 ,M. Sheplie 2004).

The advantage of this method is the halving of the number of partial products. This is important in circuit design asit relates to the propagation delay in the running of the circuit, and the complexity and power consumption of its implementation.

4. Experimental Work

After analyzing the two booth multipliers, and compare their characteristics in terms of multiplication speed, no of computations required, no of hardware, we come on finding that radix 4 booth multipliers is better than Radix-2 booth multipliers. By implementing both Radix-2 & Radix -4 multiplier we analysis that Radix -4 multiplier computation speed is higher then Radix-2 . We have done the coding of both multipliers separately in VHDL & simulate it to get the accurate waveforms as output each multiplier shown in Figure 4. Also get the device utilization summary, where we get the exact no of i/p, o/p/ no of slices requirement etc for the multiplier. In Radix-4 design simulation result is same as Radix-2 scheme. Only difference between these two schemes is synthesis report. These results are given in table 3.

Table 3: Comparison Between Radix2 and Radix 4

Device Utilization Summary	**Radix 2**	**Radix 4**
Number of Slices	397	71
Number of 4 input LUTs	184	100
Number of bonded INPUT	16	16
Number of bonded OUTPUT	16	16
Macro Statistics		
# Latches	24	12
8-bit latch	24	12
# Xors	71	23
1-bit xor2	64	21
8-bit xor2	7	2
Timing Summary		
Minimum period:	5.454ns	4.750ns
Minimum input arrival time before clock	7.936ns	4.014ns
Maximum output required time after clock:	6.216ns	6.205ns

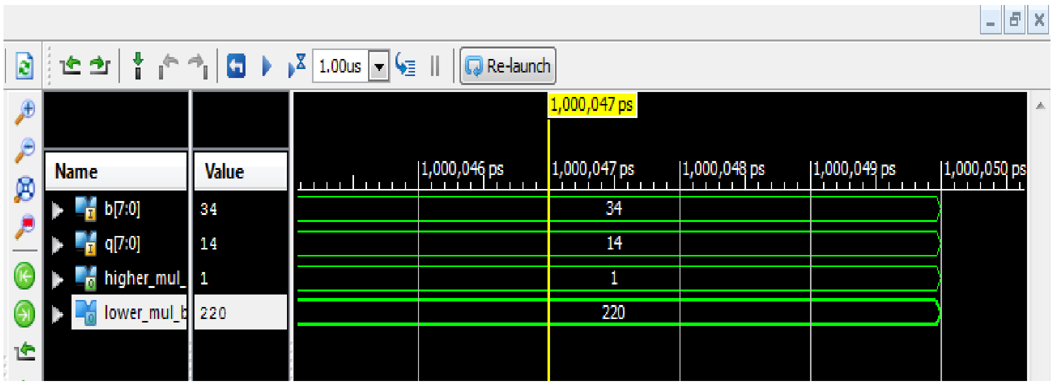

Figure 4: Output Waveform

5. Conclusion

Radix-2,4 Booth Multipliers are implemented here; the complete process of the implementation is giving higher speed of operation. The Speed and Circuit Complexity is compared, Radix-4 Booth Multiplier is giving higher speed as compared to Radix-2 Booth Multiplier and Circuit Complexity is also less as compared to it.

References

[1] Hsin-Lei Lin, Robert C. Chang, Ming-Tsai Chan (2004), Design of a Novel Radix-4 Booth Multiplier, IEEE Asia-Pacific Conference on Circuits and Systems, Vol.2, pp. 837-840.

[2] H. Lee, “A High-Speed Booth Multiplier”, ICCS, (2002).

[3] M. Sheplie (2004), High performance array multiplier, IEEE transactions on very large scale integration systems, vol. **12**, no. 3, pp. 320-325.

[4] Robert C. Ghest, Saratoga; Hua-Thye Chua, Cupertino; John M. Birkner, Santa Clara, (1979) High speed combinatorial digital multiplier, *United States Patent* ,**US 4153938 A**

[5] Vincent P. Heuring, Harry F. Jordon (2003), *Computer Systems Design and Architecture*, Pearson Education, Singapore

[6] Y.N. Ching (2005), Low-power high-speed multipliers, IEEE Transactions on Computers, vol. **54**, no. 3, pp. 355-361,