Implementation of Fused Add and Multiply Operator using Radix 8 Algorithm

K. Ramya, V. B. K. L Aruna

Abstract*— many complex arithmetic operations are based on addition and multiplication operations which are used in many DSP applications. This paper presents implementation of add-multiply operation by using conventional and fused design architectures. In the proposed paper a modified booth recoding technique for implementation of fused add-multiply (FAM) operator is introduced. By using modified booth recoding technique [2] partial products in multiplication will be reduced to half and further implementation of FAM by using radix8 booth algorithm is done which reduces the partial products further to one third of the number [1].After comparing all the architectures, the proposed radix8 booth algorithm yields considerable reductions in terms of area and critical path delay as compared to radix4 booth algorithm. The proposed design is targeted on a XILINX virtex-6 device and examined using VHDL simulator in XILINX 14.2 version.

Index Terms*— Add-multiply operator, Modified booth recoding, Partial products and Radix8 algorithm.

I. INTRODUCTION

DSP applications are widely used in many modern consumer electronics, communications etc. Typical DSP applications are mainly based on arithmetic calculations. In order to increase the performance of DSP applications there is a need to reduce the complexity in arithmetic calculations. So for that purpose Multiply-Accumulator (MAC) and Multiply-Add (MAD) units were introduced instead of conventional ones. Except the MAC/MAD operations, many DSP applications are based on Add-Multiply (AM) operations. This paper discusses about the design of Add-Multiply (AM) operation. One way to design AM operation is first allocate one adder for adding two numbers and then drive the output of adder as input to the multiplier which increases both delay and area. To reduce the area here introduces a fused design for add-multiply operation.

In this fused design adder and encoding block will be fused in one block and also the direct recoding of the sum of two numbers in its modified booth(S-MB) [3] form leads to a more efficient implementation of the fused Add-Multiply (FAM) unit compared to the conventional design. For multiplication operation modified booth algorithm is introduced in both conventional and fused designs. In modified booth algorithm grouping of bits will be done according to their radix and after grouping based on modified booth encoding table partial products will be generated. Those partial products will be added by using carry save adder (CSA) tree and carry-look-ahead adder (CLA). After implementation of add-multiply operator using radix4, radix8 booth algorithms comparisons have been done.

This paper is organized as follows: Section II discusses about the motivation and presents a technical background for the implementation of FAM units. In Section III, the proposed radix8 booth algorithm for FAM Operator is presented. In Section IV, experimental evaluations are done between the proposed algorithm and the previous algorithm in terms of area complexity, critical path latency and throughput. Section V concludes the work.

II. MOTIVATION AND BACKGROUND

A. Motivation

This paper, focuses on AM units which implement the operation OUT = (A+B).X. In conventional design (Fig. 1(a)) the inputs A and B are driven to an adder and then Y=A+B and X are driven as inputs to a multiplier in order to get OUT. But the conventional design leads to increase in critical path delay and area, in order to reduce the area FAM (Fig.1 (b)) is introduced [3]. In FAM design the adder block and the booth encoding block are fused in one block and also the direct recoding of the sum of two numbers in its MB form leads to a more efficient implementation of the fused Add-Multiply (FAM) unit compared to the conventional design. So there is one adder at the end.

B. Background

S-MB Recoding technique:

Initially for efficient addition operation (A+B), here introduces recoding schemes S-MB1in (1, 2), S-MB2 in (3, 4), S-MB3 in (5, 6). By using any one of the schemes, implementation of the addition operation is done. Sum and carry equations for recoding schemes [3] are

S-MB1:

Carry = (a or (not (b))) and (c or (a and (not (b)))) (1)

Sum =a xor b xor c (2)

S-MB2:

Carry =(c and (a xor b) (3)

Sum =a xor b xor c (4)

S-MB3:

Carry =(c or (not (a xor b))) (5)

Sum =a xor b xor c (6)

Revised Version Manuscript Received on July 04, 2015.

K. Ramya, Department of ECE, Velagapudi Ramakrishna Siddhartha Engineering College, Vijayawada (Andhra Pradesh), India.

V B K L Aruna, Department of ECE, Velagapudi Ramakrishna Siddhartha Engineering College, Vijayawada (Andhra Pradesh), India.

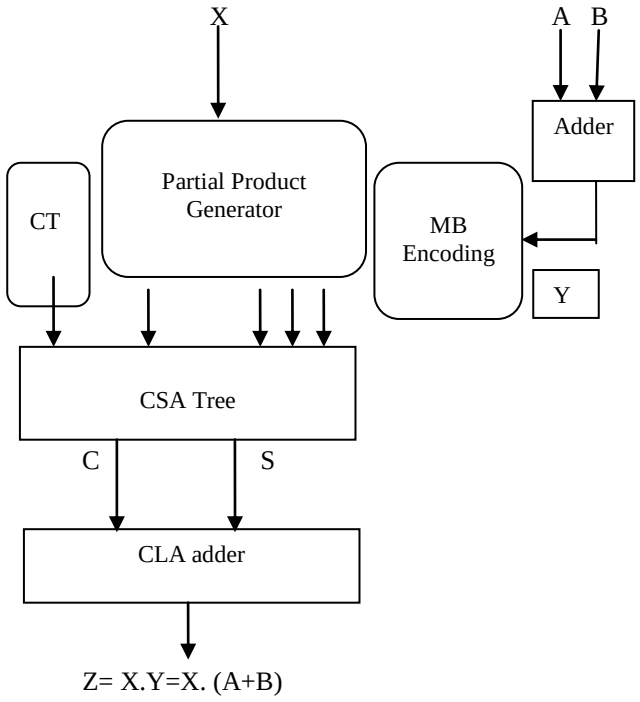


Fig.1a. AM operator based on the conventional design

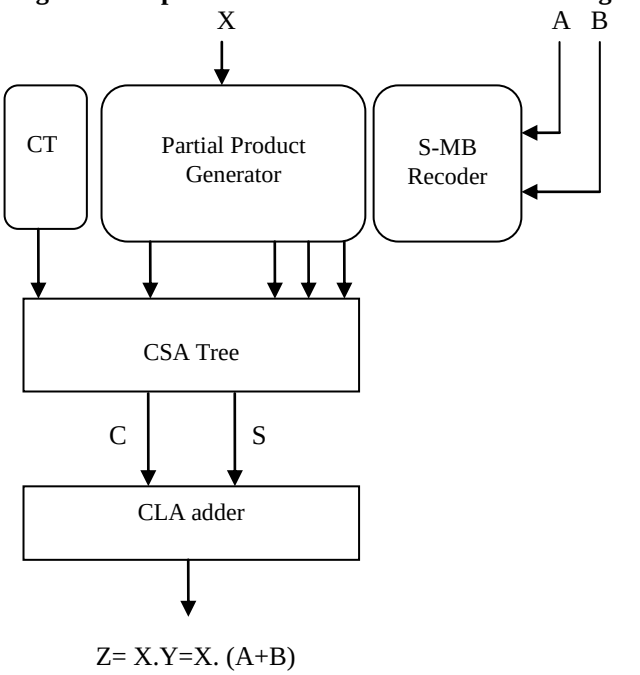
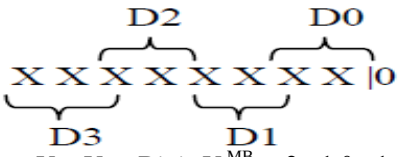


Fig.1b. AM operator based on the fused design with direct recoding of the sum of A and B in its MB representation. The multiplier is a basic parallel multiplier based on the MB algorithm.

C. Modified Booth

Modified Booth is the most common form used in multiplication [4-6]. It is a redundant radix-4 encoding technique. The main advantage of the modified booth algorithm (Radix 4) is that the number of partial products in multiplication can be reduces by half when compared to any other radix-2 representation. The grouping of bits will be done as follows in Radix4 booth algorithm.



$Y_j^{MB}=-2Y_{2j+1}+Y_{2j}+Y_{2j-1}$.Digits$Y_j^{MB}\in-2,-1,0,+1,+2, 0\le j\le k-1$ correspond to the three consecutive bits Y_{2j+1}, Y_{2j}, $Y_{2j-1}$with one bit overlapped and considering that y−1=0. Table i shows how they are formed by summarizing the modified booth. by using this booth recoding table partial products will be generated. those partial products will be added by using csa adder tree and cla.

TABLE I. Modified Booth Encoding Table

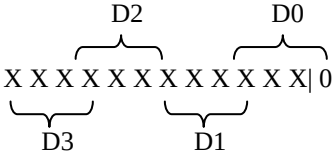
Binary			Y_j^{MB}	MB Encoding			Input Carry C_{in}, j
Y_{2j+1}	Y_{2j}	Y_{2j-1}		Sign $=s_j$	x 1= one_j	x 2= two_j	
0	0	0	0	0	0	0	0
0	0	1	+1	0	1	0	0
0	1	0	+1	0	1	0	0
0	1	1	+2	0	0	1	0
1	0	0	-2	1	0	1	1
1	0	1	-1	1	1	0	1
1	1	0	-1	1	1	0	1
1	1	1	0	1	0	0	0

III. PROPOSED RADIX 8 BOOTH ALGORITHM TO DESIGN ADD MULTIPLY OPERATION

This section discusses about the implementation of FAM using Radix8 booth algorithm. Initially for (A+B) operation afore mentioned S-MB recoding technique is used and for multiplication operation i.e. (A+B).X, radix8 booth algorithm is used.

A. Radix8 Booth Algorithm for Add-multiply Operation

Radix-8 recoding reduces the time that takes to sum up the partial products. These partial products are reduced to n/3 [1] for n bits of multiplier and multiplicand when compared to n/2 in radix-4[1]. Another advantage in radix-8 recoding is the number of transistors is less in count resulting in a reduced critical path delay and also the area is reduced when compared to radix-4.Radix-8 recoding uses the same algorithm as radix4, but instead of grouping three bits, four bits will be grouped at a time.Radix-8 encoding table [1] is shown in Table II.



Generating partial products:

1. Partial product for +2 is obtained by shifting the multiplicand to left once.
2. Partial product for +4 is obtained by shifting multiplicand left twice.
3. Partial product for -1 is obtained by performing 2's complement on multiplicand.
4. Partial product for -2 is obtained by shifting -1 once to left.
5. Partial product for -4 is obtained by shifting -1 left twice.
6. Partial product for +3 is obtained by adding +2 and +1.
7. Partial product for -3 is obtained by adding -2 and -1

TABLE II. Radix-8 Encoding Table

Quartet value	Signed-digit value
0000	0
0001	+1
0010	+1
0011	+2
0100	+2
0101	+3
0110	+3
0111	+4
1000	-4
1001	-3
1010	-3
1011	-2
1100	-2
1101	-1
1110	-1
1111	0

After generating partial products group of half adders and full adders are used for their addition.

IV. EXPERIMENTAL EVALUATION

In this section, the performance comparisons are made between the proposed algorithm and architecture with the previous architectures explored in section II. Here includes different booth algorithms in an Add-Multiply operator (Fig. 1(b)) and implemented them using VHDL, targeted on a Xilinx virtex6 device. The results of the implementations are verified by performing simulation with some random input vectors. In order to evaluate designs, the area, critical path latency and throughput are compared. The performance results for the designs in conventional, FAM using radix4 booth algorithm, FAM using radix8 booth algorithm are shown in Table III.

A. Explanation

In the below simulation results the operation OUT = (A+B).X is illustrated. Fig.2 explains add-multiply operation for 8-bit inputs. Fig.3 explains the operation of add-multiply for 16-bit inputs. Here in Fig.2, a and b are the 8 bit inputs for adder and then the output signal y from the adder is also 8 bit which is fed as input to the multiplier and the other input for multiplier is x, finally the output for add-multiply operation of 16 bit length is generated i.e. fop[15:0]. In Fig.3, a and b are the 16 bit inputs to an 16-bit adder and then the output of 16 bit length is produced which is fed as input to the multiplier and the other input for multiplier is x of 16 bit length, finally the output of 32 bit length is generated i.e. fop[31:0].

B. Simulation results

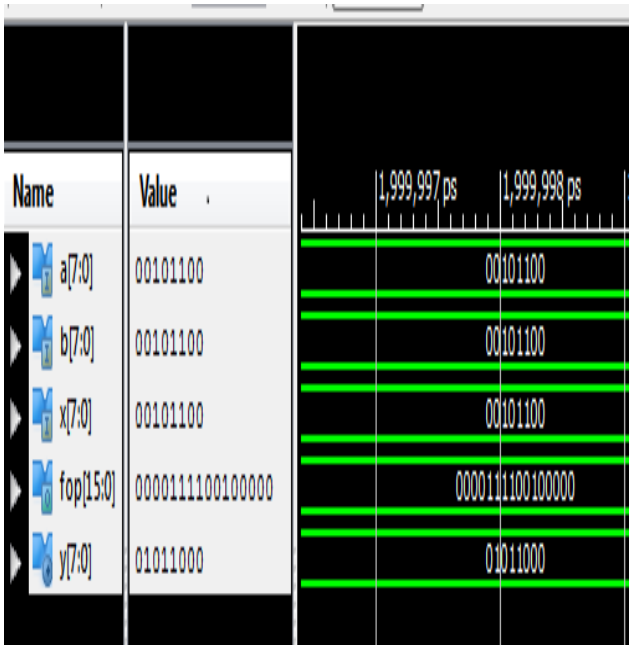


Fig.2. Final result for 8-bit add-multiply operation

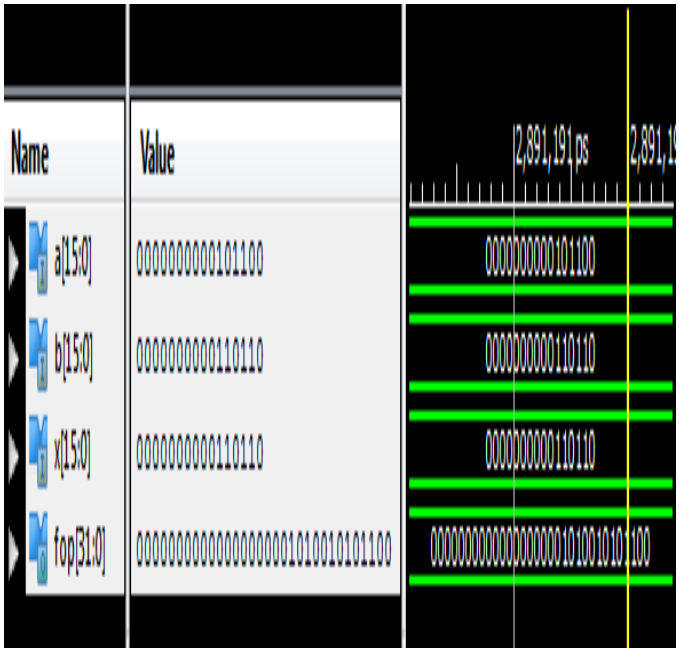


Fig.3. Final result for 16-bit add-multiply operation

From the below table, observed that the proposed Radix 8 algorithm which is used for the implementation of fused add-multiply operator yields less area and latency as compared to the previous Radix 4 algorithm.

TABLE III. Fused Add-multiply (FAM) Comparisons

8-bit			
	Conventional	**FAM+ Radix4**	**FAM+ Radix8**
Area(LUTs)	155	154	146
Latency(ns)	14.732	15.242	10.048
Throughput (1/ns)	0.067	0.065	0.099
16-bit			
	Conventional	**FAM+ Radix4**	**FAM+ Radix8**
Area(LUTs)	317	316	301
Latency(ns)	17.088	14.673	9.890
Throughput (1/ns)	0.058	0.068	0.101

V. CONCLUSION

This paper discuss about optimizing the design of the Fused-Add Multiply (FAM) operation. For this propose a high speed Radix8 booth algorithm is proposed for implementation of Fused add-multiply operation. Further the simulation results of proposed techniques were compared and the proposed multiplier is optimized according to speed, generating partial products and sums. The add-multiply operator can be used in filters, Fast Fourier Transforms (FFT) and in many complex arithmetic operations.

ACKNOWLEDGMENT

The authors would like to thank the reviewers for their constructive comments.

REFERENCES

1. Paladugu Srinivas Teja,"Design of Radix-8 Booth Multiplier Using Koggestone Adder for High Speed Arithmetic Applications," EEIEJ, vol. 1, no. 1, February 2014.
2. P. S. Tulasiram and R. Seshasayanan, "Implementation of Modified Booth Recoded Wallace Tree Multiplier for fast Arithmetic Circuits," *Journal of Advanced Research in Computer Science and Software Engineering*, Volume 4, Issue 10, October 2014.
3. Kostas Tsoumanis, "An Optimized Modified Booth Recoder for Efficient Design of the Add-Multiply Operator," *IEEE transactions on circuits and systems—I: regular papers*, vol. 61, no. 4, April 2014.
4. Z. Huang, "High-Level Optimization Techniques for Low-Power Multiplier Design," Ph.D., University of California, Department of Computer Science, Los Angeles, CA, 2003.
5. Z. Huang and M. D. Ercegovac, "High-performance low-power left-to-right array multiplier design," *IEEE Trans. Comput.*, vol. 54, no. 3, pp.272–283, Mar. 2005.
6. O. L. Macsorley, "High-speed arithmetic in binary computers," *Proc.IRE*, vol. 49, no. 1, pp. 67–91, Jan. 1961.
7. Lakshmanan, m. othman, m.a.m. ali, "Design and Characterization of Parallel Prefix Adders using FPGA'S," *journal of computers*, vol. 5, no. 10, October 2012.