ECSE 548: Modified 8-bit Booth Multiplier

Marco Kassis, Aryan Mojtahedi, Dimitrios Stamoulis and Louis-Charles Trudeau
Department of Electrical and Computer Engineering, McGill University, Montreal, Canada

***Abstract*—A Multiplier is one of the most important elements of any modern processor, whether it be for the sake of pure multiplication to evaluate for arithmetic purposes or for other functions that make use of that function. It is one of the processes that consume a significant amount of processing time, since it is a series of additions, making it very important to optimize. This report presents the Modified Booth algorithm applied on an 8-bit level. The multipliers performance is then compared to that of the Baugh-Wooley Multiplier with respect to area, transistor count and delay. Integration within the MIPS core is also implemented and presented in this paper. The CAD tool used for the design process is Electric, while testbenching was done by Verilog file exports on ModelSim, while the test vectors were generated using a C program.**

***Keywords*—Multiplier; Modified Booth Algorithm; MIPS; Baugh-Wooley Multiplier.**

I. Introduction

THE multiplication of 2 given 8-bit vectors, M and N, results in a vector larger than both of them, namely 16-bits. To regularly perform this multiplication, it has to be done bitwise, generating N partial products of the same size as M, shifted and then added to compute the value of the multiple.

Accordingly, using the most basic form of the array multiplier, we realize that that the most significant bit (MSB) takes 2M-2 time units till it reaches its final output. This is a significant waste of processing time, given that the multiplier is essentially on module within the entire processor. Hence, a logical optimization is to group the partial products together to reduce their number, hence reducing the number of additions that need to be done, which would reduce the amount of time taken to compute the product.

Modified Booth algorithm [2] introduces this type of grouping based on the inputs being inserted into the multiplier, which is discussed in the upcoming Section. *Our goal is to produce a fast 8-bit multiplier, reducing both area and time delay compared to the regular Baugh-Wooley Multiplier [3]. The multiplier implements the Modified Booth Logic and is integrated within the reduced 8-bit MIPS processor [4].*

II. Booth Logic

The Booth logic depends on an important metric r, which defines the number of partial products to be grouped together. Accordingly, the radix of the system is defined to be 2r. The radix itself defines the weight by which each and every of the partial products differ from the one to the following. We hence observe that the weight of the $r = 2$, is a radix of 4, since grouping together 2 elements will result in a shift of 2 between each of the partial products which is equivalent to multiplying by 4 from one to the following. The utilization of a radix-4 system provides a reduction in the number of partial products reaching $\frac{M+1}{2}$ instead of M, which is almost half, hence a significant improvement. The size of each vector is N+1.

The radix-4 system is the one used in this project. The 2 vectors being multiplied will be referred to as X and Y for convenience within this Section. This leads to having the 4 possible values of the partial products of Y being 0, Y, 2Y and 3Y. 2Y can be easily achieved using the single shift of the partial product; however, 3Y would require an adder, eliminating the whole point of the Booth logic. Accordingly, a workaround would be to consider 3Y as Y+4Y, while 2Y would become -2Y+4Y, with the first part of the sum accounted for earlier then the second part following that. This leads to observing of 3 bits at a time, the MSB from the previous subvector taken, with the other current 2. Accordingly Table I shows the truth table for this type of implementation.

TABLE I. Booth Logic - Truth Table

x_{2i+1}	x_{2i}	x_{2i-1}	PP_i	$SINGLE_i$	$DOUBLE_i$	NEG_i
0	0	0	0	0	0	0
0	0	1	Y	1	0	0
0	1	0	Y	1	0	0
0	1	1	2Y	0	1	0
1	0	0	-2Y	0	1	1
1	0	1	-Y	1	0	1
1	1	0	-Y	1	0	1
1	1	1	-0	0	0	1

The single, double and negative signals are the selected values that signify the operation to be implemented, which is elaborated on in the encoder part of next Section. The dot diagram with sign extension is seen in Figure 1.

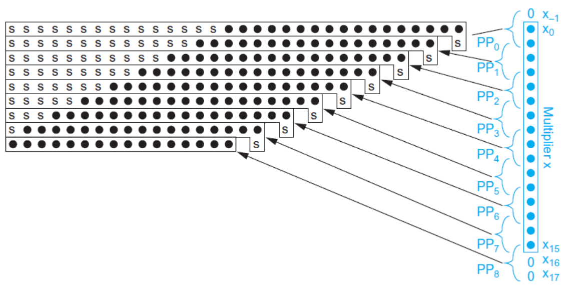

Fig. 1. Dot Diagram with Sign Extension, as presented in [4]

III. Architecture

The main logic to implement the Booth Multiplier is to be divided into several parts describing the architecture of the multiplier. These parts are: the encoder, the partial products generator, the compressor tree and the final carry propagate adder. All the parts are presented in the following Subsections, with its critical path being described at the end of each.

A. Encoder

The encoder is the circuit elements that observe the 3 bits at hand to determine what operation is to be done on the Y vector being input to the system, based on the truth table in FIGURE1. Accordingly, it is a combinational circuit design that takes as input 3 bits of multiplier X, observes them and asserts the proper signal to be used to describe the operation on Y, either keeping it as is, or doubling, with the third option of negating either of them. The circuit implemented in this project is shown in Figure 2.

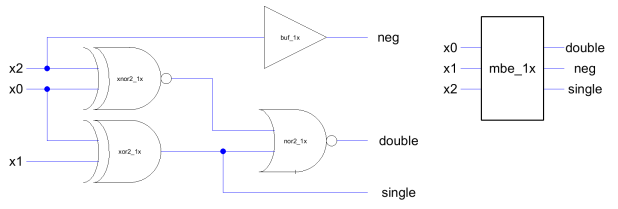

Fig. 2. Encoder Circuit

Since the X vector is 8-bits of size, taking 3 inputs at a time with the repetition of the MSB for each iteration, hence 4 bitslices of the encoder shown above would be needed to realize the logic needed, working in parallel, so delay is the computed for one bitslice only. The layout for that is in Figure 3.

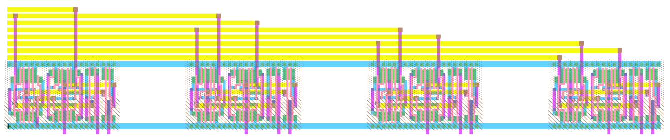

Fig. 3. Encoder Layout

TABLE II. ENCODER - CRITICAL PATH ANALYSIS

Critical Path Delay	2 XORs

B. Partial Products Generator (PPG)

The PPG is the following part of the circuitry that makes use of the data presented from the encoder signals produced. Consequently, it is the first block that takes in the Y vectors as inputs along with the Double, Single, and Negate signals to be treated, and it hence produces the partial products and sign signals to be used by the following step. The implementation of the PPG in schematic and in layout are shown in Figure 4.

TABLE III. PPG - CRITICAL PATH ANALYSIS

Critical Path Delay	2 NANDs + 1 XOR + 10 NANDs

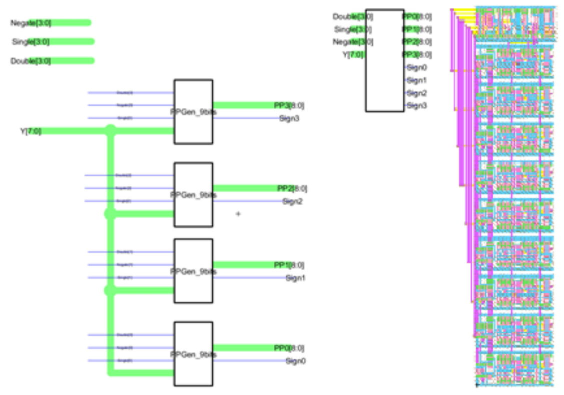

Fig. 4. PPG schematic and layout

C. Compressor Tree

The following part of the logic is the compressor tree, which computes addition using a full adder logic as shown in the bitslice in Figure 5.

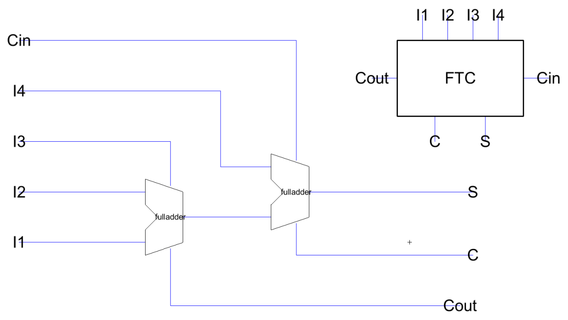

Fig. 5. Four-to-Two Compressor Circuit

The advantage of the tree architecture is that the delay through it is not as high as any other alternative implementation of the architecture, which is evident in the combined design seen in Figure 6.

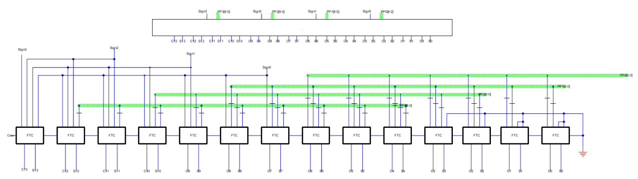

Fig. 6. Compressor Tree Desing

TABLE IV. COMPRESSOR TREE - CRITICAL PATH ANALYSIS

Critical Path Delay	2 XORs

D. Carry Propagate Adder (CPA)

This is the final stage that computes the overall value of the product of X and Y. It adds up the outputs of the following stage, with the first block having only 2, hence a half adder was implemented to save up on transistor count.

Exploiting the fact that the critical path delay of the overall design combined comes through the last block of full adder, with that critical delay being larger than that of the CPA adder itself (with a worst case delay assumed for ripple-carry adder), we realize that it is ultimately more beneficial to optimize for transistor count rather than delay. This resulted in designing a ripple-carry adder.

For the full adder blocks, multiple realizations where compared seen in Table V.

TABLE V. DIFFERENT FULL ADDERS REALIZATIONS - COMPARISON

	FA1	**FA2**	**FA3**
S_{out}	$A \otimes B \otimes C_{in}$	$(A \otimes B) \otimes C_{in}$	$\overline{C_{out}}(A + B + C) + ABC$
C_{out}	$AB + AC_{in} + BC_{in}$	$AB+C_{in}(A \otimes B)$	$AB + AC_{in} + BC_{in}$
Transistor Count/bit	32	30	28

It is seen that the FA3 design results in the least number of transistors per bit due to exploiting the readily available signal from $\overline{C_{out}}$. Luckily, it is the one available in the given standard library. Transistor level schematic is shown in Figure 7.

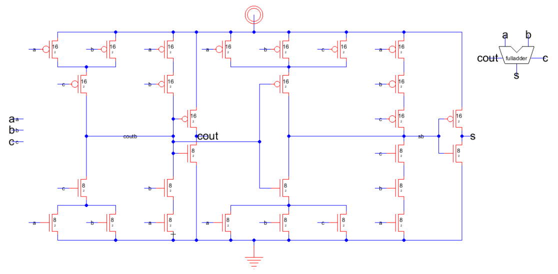

Fig. 7. Full Adder - Transistor level schematic

TABLE VI. CPA - CRITICAL PATH ANALYSIS

Critical Path Delay : CPA only	12 MAJ + 1 AND + 1 XOR
Critical Path Delay : Through overall	1 XOR

E. *Whole Design*

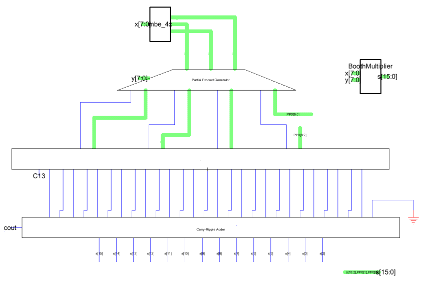

Fig. 8. Whole schematic

The whole design in Figure 8 is the schematic level, while the layout level is shown below as part of the MIPS core.

IV. MIPS INTEGRATION

In order to integrate the design into the 8-bit MIPS core, modifications to the datapath needed to be implemented along with the Finite State Machine (FSM) to define the states of the processor for the PLA controller. Accordingly, M-type command was introduced and it is defined as follows:

TABLE VII. EXTENDED MIPS ISA: M-TYPE COMMAND

opcode	**ra**	**rb**	**rd**	**XX**
6 bits	5 bits	5 bits	5 bits	XX

This new instruction set architecture is what is be used at the assembly level when calling for the multiplier function which would accordingly call on the proper pathway from the modified datapath shown in Figure 9.

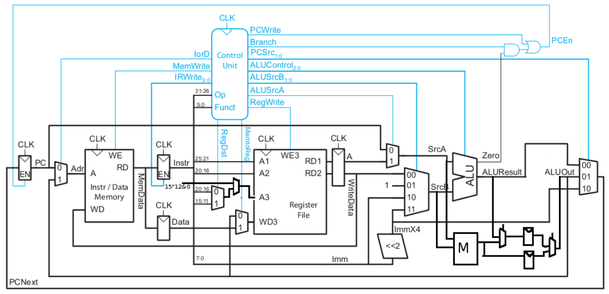

Fig. 9. Modified MIPS datapath, based on the initial desing presented in [4]

This is then also reflected in modifications for the PLA generator, which is then embedded into the controller. The layout of the entire design is then integrated within the MIPS core, with the ALU shifted to save space for the multiplier as shown in Figure 10.

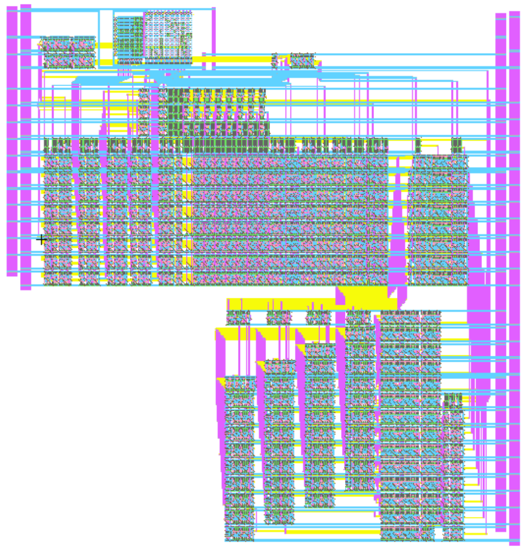

Fig. 10. Entire Desing - Layout

V. Comparison: Baugh-Wooley Multiplier

Baugh-Wooley Multiplier (BWM) is the 2's compliment option for the basic array multiplier. Figure 11 shows the schematic for it.

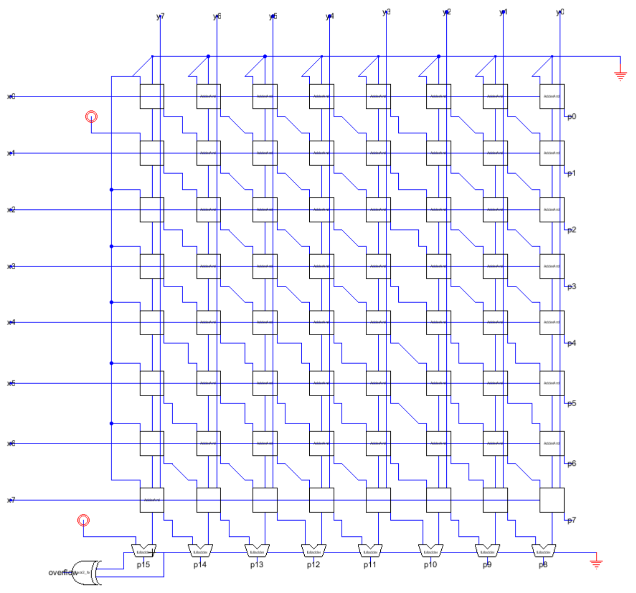

Fig. 11. BWM schematic

The main differences between BWM and the regular array multiplier is that for the blocks on the bottom and left sides, the x_i and y_i are NANDed instead of ANDed, along with a 2 V_{dd} connections as shown in the figure instead of ground. Due to regularity and similarity in all blocks, the design demonstrates compactness in the layout level as seen in Figure 12.

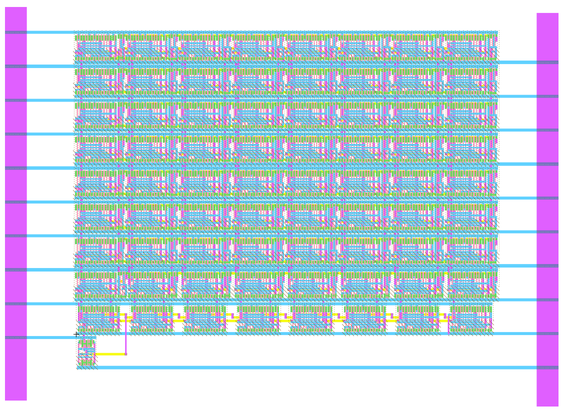

Fig. 12. BWM layout

A. 8-bit Comparison

Table VIII demonstrates that Transistor Count and Area are relatively superior for the BWM, with the Booth Multiplier superior for the Critical Path Delay.

TABLE VIII. Modified Booth & BWM: 8-bit comparison

Comparison	Modified Booth	BWM 0
Transistor Count	2886	2384
Area	$2.4M\lambda^2$	$1.3M\lambda^2$
Critical Path Delay	8 XORs + 10 ANDs + 2 NANDs	8 XORs + 9 ANDs + 8 ORs

B. 16-bit Comparison

As an additional theoretical extrapolation, the critical path delays of both multipliers were analyzed. The results are shown in the following Table IX:

TABLE IX. Modified Booth & BWM: 16-bit comparison

Comparison	Modified Booth	BWM
Critical Path Delay	9 XORs + 18 ANDs + 2 NANDs	16 XORs + 17 ANDs + 16 MAJs

A plot of the total number of gates the process needs to go through vs the number of bits is shown in Figure 13.

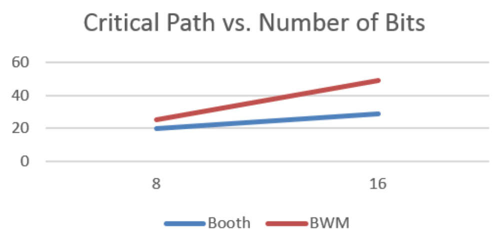

Fig. 13. BWM and Booth Multiplier Comparison: Total Number of gates

VI. Evaluation

Evaluating this, we see that the main strength of the Booth Multiplier is the smaller critical path delay compared to BWM. This is also a function of the number of bits being used; as they increase, the delay slope for BWM steeps higher than the Booth Multiplier making the latter more efficient.

VII. Testing

All designs were passed through an initial testing phase for functionality: Booth multiplier through Quartus and Array Multiplier thorugh a written Verilog code. The final designs were exported to Verilog decks and tested using ModelSim. All test vectors were completely exhaustive, covering all the 65536 different input vectors. The MIPS assembly codes were generated using MILE, a MIPS emulator [5], and they were tested using Modelsim.

References

[1] A. Booth, *A signed binary multiplication technique*, Quarterly J. Mechanics and Applied Mathematics, vol. IV, pt. 2, Jun. 1951, pp. 236-240.

[2] O. MacSorley, *High-Speed arithmetic in binary computers*, Proc. IRE, vol. 49, pt. 1, Jan. 1961, pp. 6791.

[3] M. Hatamian and G. Cash, *A 70-MHz 8-bit 8-bit parallel pipelined multiplier in 2.5-um CMOS*, JSSC, vol. 21, no. 4, Aug. 1986, pp. 505-513.

[4] Neil Weste and David Harris. *CMOS VLSI Design: A Circuits and Systems Perspective*, 4th edition, Addison-Wesley Publishing Company, USA.

[5] MILE home page, http://www.cslab.ece.ntua.gr/courses/comparch/assign.go.