VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belagavi-590018, Karnataka

Mini-Project Report
on

"8-bit Booth Multiplier"

Submitted by

USN	Name
1BI17EC089	Prathik P
1BI17EC091	R D Karthik
1BI17EC079	Pavan Kalyan B S

Under the Guidance of

Dr. KALPANA A B

Associate Professor

Department of ECE, BIT

Bengaluru-560004

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

BANGALORE INSTITUTE OF TECHNOLOGY

K.R. Road, V.V.Pura, Bengaluru-560 004

2019-20 (Odd)

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

"JnanaSangama", Belagavi-590018, Karnataka

BANGALORE INSTITUTE OF TECHNOLOGY

Bengaluru-560 004

DEPARTMENT OF ELECTRONICS & COMMUNICATION ENGINEERING

Certificate

Certified that the Mini Project work entitled **"8-bit Booth Multiplier"** carried out by

USN	NAME
1BI17EC089	Prathik P
1BI17EC091	R D Karthik
1BI17EC079	Pavan Kalyan B S

of V semester, **Electronics & Communication** Engineering branch as partial fulfillment of the course **Verilog HDL (17EC53)** prescribed by **Visvesvaraya Technological University, Belgaum** during the academic year 2019-20. It is certified that all corrections/suggestions indicated for Internal Assessment have been incorporated in the report.

The Mini Project report has been approved as it satisfies the academic requirements in respect of project work in Verilog HDL.

Dr. Kalpana A B
Associate Professor,
Department of ECE, BIT,
Bengaluru-560004

CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Introduction

Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation. The algorithm was invented by Andrew Donald Booth in 1950. Booth's algorithm is of interest in the study of computer architecture. Booth‘s Algorithm is a smart move for multiplying signed numbers. It initiate with the ability to both add and subtract there are multiple ways to compute a product. Booth‘s algorithm is a multiplication algorithm that utilizes two‘s complement notation of signed binary numbers for multiplication.

Earlier multiplication was in general implemented via sequence of addition then subtraction, and then shift operations. Multiplication can be well thought-out as a series of repeated additions. The number which is to be added is known as the multiplicand, and the number of times it is added is known as the multiplier, and the result we get is the multiplication result. After Each step of addition a partial product is generated. When the operands are integers, the product in general is twice the length of operands in order to protect the information content. This repetitive addition method that is recommended by the arithmetic definition is slow as it is always replaced by an algorithm that makes use of positional depiction.

We can decompose multipliers into two parts. The first part is committed to the generation of partial products, and the second part collects and then adds them. The fundamental multiplication principle is twofold i.e. evaluation of partial products and gathering of the shifted partial products. It is performed by the consecutive additions of the columns of the shifted partial product matrix.

1.2 Motivation

In any aspect of computing, the speed of the arithmetic unit is of great concern. Today we know that multiplier is using in every basic circuit. Today all the ALU system is based on a multipliers, complete arithmetic Logical part is based on multipliers.

CHAPTER 2

PROBLEM STATEMENT

2.1 Problem Statement

The purpose of this project is to create a 8 by 8 multiplier using Booth's multiplication algorithm. After applying Booth's algorithm to the inputs, simple addition is done to produce a final output.

2.2 Objectives

- By specifications provided for the project, the multiplier must accept 8 bit signed inputs and output a 16 bit resultant.
- The theoretical smallest and largest input values are 127 and -128. The largest output value of the multiplier is 16384 or 0b0100000000000000 and the smallest possible resultant is 16256 or 0b1100000010000000.

CHAPTER 3

SYSTEM REQUIREMENTS

3.1 Hardware requirements

A PC with a minimum specification of Intel Core i3(or equivalent) or above.

3.2 Software Requirements

Xilinx software (v2009 and above) with Modelsim Simulator.

CHAPTER 4

ARCHITECTURE

A Booth multiplier achieves a reasonable compromise on speed and size because it does not need additional supporting logic for counters such as those needed in serial or serial/parallel multipliers. Instead, the multiplication is performed by generating partial products from decoding chunks of the multiplier and then manipulating the multiplicand by negating, shifting, zeroing or applying any combination to it.

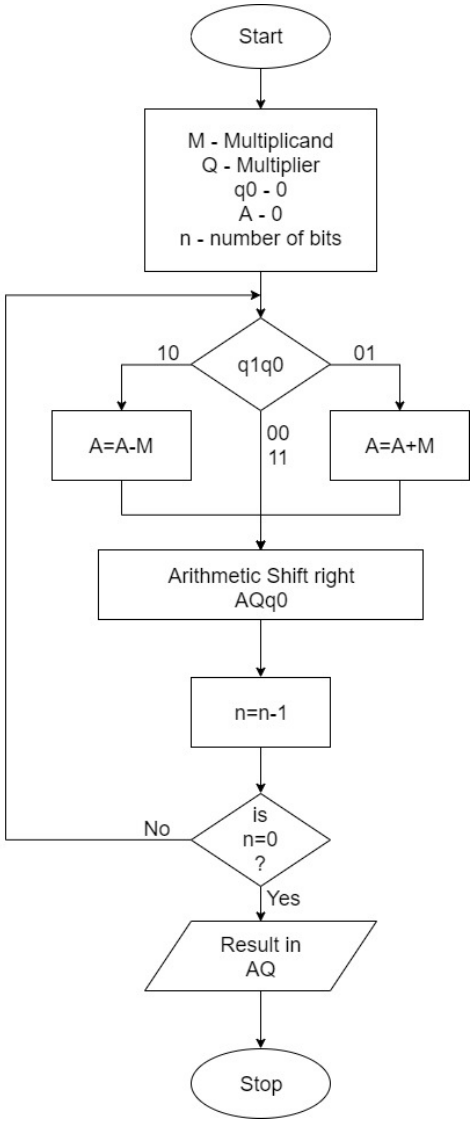


Figure 4.1 – Block Diagram of Booth Multiplier

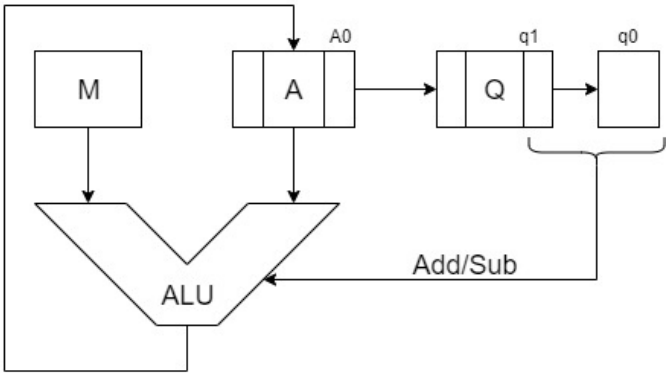


Figure 4.2 – Hardware Structure Implementation

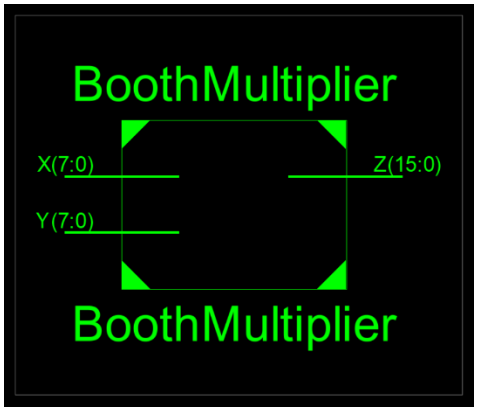


Figure 4.3 – RTL Schematic Block

Booth's algorithm examines adjacent pairs of bits of the 'N'-bit multiplier Y in signed two's complement representation, including an implicit bit below the least significant bit, $\mathbf{y_{-1}} = 0$. For each bit $\mathbf{y_i}$ for **i** running from 0 to N − 1, the bits $\mathbf{y_i}$ and $\mathbf{y_{i-1}}$ are considered.

Where these two bits are equal, the product accumulator P is left unchanged. Where $\mathbf{y_i} = 0$ and $\mathbf{y_{i-1}} = 1$, the multiplicand times $\mathbf{2^i}$ is added to P; and where $\mathbf{y_i} = 1$ and $\mathbf{y_{i-1}} = 0$, the multiplicand times $\mathbf{2^i}$ is subtracted from P. The final value of P is the signed product.

Figure 4.4 – RTL Schematic

Consider a positive multiplier consisting of a block of 1s surrounded by 0s. For example, 00111110. The product is given by:

$$M \times \text{''0 0 1 1 1 1 1 0''} = M \times (2^5 + 2^4 + 2^3 + 2^2 + 2^1) = M \times 62$$

where M is the multiplicand. The number of operations can be reduced to two by rewriting the same as

$$M \times \text{''0 1 0 0 0 0-1 0''} = M \times (2^6 - 2^1) = M \times 62.$$

In fact, it can be shown that any sequence of 1s in a binary number can be broken into the difference of two binary numbers:

$$(\ldots 0\overbrace{1\ldots 1}^{n}0\ldots)_2 \equiv (\ldots 1\overbrace{0\ldots 0}^{n}0\ldots)_2 - (\ldots 0\overbrace{0\ldots 1}^{n}0\ldots)_2.$$

Hence, the multiplication can actually be replaced by the string of ones in the original number by simpler operations, adding the multiplier, shifting the partial product thus formed by appropriate places, and then finally subtracting the multiplier. It is making use of the fact that it is not necessary to do anything but shift while dealing with 0s in a binary multiplier, and is similar to using the mathematical property that 99 = 100 − 1 while multiplying by 99.

This scheme can be extended to any number of blocks of 1s in a multiplier (including the case of a single 1 in a block). Thus,

$$M \times \text{''0 0 1 1 1 0 1 0''} = M \times (2^5 + 2^4 + 2^3 + 2^1) = M \times 58$$
$$M \times \text{''0 1 0 0-1 1-1 0''} = M \times (2^6 - 2^3 + 2^2 - 2^1) = M \times 58.$$

Booth's algorithm follows this old scheme by performing an addition when it encounters the first digit of a block of ones (0 1) and a subtraction when it encounters the end of the block (1 0). This works for a negative multiplier as well. When the ones in a multiplier are grouped into long blocks, Booth's algorithm performs fewer additions and subtractions than the normal multiplication algorithm.

CHAPTER 5

IMPLEMENTATION DETAILS

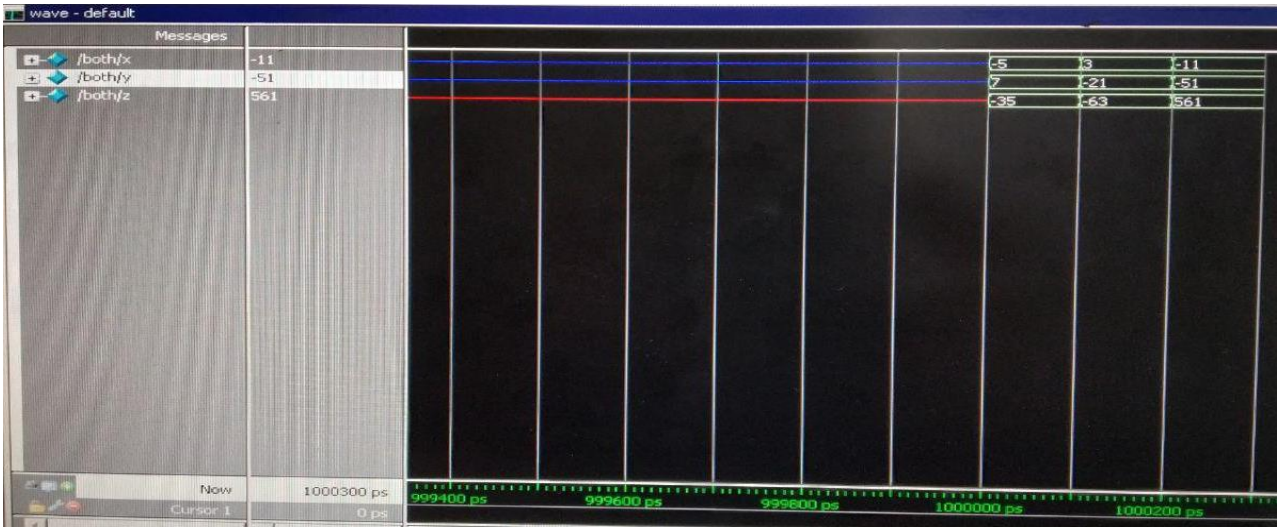
5.1 Source Code

```
module BoothMulti(X, Y, Z);

input signed [7:0] X, Y;

output signed [15:0] Z;

reg signed [15:0] Z;

reg [1:0] temp;

integer i;

reg E1;

reg [7:0] Y1;

always @ (X, Y)

begin

Z = 16'd0;

E1 = 1'd0;

for (i = 0; i < 4; i = i + 1)

begin

temp = {X[i], E1};

Y1 = - Y;

case (temp)

2'd2 : Z [15 : 7] = Z [15 : 7] + Y
```

```
2'd1 : Z [15: 7] = Z [15 : 7] + Y;

default : begin end

endcase

Z = Z >> 1;

Z[15] = Z[14];

E1 = X[i];

end

if (Y == 8'd16)

begin

Z = - Z;

end

end

endmodule
```

CHAPTER 6

RESULTS



6.1 Output 1

Let **A: -5** and **B: 7**

Multiplicand:	
Decimal:	-5
Binary:	11111011
Multiplier -	
Decimal:	7
Binary:	00000111
Two's Complement:	11111001
Steps -	
Starting Out:	0000000011111011 0
Subtract:	1111100111111011 0
Shift:	0111110011111101 1
Shift:	0011111001111110 1
Add:	0100010101111110 1
Shift:	0010001010111111 0
Subtract:	0001101110111111 0
Shift:	0000110111011111 1
Shift:	0000011011101111 1
Shift:	0000001101110111 1
Shift:	0000000110111011 1
Shift:	0000000011011101 1
Final Product (Binary):	0000000011011101
Final Product (Decimal):	-35

6.2 Output 2

Let **A: 3** and **B: -21**

Multiplicand:	
Decimal:	3
Binary:	00000011
Multiplier -	
Decimal:	-21
Binary:	11101011
Two's Complement:	00010101
Steps -	
Starting Out:	0000000000000011 0
Subtract:	0001010100000011 0
Shift:	0000101010000001 1
Shift:	0000010101000000 1
Add:	1111000001000000 1
Shift:	0111100000100000 0
Shift:	0011110000010000 0
Shift:	0001111000001000 0
Shift:	0000111100000100 0
Shift:	0000011110000010 0
Shift:	0000001111000001 0
Final Product (Binary):	0000001111000001
Final Product (Decimal):	-63

6.3 Output 3

Let **A: -11** and **B: -51**

Multiplicand:	
Decimal:	-11
Binary:	11110101
Multiplier -	
Decimal:	-51
Binary:	11001101
Two's Complement:	00110011
Steps -	
Starting Out:	0000000011110101 0
Subtract:	0011001111110101 0
Shift:	0001100111111010 1
Add:	1110011011111010 1
Shift:	0111001101111101 0
Subtract:	1010011001111101 0
Shift:	0101001100111110 1
Add:	0010000000111110 1
Shift:	0001000000011111 0
Subtract:	0100001100011111 0
Shift:	0010000110001111 1
Shift:	0001000011000111 1
Shift:	0000100001100011 1
Shift:	0000010000110001 1
Final Product (Binary):	0000010000110001
Final Product (Decimal):	561

CHAPTER 7

APPLICATIONS

Used in many ALU circuits, ALU unit of computer to calculate multiplication of signed and unsigned number in binary form, this is how computer internally calculate signed number multiplication, reduces the delay of multiplier by using Booth multiplier.

CHAPTER 8

CONCLUSION AND FUTURE ENHANCEMENT

8.1 Conclusion

The integral circuitries & communication in digital world has a cardinal role in the today's world. The IC is the main constituent of a PCB. Also, ICs have a significant work in field of electronics. In order to make an IC efficient & with mess heat, it is formulated in such a way with minimal area, delay & absorption of power. In this document, the area & delay are made minimal. Booth Multiplier handles both positive and negative multiplier uniformly. It achieves efficiency in the number of additions required when the multiplier has a few large blocks of 1's.

8.2 Future Enhancement

A modified booth multiplier should concentrate on the following things.

- On reducing the total number of partial products generated. This may include any coding methods or reduction of computation complexity of generation partial products.
- A significant amount of delay is consumed in finding two's complement of multiplicand. So this delay should be reduced.
- The optimization of adder structure. Once partial products generated, they have to be grouped and added in a systematic manner consuming less delay. This may consider the use of parallelism of the process. Next is focus is on the method included in adding the two operands; the carry propagation should be treated efficiently. Finally for the hardware implementation, suitable hardware descriptive language should be chosen and the code should be well optimized, synthesized and simulated using the optimum tool.

BIBLIOGRAPHY

[1] Booth, Andrew Donald, *‘"A Signed Binary Multiplication Technique”,* The Quarterly Journal of Mechanics and Applied Mathematics. **IV** (2): 236–240.

[2] Chen, Chi-hau, *‘*Signal processing handbook*”,* CRC Press. p. 234. ISBN 978-0-8247-7956-6

[3] Guruprasad, “Modified Booth Multiplier & it’s Applications”, October 2008

[4] https://en.wikipedia.org/wiki/Booth%27s_multiplication_algorithm

[5] https://www.codesexplorer.com/2017/02/4-bit-booth-multiplier-verilog-code.html