Project report on

BOOTH's MULTIPLIER

Submitted by:

HARDWARE HUSTLER

Group Members:

1. Rachit Gupta BT21ECE064
2. Mohammad Asif BT21ECE004
3. Sujal Agarwal BT21ECE055

A report submitted for the partial fulfilment of the requirements of the course

ECL-303 Hardware Description Languages

Submission Date: 09/11/2023

Under the guidance of:

Dr. Mayank B. Thacker

Department of Electronics and Communication Engineering

भारतीय सूचना प्रौद्योगिकी संस्थान, नागपुर

Indian Institute of Information Technology, Nagpur

ACKNOWLEDGEMENT

We express our deep sense of gratitude to our guide professor, Dr. MAYANK THACKER, for his valuable guidance and encouragement. We would also like to express our gratitude to all those who were involved directly or indirectly with the completion of this project.

Table of Contents:

1. ABSTRACT

Booth's multiplier is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation. It was invented by Andrew Donald Booth in 1950 while doing research on crystallography at Birkbeck College in Bloomsbury, London. The algorithm uses fewer additions and subtractions than more straightforward algorithms, making it more efficient. In this project we have implemented Booth's multiplication algorithm to multiply two 8-bit signed numbers.

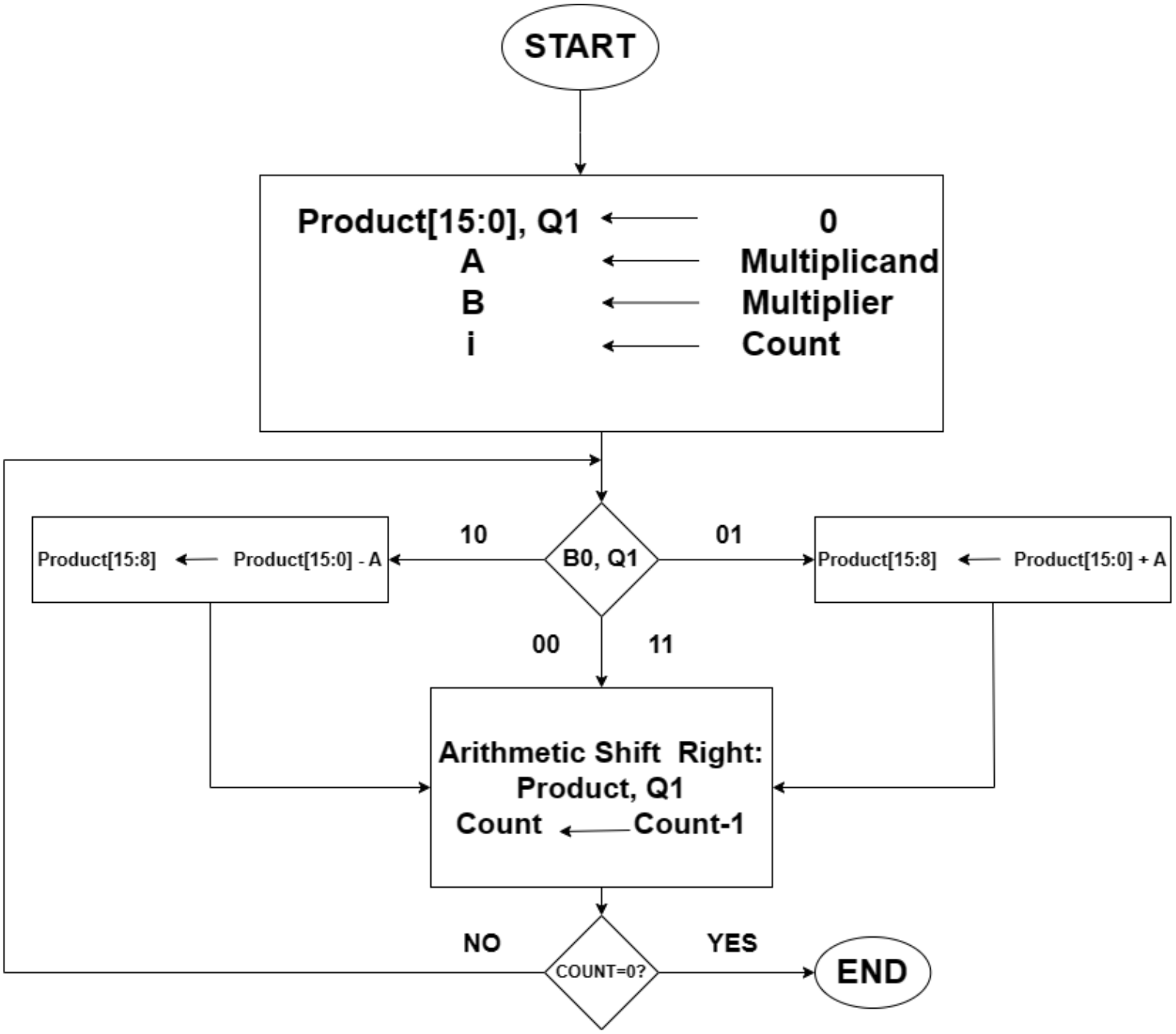
2. INTRODUCTION

Booth's multiplier is a fascinating topic to explore in the field of computer architecture. It is a multiplication algorithm that uses fewer additions and subtractions than more straightforward algorithms, making it more efficient. The project aims to implement the Booth's multiplier algorithm using Verilog HDL. The project will involve designing and simulating the Booth's multiplier using Verilog HDL. The project will also involve testing the design and verifying the results on FPGA board.

3. ALGORITHM

- The algorithm works by placing the multiplicand and multiplier in the 8-bit registers A and B, respectively. The result will be stored in a register Product[16] which will be of 16 bits.
- A 1-bit register is placed logically to the right of the least significant bit B0 of the B register. This is denoted by Q1. Product[15:8] (starting 8-bits of product) and Q1 are initially set to 0. B is stored in Product[7:0].
- Control logic checks the two bits B0 and Q1. If the two bits are the same (00 or 11), then all of the bits of Product and Q1 are shifted 1 bit to the right.
- If they are not the same and if the combination is 10, then the multiplicand is subtracted from Product[15:8]. If the combination is 01, then the multiplicand is added to Product[15:8].
- In both cases, the results are stored in Product[15:8], and after the addition or subtraction operation, Product and Q1 are right shifted. The shifting is the arithmetic right shift operation. This is to preserve the sign of the number in Product.
- The result of the multiplication will appear in the Product register.

4. FLOWCHART


START
Product[15:0], Q1 ← 0
A ← Multiplicand
B ← Multiplier
i ← Count
B0, Q1
10
01
00
11
Product[15:8] ← Product[15:0] - A
Product[15:8] ← Product[15:0] + A
Arithmetic Shift Right:
Product, Q1
Count ← Count-1
COUNT=0?
NO
YES
END

5. VERILOG CODE IMPLEMENTATION

```
//display in hex
module disp4(z, o);
input [3:0] z;
output reg [6:0] o;
always @(z)
begin
 case (z)
4'b0000 : o = 7'b0000001;
4'b0001 :  o = 7'b1001111;
4'b0010 :  o = 7'b0010010;
4'b0011 :   o = 7'b0000110;
4'b0100 :  o = 7'b1001100;
4'b0101 :   o = 7'b0100100;
4'b0110 :   o = 7'b0100000;
4'b0111 :    o = 7'b0001111;
4'b1000 :  o = 7'b0000000;
4'b1001 :   o = 7'b0000100;
4'b1010 :   o = 7'b0000010;
4'b1011 :    o = 7'b1100000;
4'b1100 :   o = 7'b0110001;
4'b1101 :    o = 7'b1000010;
4'b1110 :    o = 7'b0110000;
4'b1111 :     o = 7'b0111000;
endcase
end
endmodule
// main module
module
main(product,ip,switch1,push1,
push2,out0,out1,out2,out3,out4
,out5,LEDOUT);
output reg [15:0] product;
output [7:0]LEDOUT;
output
[6:0]out0,out1,out2,out3,out4,o
ut5;
input [7:0]ip;
input switch1,push1,push2;
reg q1;
reg [1:0]temp;
reg [7:0]nM;
integer i;
reg [7:0]A,B;

//Take input
always@(posedge push1 )
begin
if(switch1==0) A=ip;
else A=-ip;
end
```

```
always @(posedge push2)
begin
if(switch1==0) B=ip;
else B=-ip;
end

// display input
disp4 stage4(ip[3:0],out4);
disp4 stage5(ip[7:4],out5);
assign LEDOUT=ip;

//Main Booth's Algorithm
always @(A,B)
begin

//initial assignment
product[15:8]=8'd0;
product[7:0]=B;
q1=1'b0;
nM=-A;

for(i=0;i<8;i=i+1)
begin
    temp={B[i],q1};
    case(temp)
  2'b01:
product[15:8]=product[15:8]+A;
    2'b10:
product[15:8]=product[15:8]+n
M;
    endcase
    q1=B[i];
    product=product>>1;
    product[15]=product[14];
end
end

//Display the Output
disp4
stage0(product[3:0],out0);
disp4
stage1(product[7:4],out1);
disp4
stage2(product[11:8],out2);
disp4
stage3(product[15:12],out3);
endmodule
```

6. CONCLUSION

The simulation results obtained on FPGA board confirms that Booth's algorithm is an efficient way of multiplying 2 signed numbers.

7. REFRENCES

1.https://www.geeksforgeeks.org/computer-organization-booths-algorithm/

2. https://www.chipverify.com/tutorials/verilog