8 Bit Booth Multiplier

By:

Nikhil Dwivedi Ritwik Shome D Gowtham

**ABSTRACT**

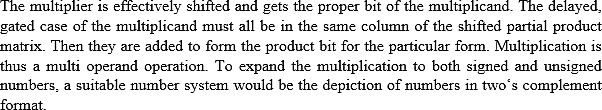
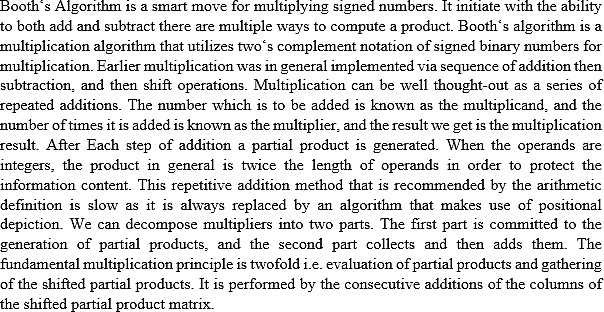
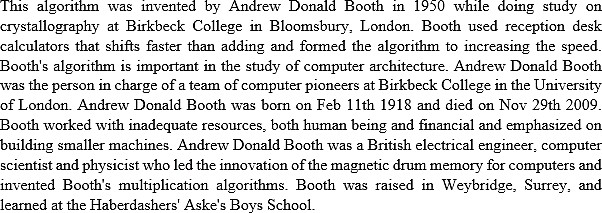
Presently, the design of a compact multiplier is playing a vital role in the stream of VLSI signal processing, DSP, Modem wireless communication etc. Multiplication in the hardware can be implemented in two ways either by using more hardware by achieving fast execution or by using less hardware and slow execution. The area of speed of the speed of the multiplier is an important issue, increment in speed results in large area consumption and vice versa. Multiplier play a very vital role in most of the high-performance system. Performance of a system depend to great extent on the performance of the multiplier thus multiplier should be fast and consume less area and hardware. This idea forced us to study and review about the booth’s algorithm, modified booth’s algorithm and its radix-2, radix-4, radix-8 forms. Simulation and the synthesis is applying the ModelSim and Xlinx 12 based on Verilog HDL, FPFA, and Spartan 6LX9 board is used for implementation.

# TABLE OF CONTENTS

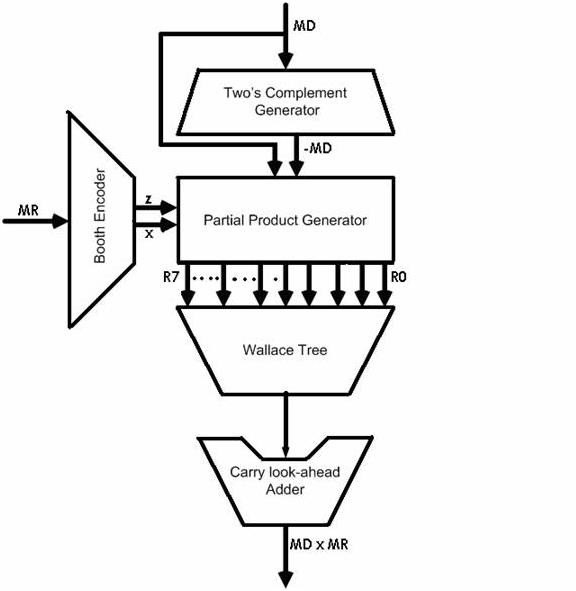
1. INTRODUCTION
2. LITERATURE SURVEY
3. BLOCK DIAGRAM
4. CIRCUIT DIAGRAM
5. PROPOSED METHODOLOGY
6. RESULTS OBTAINED
7. FUTURE SCOPE
8. REFERENCES
9. APPENDIX

**INTRODUCTION**

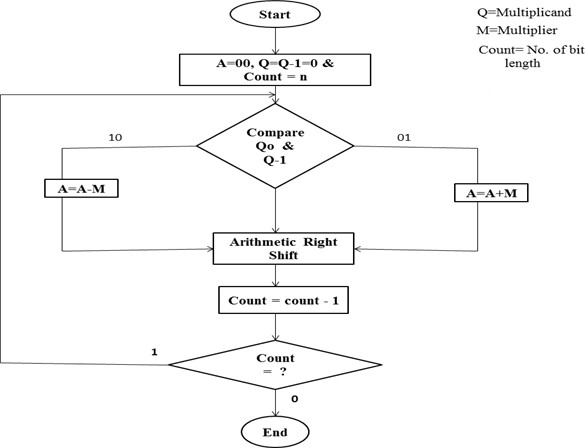
Booth’s algorithm is a multiplication algorithm that multiplies two signed binary number in two’s complement notation. Booth Algorithm is a subject of Computer Architecture and Computer Organisation.



# BLOCK DIAGRAM

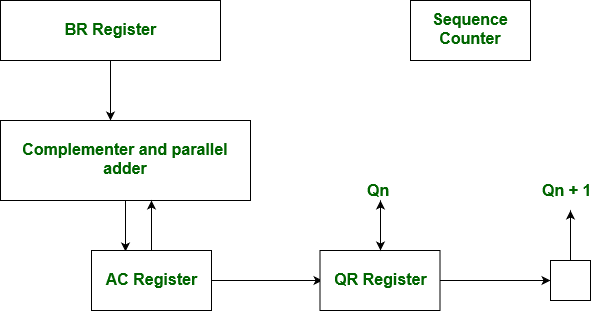


**FLOWCHART**

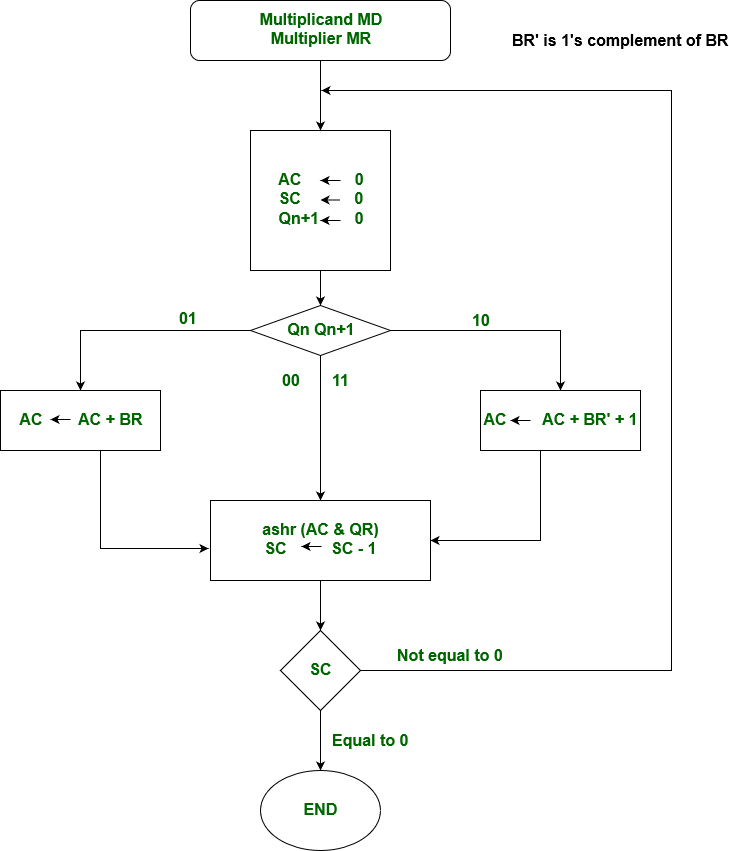


**PROPOSED METHODOLOGY**

Following is the hardware implementation of booth algorithm.



Consider the registers as A, Q, AC, BR and QR respectively. Here Qn is the LSB of multiplier of the register. We have added an extra flip flop Qn+1 to the register QR for inspection of multiplier. Following is the flow chart for the algorithm

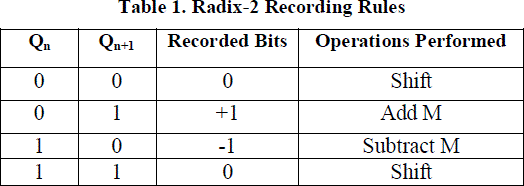


The Qn+1 bit and the AC are initially at the ground level i.e. 0. The no of bits of Q is assigned to SC. The LSB of Q i.e. Q0 and Qn+1 are compared when the two bits comes out to be 10 then we need to first perform the subtraction of M from AC i.e. AC-M operation is done.

When the 2 bits comes out to be 01 then we need to first perform the addition of M to AC i.e. AC+M is done.

When the two bits comes out to be same i.e. either 00 or 11 then no addition of subtraction operation is done to the accumulator AC.

Then the next step after comparing the two bits for all four given condition is to perform right shift operation of the accumulator AC and multiplier M along with the Qn+1. This is arithmetic shift right. The sequence counter is decremented and the computational loop is repeated n times.



**Procedure to do the multiplication manually**

1. Take M as multiplicand.
2. Take Q as multiplier.
3. Consider a 1-bit register Q-1which is initialized to 0.
4. Consider a register A which is initialised to 0.

**Conditions**

1. If Q0Q-1 are either 00 or 11 then, perform arithmetic right shift by 1 bit.
2. If Q0Q-1=10 then perform A←A-M

And then perform arithmetic right shift.

1. If Q0Q-1=01 then perform A←A+M

And then perform arithmetic right shift.

**For Example:**

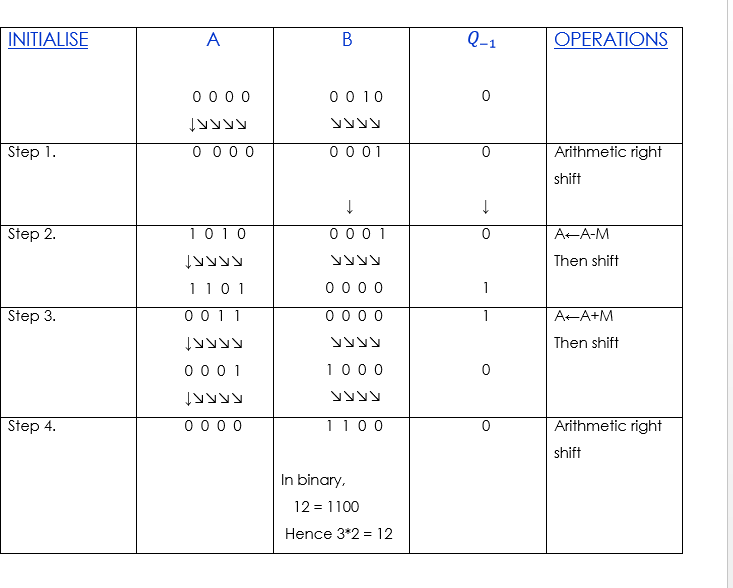
Consider the multiplication of 6 and 2.

Here taking 6 as multiplicand (M) and taking 2 as multiplier (Q). Writing 6 and 2 in Binary form.

M=6=0 1 1 0

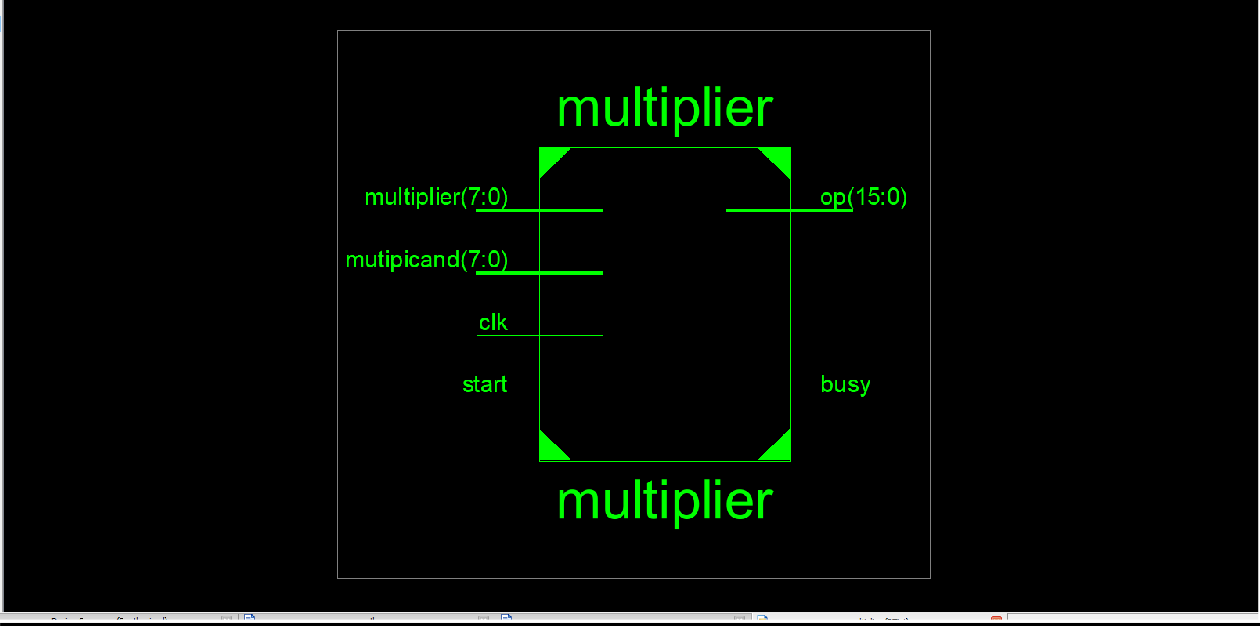
Q=2=0 0 1 0 (Q3Q2Q1Q0)

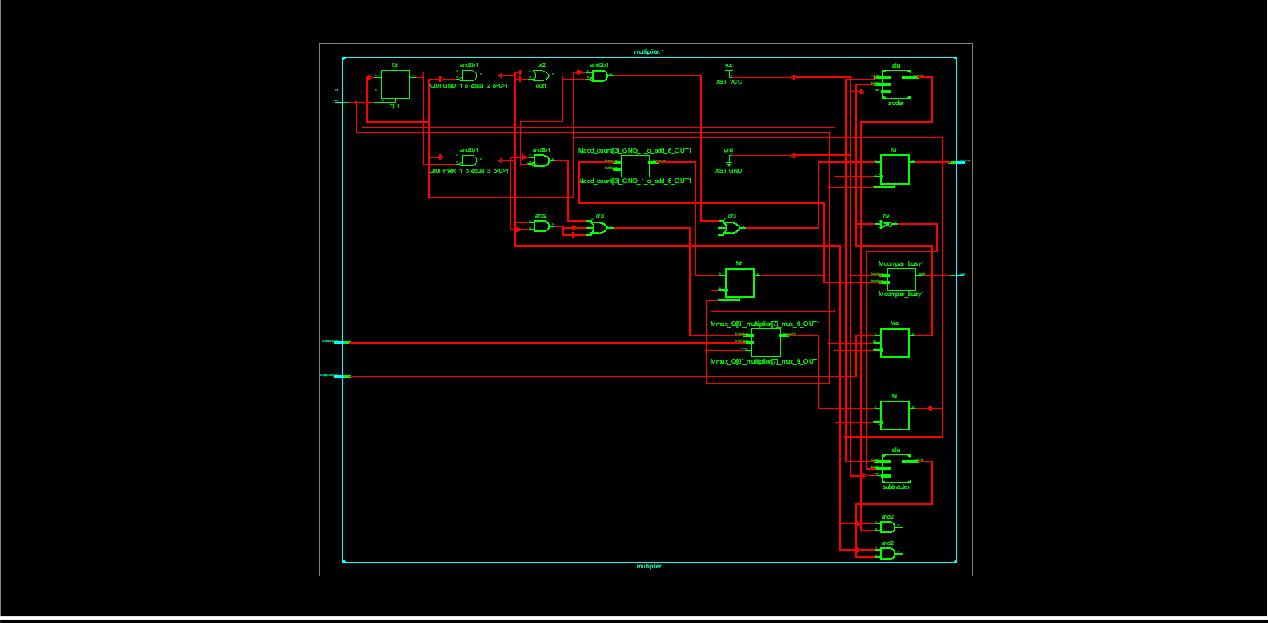
Booth’s algorithm calculates the product in n steps where n is the number of bits used to represent the numbers.



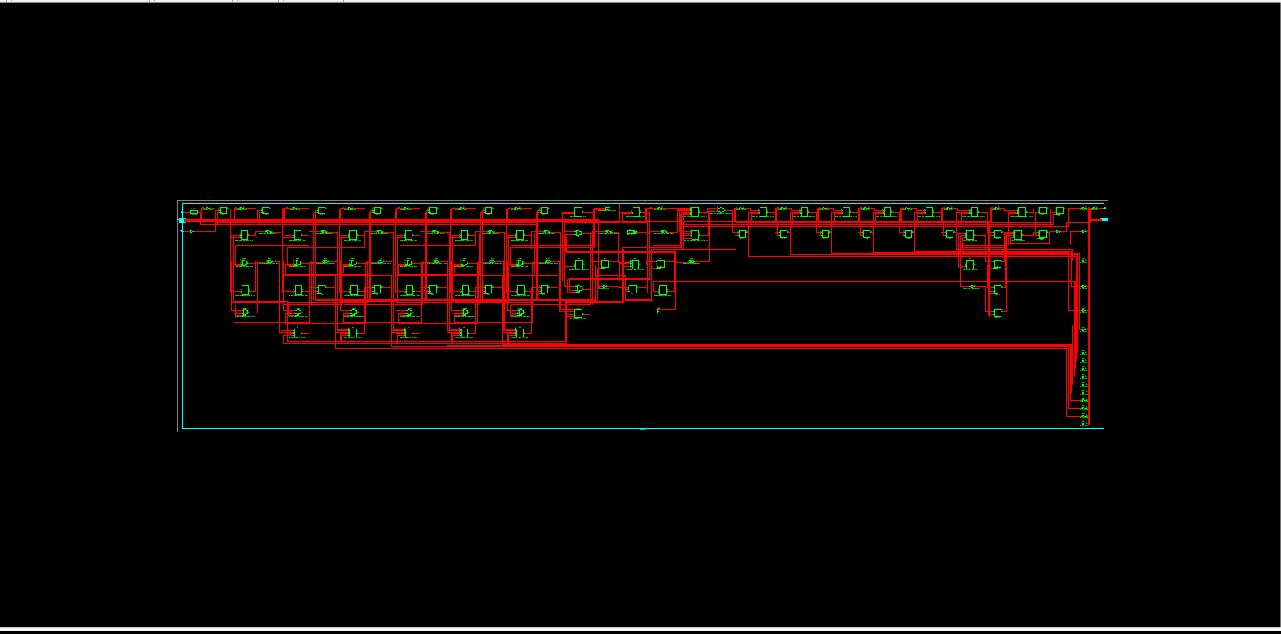
# RESULTS OBTAINED

**RTL Schematic**





**Technological Schematic**



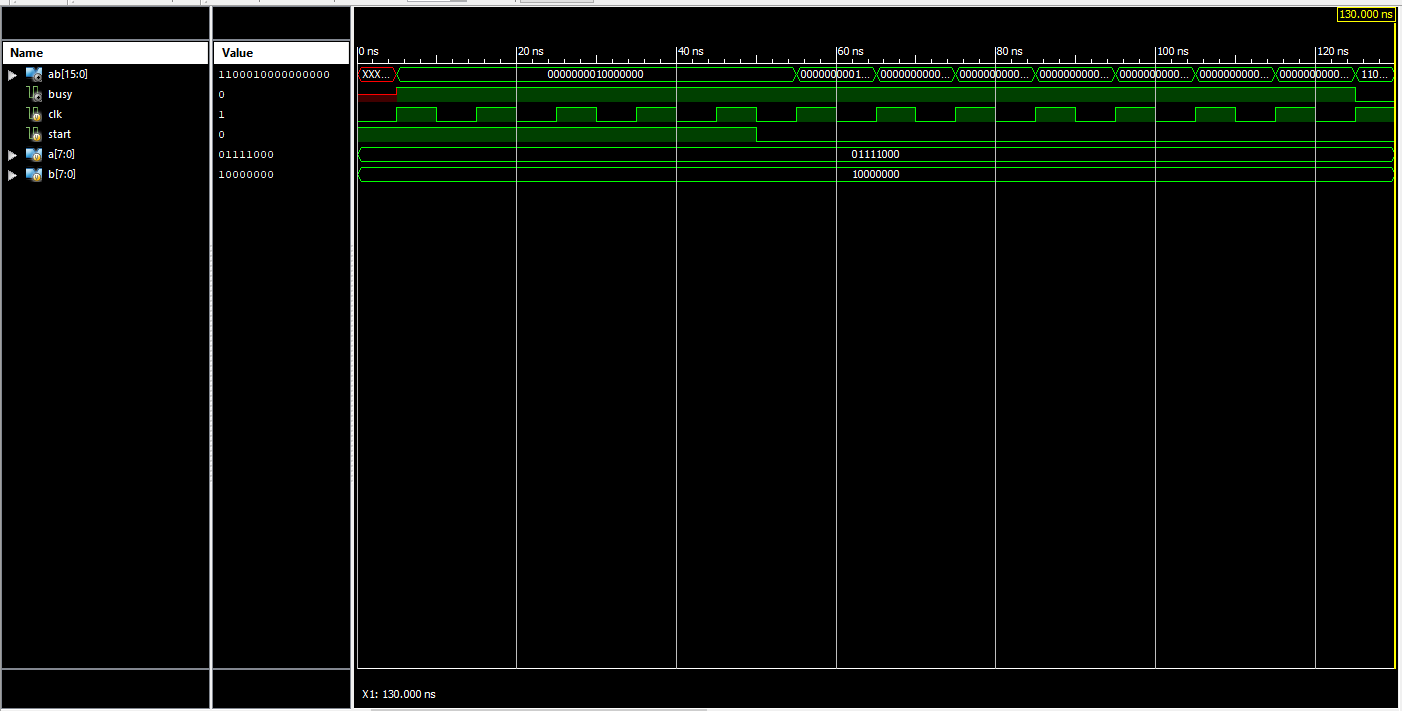
1. **When 6 is multiplied by 3**



1. **When 9 is multiplied by 9**



1. **When 120 is multiplied by 128**



**FUTURE SCOPE**

This project has a different kind of algorithm and it can never be extinct. The uniqueness of this algorithm will make everyone use it very conveniently. The algorithm is very easy to understand and apply.

The algorithm can be used for both positive and negative integer so it is used very widely as compared to other multipliers.

1. **PROGRAMMING CODE**

**APPENDIX**

module multiplier(op, busy, mutipicand, multiplier, clk, start); output [15:0] op;

output busy;

input [7:0] mutipicand, multiplier; input clk, start;

reg [7:0] A, Q, M;

reg Q\_1;

reg [3:0] count;

wire [7:0] sum, difference; always @(posedge clk) begin

if (start) begin A <= 8'b0;

M <= mutipicand;

Q <= multiplier;

Q\_1 <= 1'b0;

count <= 4'b0; end

else begin

case ({Q[0], Q\_1})

2'b0\_1 : {A, Q, Q\_1} <= {Q[0], sum, Q};

2'b1\_0 : {A, Q, Q\_1} <= {Q[0], difference, Q};

default: {A, Q, Q\_1} <= {Q[0], A, Q};

endcase

count <= count + 1'b1; end

end

alu adder (sum, A, M, 1'b0);

alu subtracter (difference, A, ~M, 1'b1); assign op= {A, Q};

assign busy = (count < 8); endmodule

//The following is an alu.

//It is an adder, but capable of subtraction:

//Recall that subtraction means adding the two's complement--

//a - b = a + (-b) = a + (inverted b + 1)

//The 1 will be coming in as cin (carry-in) module alu(out, a, b, cin);

output [7:0] out;

input [7:0] a;

input [7:0] b; input cin;

assign out = a + b + cin; endmodule

## TESTBENCH

module testbench; reg clk, start;

reg [7:0] a, b;

wire [15:0] ab; wire busy;

multiplier multiplier1(ab, busy, a, b, clk, start); initial begin

clk = 0;

$display("first example: a = 9 b = 9"); a = 9; b = 9; start = 1; #50 start = 0; #80 $display("first example done");

$finish; end

always #5 clk = !clk;

always @(posedge clk) $strobe("ab: %d busy: %d at time=%t", ab, busy,

$stime); endmodule