Implementation of 4-bit Booth's Algorithm Multiplier in Verilog

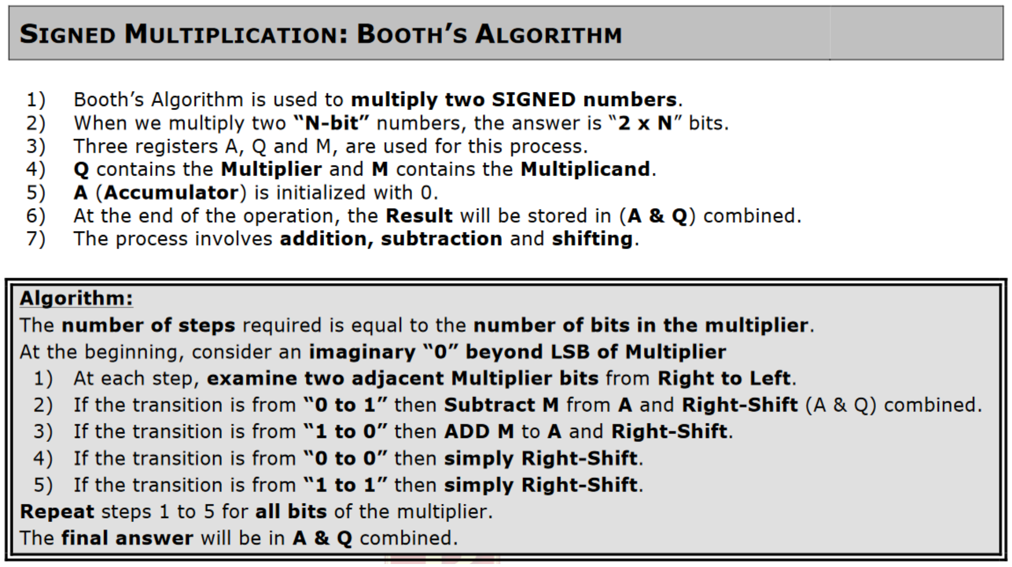
Chunduri Sai Abhishek[1]

[1] *Indian Institute of Space Science and Technology, Trivandrum*
[1] *chunduri.sc18b114@ug.iist.ac.in*

ABSTRACT

Booth's multiplication algorithm is a multiplication algorithm that multiplies two signed binary numbers in two's complement notation. 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, $y_{-1} = 0$. For each bit y_i for i running from 0 to N-1, the bits y_i and y_{i-1} are considered. Where these two bits are equal, the product accumulator P is left unchanged. Where $y_i = 0$ and $y_{i-1} = 1$, the multiplicand times 2^i is added to P; and where $y_i = 1$ and $y_{i-1} = 0$, the multiplicand times 2^i is subtracted from P. The final value of P is the signed product.

1 Algorithm



SIGNED MULTIPLICATION: BOOTH'S ALGORITHM

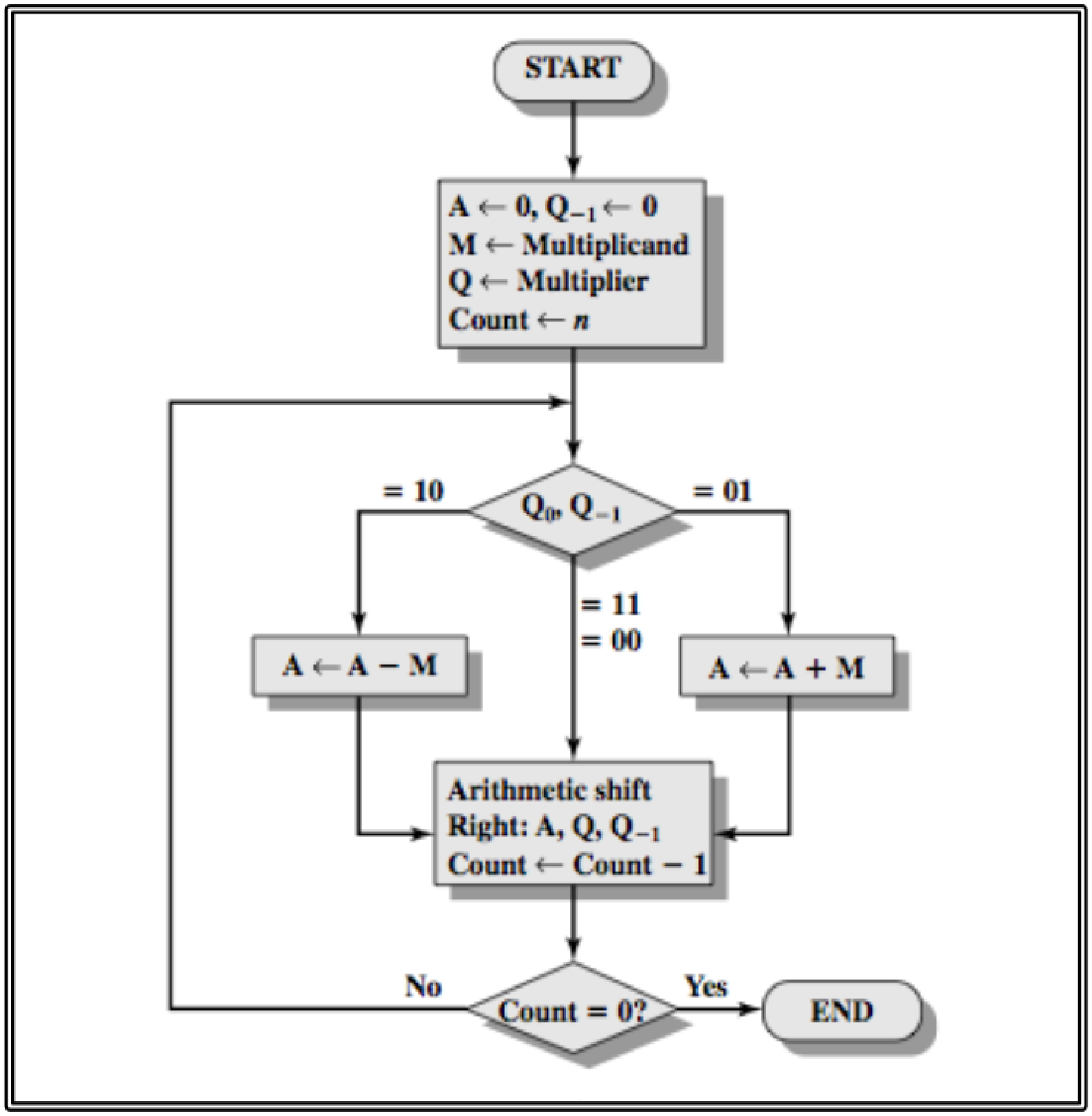
1) Booth's Algorithm is used to **multiply two SIGNED numbers**.
2) When we multiply two **"N-bit"** numbers, the answer is "**2 x N**" bits.
3) Three registers A, Q and M, are used for this process.
4) **Q** contains the **Multiplier** and **M** contains the **Multiplicand**.
5) **A** (**Accumulator**) is initialized with 0.
6) At the end of the operation, the **Result** will be stored in (**A & Q**) combined.
7) The process involves **addition, subtraction** and **shifting**.

Algorithm:
The **number of steps** required is equal to the **number of bits in the multiplier**.
At the beginning, consider an **imaginary "0" beyond LSB of Multiplier**

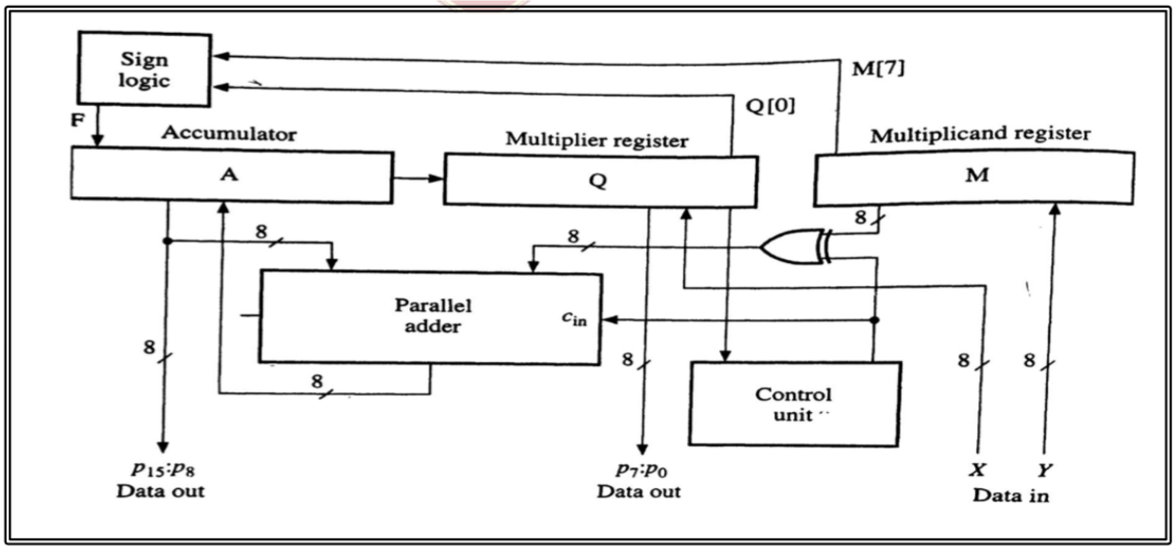
1) At each step, **examine two adjacent Multiplier bits** from **Right to Left**.
2) If the transition is from **"0 to 1"** then **Subtract M** from **A** and **Right-Shift** (A & Q) combined.
3) If the transition is from **"1 to 0"** then **ADD M** to **A** and **Right-Shift**.
4) If the transition is from **"0 to 0"** then **simply Right-Shift**.
5) If the transition is from **"1 to 1"** then **simply Right-Shift**.

Repeat steps 1 to 5 for **all bits** of the multiplier.
The **final answer** will be in **A & Q** combined.

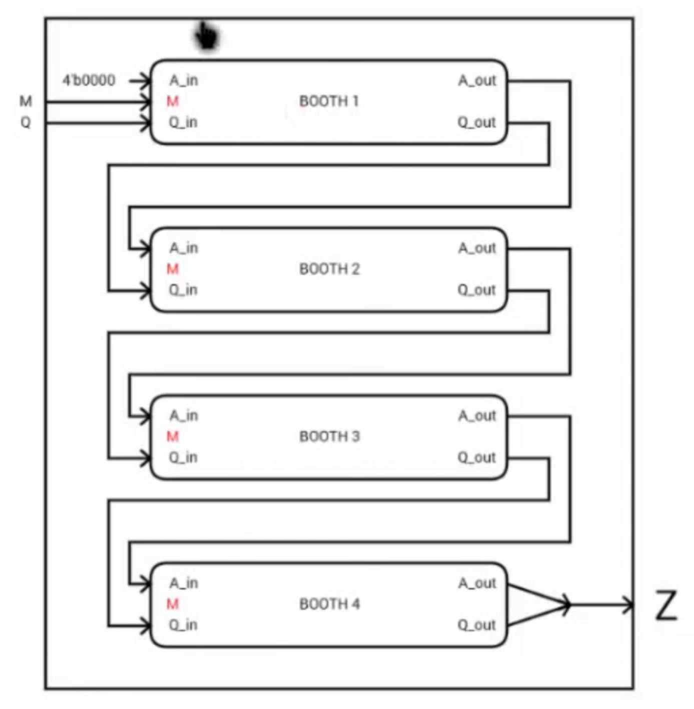
1.1 Flow Chart



2 Block Diagram



3 Module Implemented in Verilog



4 Verilog Codes

4.0.1 Booth's Multiplier Block

```
`timescale 1ns / 1ps

// This module will be checking the Q_i-1 bit and Q_i bit and take decision //

// If Q_i-1 bit to Q_i transistion is 0->1 , then A-M , followed by right shift //
// If Q_i-1 bit to Q_i transistion is 1->0 , then A+M, followed by right shift //
// If Q_i-1 bit to Q_i transistion is 0->0 or 1->1 , then do only the right shift //


module booths_multiplier_block(
    input [3:0] A_in,
    input [3:0] M,
    input [4:0] Q_in,
    output [3:0] A_out,
    output [4:0] Q_out
    );

// A ---> Accumilator , Q ---> Multiplier , M ---> Multiplicand .... //
// output is the registers A and Q taken together in respective order //

// Note that our Q_out register is of 5 bits and not 4 bits, since it has the additional Q_i-1 bit //
// the last two bits in Q register are our Q_i-1 and Q_i bits //




// Temporary Registers //

reg [3:0] A_temp;
reg [4:0] Q_temp;

wire [3:0] A_sum = A_in + M ;
wire [3:0] A_sub = A_in + ~M + 1 ;  // subtracation is same as adding 2's complement = 1's complement + 1 //



always@(A_in,M,Q_in,A_sum,A_sub) begin


  case(Q_in[1:0])  // the last two bits in Q register are our Q_i-1 and Q_i bits //

      2'b00,2'b11 : begin
```

```
                A_temp = {A_in[3],A_in[3:1]};
                Q_temp = {A_in[0],Q_in[4:1]};          //  Right Shift Algorithm //
                end

      2'b01 :      begin
                A_temp = {A_sum[3],A_sum[3:1]};
                Q_temp = {A_sum[0],Q_in[4:1]};          // A+M and Right Shift Algorithm//
                end


      2'b10 :      begin
                A_temp = {A_sub[3],A_sub[3:1]};
                Q_temp = {A_sub[0],Q_in[4:1]};          // A-M and Right Shift Algorithm//
                end

  endcase
end


assign A_out = A_temp;
assign Q_out = Q_temp;


endmodule
```

4.0.2 Booth's Multiplier Top Level Module

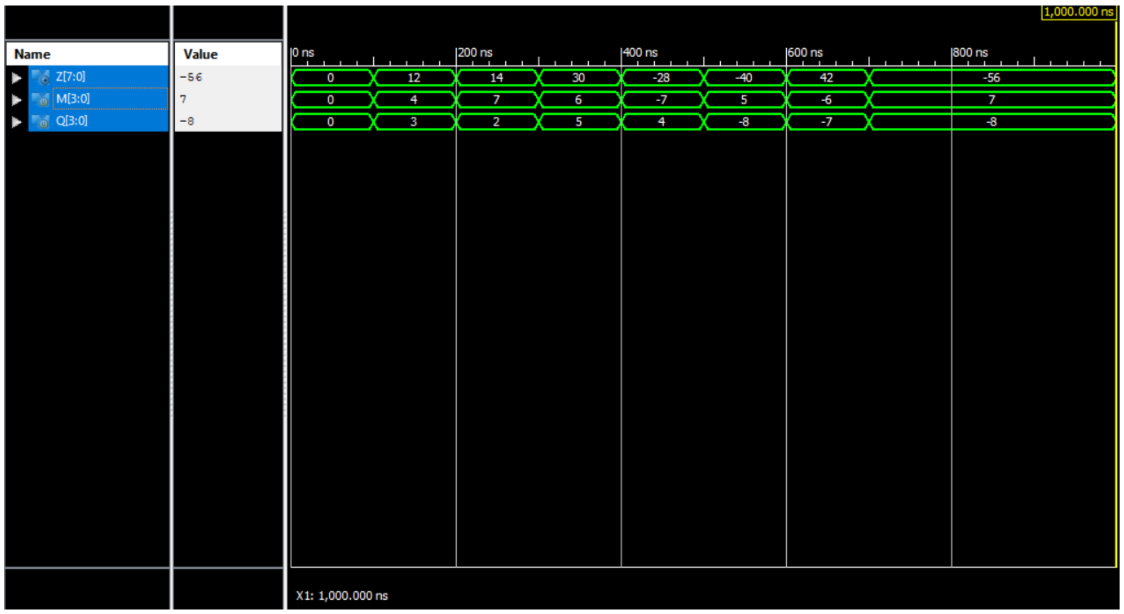
```
`timescale 1ns / 1ps

// 4-bit Booth's Multiplier//
// 3 bits is the data and the MSB is the Sign bit //
// M and Q can lie in between [-8 to +7] //

module Booths_multiplier_top_module(
    input [3:0] M,
    input [4:0] Q,
    output [7:0] Z
    );


wire [3:0] A_out1;
wire [4:0] Q_out1;

wire [3:0] A_out2;
wire [4:0] Q_out2;

wire [3:0] A_out3;
wire [4:0] Q_out3;

wire [3:0] A_out4;
wire [4:0] Q_out4;

```

```
reg [7:0] Z_temp;



//Accumilator is initially with '0000' //
// Here we make Q as 4 bit register , but in top module it's 5
    bits .... so we seperately instantiate the Q_i-1 bit as '0'
    //


booths_multiplier_block booth1 (

  .A_in(4'b0000),
  .M(M),
  .Q_in({Q,1'b0}),
  .A_out(A_out1),
  .Q_out(Q_out1)
);


booths_multiplier_block booth2 (

  .A_in(A_out1),
  .M(M),
  .Q_in(Q_out1),
  .A_out(A_out2),
  .Q_out(Q_out2)
);


booths_multiplier_block booth3 (

  .A_in(A_out2),
  .M(M),
  .Q_in(Q_out2),
  .A_out(A_out3),
  .Q_out(Q_out3)
);

booths_multiplier_block booth4 (

  .A_in(A_out3),
  .M(M),
  .Q_in(Q_out3),
  .A_out(A_out4),
  .Q_out(Q_out4)
);


assign Z = {A_out4,Q_out4[4:1]} ;


endmodule
```

4.0.3 Testbench

```
`timescale 1ns / 1ps


module testbench;

  // Inputs
  reg [3:0] M;
  reg [3:0] Q;

  // Outputs
  wire [7:0] Z;

  // Instantiate the Unit Under Test (UUT)
  Booths_multiplier_top_module uut (
    .M(M),
    .Q(Q),
    .Z(Z)
  );

  initial begin
    // Initialize Inputs
    M = 0; Q = 0;
    #100 M = 4; Q = 3;
    #100 M = 7 ; Q = 2;
    #100 M = 6; Q = 5;
       #100 M = - 7; Q = 4;
    #100 M = 5; Q = -8;
    #100 M = -6; Q = -7;
    #100 M = 7; Q = -8;



  end

endmodule
```

5 Simulation Results


1,000.000 ns
Name
Value
Z[7:0]
-56
M[3:0]
7
Q[3:0]
-8
0 ns
200 ns
400 ns
600 ns
800 ns
0
12
14
30
-28
-40
42
-56
0
4
7
6
-7
5
-6
7
0
3
2
5
4
-8
-7
-8
X1: 1,000.000 ns