Dayananda Sagar College of Engineering
Department of Electronics and Communication Engineering
Shavige Malleshwara Hills, Kumaraswamy Layout, Bengaluru – 560 078.
(An Autonomous Institute affiliated to VTU, Approved by AICTE & ISO 9001:2008 Certified)
Accredited by National Assessment and Accreditation Council (NAAC) with 'A' grade

Open Ended Experiment

Course: ABNC-II Laboratory
Course Code: 21EC481
Lab Batch: A1

Semester : 4
Date: 24/08/2023

A Report on

BOOTH MULTIPLIER

Submitted by

USN: 1DS21EC001 **NAME: A. HIMANSHU**
USN: IDS21EC006 **NAME: ABHINAV SUNDRIYAL**
USN: 1DS21EC016 **NAME: ADITYA PAWASKAR**

Faculty In-charge

Dr. Jamuna. S, Professor, ECE dept., DSCE.
Prof. Likhitha. K, Asst. Professor, ECE dept., DSCE.

Signature of Faculty In-charge

INTRODUCTION

The Booth multiplier is a hardware digital circuit used for multiplying two binary numbers in signed two's complement representation. The algorithm was invented by Andrew Donald Booth in 1950 while doing research on crystallography at Birkbeck College in Bloomsbury, London. It is a more efficient method of performing multiplication compared to the traditional method of shifting and adding, especially for larger numbers. The Booth algorithm, named after its inventor Andrew Donald Booth, reduces the number of partial products that need to be generated and added together during the multiplication process. The Booth multiplier had a profound impact on the field of computer architecture and digital arithmetic. Its introduction marked a significant advancement in the efficiency of binary multiplication, which was a critical operation in early computers for tasks ranging from scientific calculations to cryptography.

The concept of optimizing multiplication using bit patterns introduced by the Booth multiplier laid the foundation for further developments in the design of arithmetic logic units (ALUs) and floating-point units (FPUs) in processors. Over time, new algorithms and techniques have emerged to optimize multiplication even further, especially in the context of high-performance computing.

The basic idea behind the Booth algorithm is to look for consecutive groups of "1"s in the multiplier and replace them with a special encoding that simplifies the multiplication process. This reduces the number of partial products generated and the number of additions required to obtain the final result.

The main advantage of a booth multiplier over a normal array multiplier is that this can be used even for signed numbers. Also, this design converts negative numbers via 2’s complement method and stores the value and hence the multiplication is possible for signed numbers.

Algorithm:

Registers used: A, M, Q, Qres (Qres is the residual bit after a right shift of Q), n (counter)

Step 1: Load the initial values for the registers.
A = 0 (Accumulator), Qres = 0, M = Multiplicand, Q = Multiplier and n is the count value which equals the number of bits of multiplier.

Step 2: Check the value of {Q0,Qres}. If 00 or 11, go to step 5. If 01, go to step 3. If 10, go to step 4.

Step 3: Perform A = A + M. Goto step 5.

Step 4: Perform A = A - M.

Step 5: Perform Arithmetic Shift Right of {A, Q, Qres} and decrement count.

Step 6: Check if counter value n is zero. If yes, go to next step. Else, go to step 2.

Step 7: Stop

Explanation: -

A Booth multiplier is a digital circuit used to perform binary multiplication efficiently, particularly for signed numbers, by using a combination of shifting and addition operations. The key idea behind the Booth multiplier is to reduce the number of partial products that need to be generated and added together.

The multiplier takes two binary numbers, multiplicand and multiplier, as inputs. It operates iteratively through a series of steps, comparing adjacent pairs of multiplier bits. These pairs of bits are interpreted as a 2-bit "Booth code," which dictates the type of operation to be performed in that step. The Booth code can have three possible values: 00, 01, or 10.

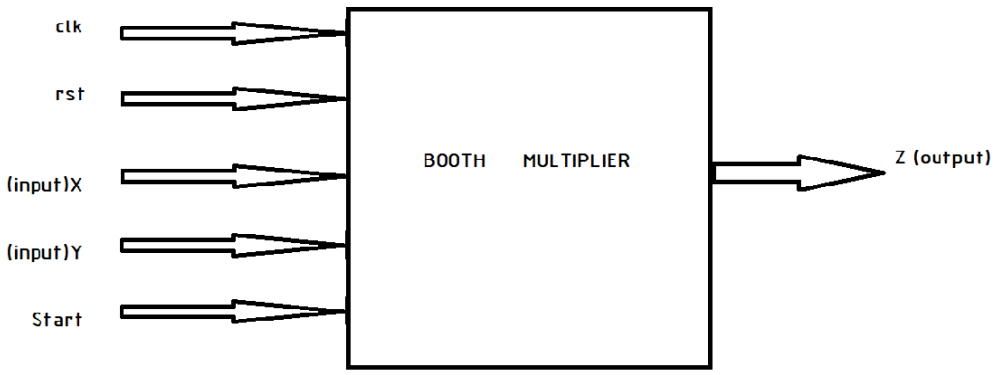
1. If the Booth code is 00, no operation is needed in this step, and both the multiplicand and partial product are shifted right by one position.
2. If the Booth code is 01, a subtraction operation takes place, where the multiplicand is subtracted from the partial product. The result is then shifted right by one position.
3. If the Booth code is 10, an addition operation occurs, where the multiplicand is added to the partial product. Again, the result is shifted right by one position.

By employing this approach, the Booth multiplier reduces the number of additions required compared to traditional methods that generate all partial products independently. This optimization is especially advantageous for numbers with long sequences of 0s, as it effectively "skips" the addition steps for these sections.

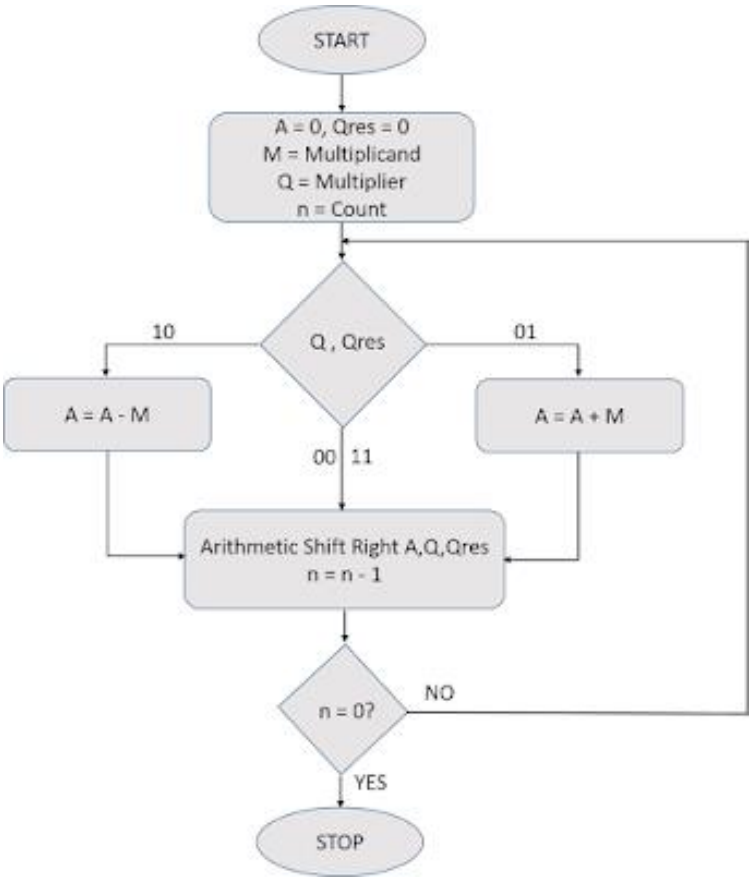
In summary, a Booth multiplier works by iteratively applying a combination of shifts and additions/subtractions based on the Booth codes of adjacent multiplier bits. This method significantly reduces the number of required operations, leading to faster binary multiplication, particularly suitable for hardware implementations in processors and digital systems.

BLOCK DIAGRAM AND FLOW CHART

Block Diagram:



Flow chart:



VERILOG CODE:

```
module BoothMul(clk,rst,start,X,Y,valid,Z);
input clk;
input rst;
input start;
input signed [3:0]X,Y;
output signed [7:0]Z;
output valid;
reg signed [7:0] Z,next_Z,Z_temp;
reg next_state, pres_state;
reg [1:0] temp,next_temp;
reg [1:0] count,next_count;
reg valid, next_valid;
parameter IDLE = 1'b0;
parameter START = 1'b1;
always @ (posedge clk or negedge rst)
begin
if(!rst)
begin
  Z          <= 8'd0;
  valid      <= 1'b0;
  pres_state <= 1'b0;
  temp       <= 2'd0;
  count      <= 2'd0;
end
else
begin
  Z          <= next_Z;
  valid      <= next_valid;
  pres_state <= next_state;
```

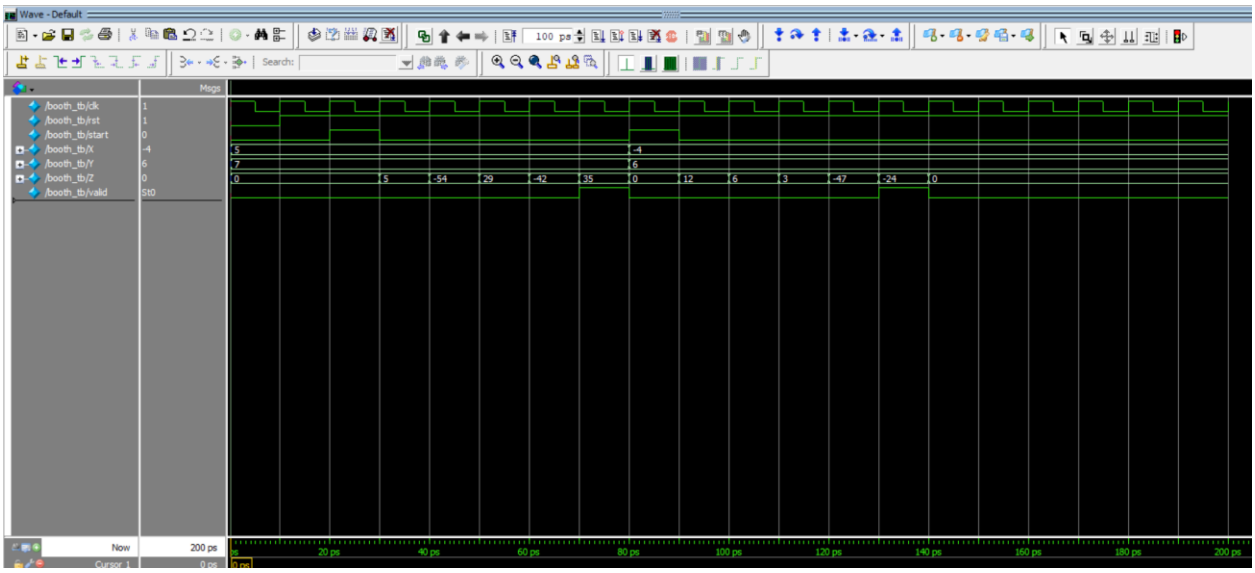
```
  temp      <= next_temp;
  count     <= next_count;
end
end
always @ (*)
begin
case(pres_state)
IDLE:
begin
next_count = 2'b0;
next_valid = 1'b0;
if(start)
begin
   next_state = START;
   next_temp  = {X[0],1'b0};
   next_Z     = {4'd0,X};
end
else
begin
   next_state = pres_state;
   next_temp  = 2'd0;
   next_Z     = 8'd0;
end
end
START:
begin
   case(temp)
   2'b10:   Z_temp = {Z[7:4]-Y,Z[3:0]};
   2'b01:   Z_temp = {Z[7:4]+Y,Z[3:0]};
   default: Z_temp = {Z[7:4],Z[3:0]};
endcase
```

```
next_temp  = {X[count+1],X[count]};
next_count = count + 1'b1;
next_Z     = Z_temp >>> 1;
next_valid = (&count) ? 1'b1 : 1'b0;
next_state = (&count) ? IDLE : pres_state;
end
endcase
end
endmodule
```

TEST BENCH CODE:

```
module booth_tb;
reg clk,rst,start;
reg signed [3:0]X,Y;
wire signed [7:0]Z;
wire valid;
always #5 clk = ~clk;
BoothMul inst (clk,rst,start,X,Y,valid,Z);
initial
$monitor($time,"X=%d, Y=%d, valid=%d, Z=%d ",X,Y,valid,Z);
initial
begin
X=5;Y=7;clk=1'b1;rst=1'b0;start=1'b0;
#10 rst = 1'b1;
#10 start = 1'b1;
#10 start = 1'b0;
@valid
#10 X=-4;Y=6;start = 1'b1;
#10 start = 1'b0;
end
endmodule
```

SIMULATION RESULTS:



RESULT:-

The verilog code for booth multiplier was simulated and waveforms were observed in modelsim.