PROJECT REPORT

8-BIT Booth Multiplier Using Verilog HDL

Submitted by:

1. Bhavya Lakshmi OS (12205644)
2. Sarath Kumar Suda (12312393)
3. Apoorv Mittal(12206778)
4. Aakula Upendra (12223224)

1.ABSTRACT: -

The Booth multiplier is a significant algorithm in binary multiplication, especially for signed numbers. This paper explores the design, implementation, and simulation of a Booth multiplier in Verilog. It highlights key features of the Booth algorithm, which reduces the number of partial products and enhances the performance of digital systems. The report outlines the components involved, provides detailed signal descriptions, and presents the RTL schematic, flowchart, and Verilog code for the Booth multiplier. The performance of the multiplier is evaluated through waveform diagrams, which demonstrate the internal workings during multiplication.

2.INTRODUCTION: -

Binary multiplication is a fundamental operation in digital systems and processors. The Booth multiplier, introduced by Andrew Donald Booth in 1951, is an efficient algorithm for multiplying signed and unsigned binary numbers. Unlike conventional multiplication methods, which operate on each bit individually, Booth's algorithm uses recoding to group bits, reducing the number of operations required.

This report investigates the Booth multiplier's implementation, focusing on the design principles, key components, and Verilog code. It also discusses the efficiency of the algorithm and its application in hardware design, particularly in reducing area and power consumption. By employing this method, digital circuits can achieve faster and more efficient multiplication, making it ideal for use in arithmetic units of processors and digital signal processors (DSPs).

3.SYSTEM OVERVIEW:-

Booth's Algorithm Flowchart –

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, goto step 5. If 01, goto step 3. If 10, goto 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, goto next step. Else, goto step 2.
Step 7: Stop

FlowChart:-

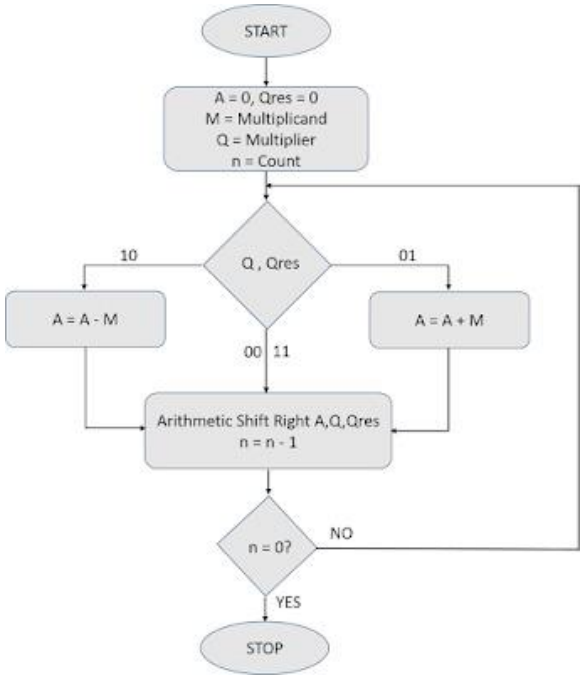

Fig: -1

4.Signal Description: -

Signal Name	Description
`clk`	Clock signal that synchronizes the operations of the Booth multiplier.
`rst`	Reset signal to initialize registers and state machine.
`start`	Control signal to start the multiplication process.
`X`	Signed 4-bit multiplicand (input).
`Y`	Signed 4-bit multiplier (input).
`Z`	Signed 8-bit product register that stores the result of the multiplication.
`valid`	Output signal indicating that the multiplication is complete and the result is valid.
`next_Z`	Temporary register storing the next value of the product (`Z`).
`Z_temp`	Intermediate register used during the calculation of the product.
`temp`	2-bit register storing recoded bits of the multiplier.
`next_temp`	Next value for the `temp` register, based on the current pair of bits in the multiplier.
`count`	2-bit counter tracking the number of iterations for Booth's algorithm.
`next_count`	Next value for the `count` register.
`pres_state`	Current state of the FSM (`IDLE` or `START`).

Fig: -2

Booth Multiplier Design

The Booth multiplier operates in two primary stages: initialization and calculation. The multiplier begins in the IDLE state, waiting for the start signal to initiate the multiplication. Once started, the design transitions to the START state, where the Booth algorithm is executed.

- **State Machine**: The FSM controls the flow of the multiplication process. It alternates between IDLE and START states based on the start signal and the progress of the calculation.
- **Recoding Logic**: In each cycle, the recoding logic looks at pairs of bits from the multiplier (Y) and performs the appropriate action:
    - **Add** the multiplicand (X) to the product if the pair is 01.
    - **Subtract** the multiplicand (X) from the product if the pair is 10.
    - **Shift** the product if the pair is 00 or 11.
- **Shifting**: After each operation (addition or subtraction), the product is shifted right by one bit. The shifting ensures that the partial products are aligned correctly.

5.RTL Code: -

```
module boothmultiplier(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
```

Fig: -3

6.Testbench: -

```
module boothmultipliertb;

reg clk,rst,start;
reg signed [3:0]X,Y;
wire signed [7:0]Z;
wire valid;

always #5 clk = ~clk;

boothmultiplier dut (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
```

Fig: -4

7.Simulation Waveform: -

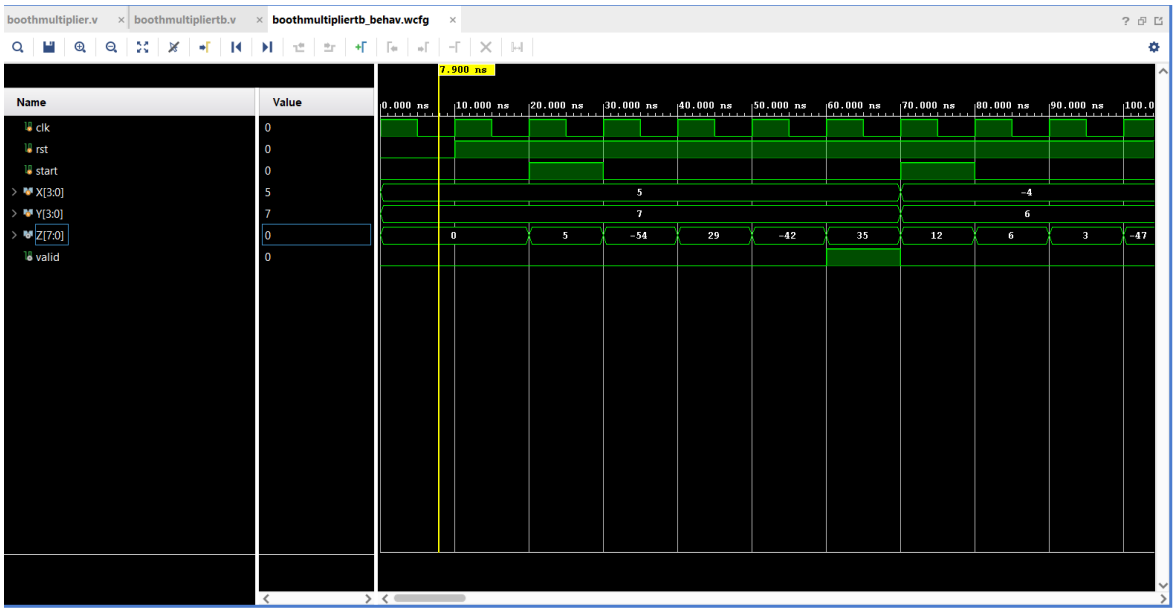

Fig: -5

8.RTL SCHEMATIC

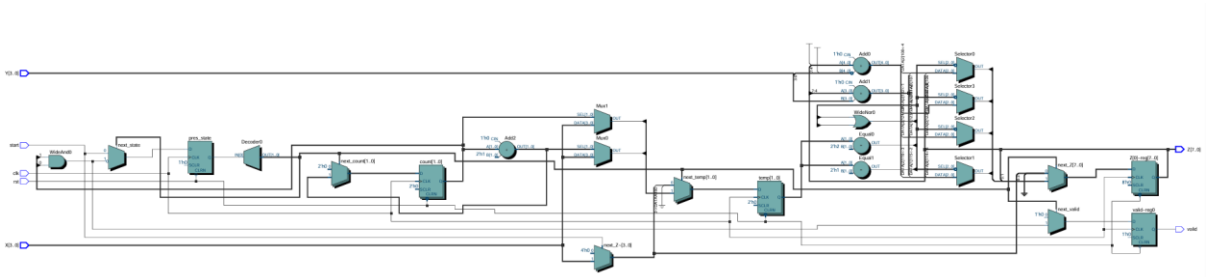

Fig: -6

9. CONCLUSION: -

The **Booth multiplier** is an efficient and effective method for signed binary multiplication. By reducing the number of partial products through recoding, it minimizes hardware complexity and improves multiplication speed. The Verilog implementation of a 4-bit Booth multiplier, using a finite state machine and recoding logic, successfully demonstrates the algorithm's benefits. With the flexibility to scale to larger bit-widths, the Booth multiplier is an essential tool for high-performance digital systems.

10.REFERENCES: -

1. Y.N. Ching, —Low-power high-speed multipliers‖, IEEE Transactions on Computers, vol. 54, no. 3, pp 355-361, 2005.

2. M. Sheplie, —High performance array multiplier‖, IEEE transactions on very large scale integration systems, vol. 12, no. 3, pp. 320-325, (2004).

3. Deepak Patidar, — Efficient Architecture for Radix-2 booth multiplication using 4:2 compressor ‖ pp. 3529-3538, (2015)

4. A High-Speed Multiplication Algorithm Using Modified Partial Product Reduction Tree P. Asadee, International Journal of Electrical and Electronics Engineering, 4: 4, (2010).

5. J. Hensley, A. Lastra, and M. Singh. An area- and energy-efficient asynchronous booth multiplier for mobile devices. In Proc. Int. Conf. Computer Design (ICCD), (2004).

6. P. D. Chidgupkar and M. T. Karad, —The Implementation of Vedic Algorithms in Digital Signal Processing‖, Global J. of Engg. Edu, vol. 8, no. 2, pp. 153–158, (2004).

7. D. Harris, R. Krishnamurthy, M. Anders, S. Mathew, and S. Hsu, —An improved unified scalable radix-2 Montgomery multiplier,‖ Proc. 17th IEEE Symp. Computer Arithmetic, pp. 172-178, 2005.