**Booth Multiplier Verilog Code**

Booth's Multiplication Algorithm is a commonly used algorithm for multiplication of two signed numbers & how to write a Verilog code for this algorithm in an FSM format.

### 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:

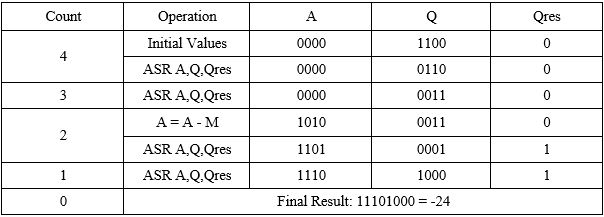
### https://1.bp.blogspot.com/-zK2KvPLJnb4/X5OkW8L8z7I/AAAAAAAAB_w/YW6nW4RXtS4oMKI9A3FFFAWgrMN0vMCbQCLcBGAsYHQ/s1428/Booth_flow.png

### Example Operation:

Consider the multiplication of  -4 and 6.

Place -4 (1100) in Q and 6 (0110) in M.

ASR stands for Arithmetic Shift Right.



### Verilog Code Logic:

Implement the Booth Multiplication algorithm for 3-bit operands (1 sign bit) in an FSM format with completely synthesizable Verilog Code.

**Inputs:**

clk, rst, X, Y (2 operands to be multiplied), start (This pulse indicates start of computation)

**Outputs:**

Z (Product) and valid (This pulse indicates the availability of final product)

1. In IDLE state, we wait for the start pulse. Upon its arrival, move to START state.
2. In START state, we follow the same algorithm and perform the computations using verilog logic as long as counter < 3. (Incrementing counter is used)
3. Upon counter completion, we send out valid pulse and goto IDLE state.

### 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**

### Testbench:

**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 Result:

### https://1.bp.blogspot.com/-jKx3pnoCpns/X5Oq9W0Fs7I/AAAAAAAACAM/2fp3ZeOBNSQ1AVmEdk6N7IyMS13aDE4WACLcBGAsYHQ/s973/booth.JPG

### Conclusions:

The Booth Multiplier Algorithm implemented in this format can be implemented on FPGA devices.

Number of clock cycles taken to produce the output depends on the counter value.

The number of bits of the incoming operands and output product can also be increased to perform mulitplication with larger signed numbers.