VLSI System Design

ELE301P

LAB - 3 - Report

Praveen B R

COE19B007

Submission Date: 09/09/2021

INDEX

4x4 Booth Multiplier

- ➢ Main Code
- ➢ Test Bench
- ➢ Output
- ➢ Waveform

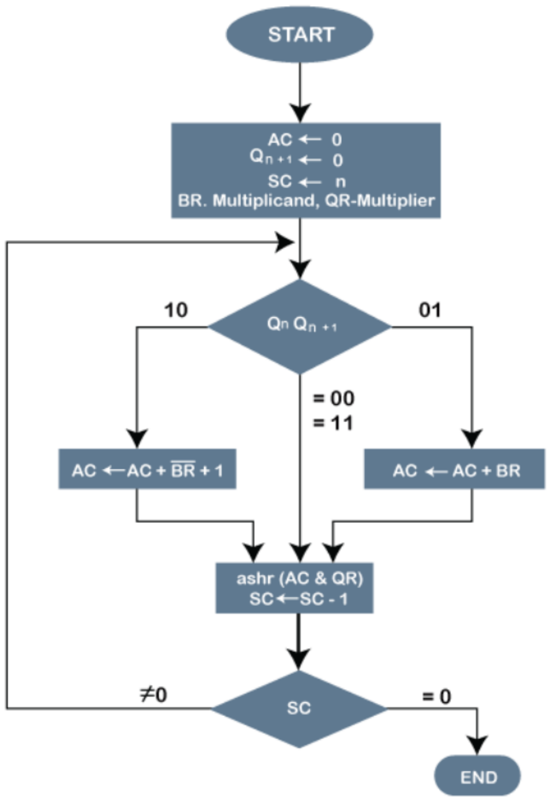
Other Questions

- ➢ Question 1
- ➢ Question 2
- ➢ Question 3

Booth Multiplier

Objective:
To implement a 4x4 Booth Multiplier using verilog in verilog code

Theory:



Booth's multiplication is meant for multiplying 2’s complement representation of signed binary numbers. Booth's multiplication is meant for multiplying 2’s complement representation of signed binary numbers. Booth algorithm converts the multiplier Y in 2’s complement form and implicitly appends a bit Y1=0 below the least significant bit. After every multiplication, the partial product thus generated is shifted according to its bit order and then all the partial products are added to obtain the final product.

Input: Two 4-bit numbers
Output: One 8-bit number

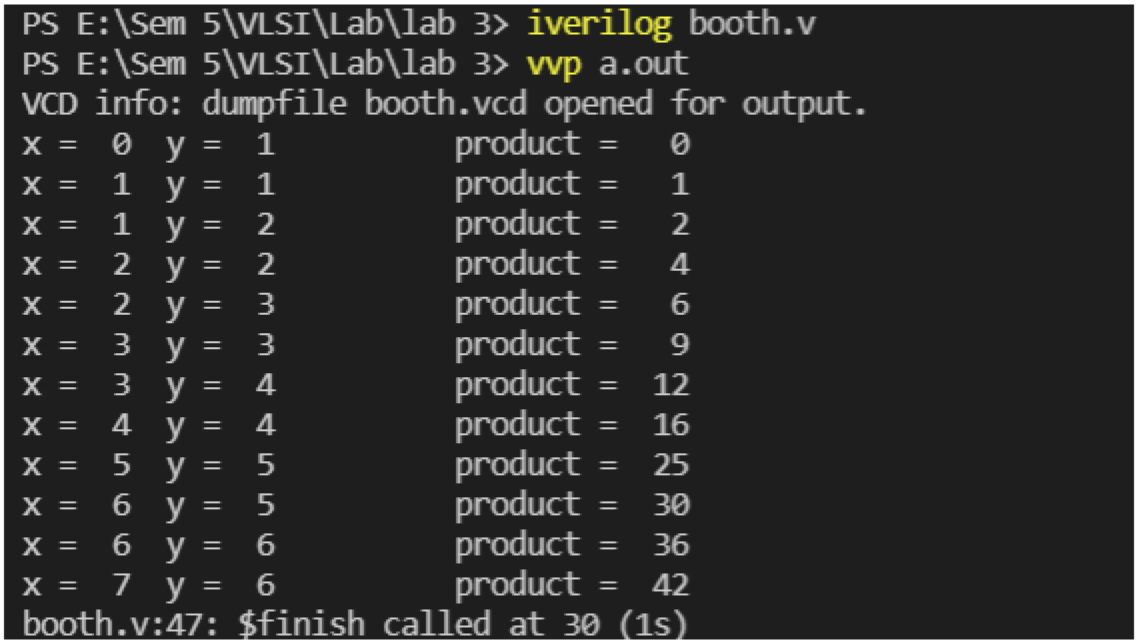
Main Code:

```
//4x4 BOOTH MULTIPLICATION
module booth_multiplication(a, b, product);  //a - Multiplicand b -
Multiplier
    input signed[3:0] a, b;
    output reg signed[7:0] product;
    reg[1:0] check;
    reg Q_1;
    integer i;

    always @(a,b)
        begin
            //Initialize valued for Prod & Q1
            product = 8'd0;
            Q_1 = 1'b0;
            product[3:0] = a;
            for(i = 0; i < 4; i = i + 1)
            begin
                //checking the concat of q[0] & q[-1] in theory
                check = {a[i], Q_1};
                if(check == 2'b10)
                    product[7:4] = product[7:4] - b;
                else if(check == 2'b01)
                    product[7:4] = product[7:4] + b;
                //arithmetic right shift
                product = product >>> 1;
                Q_1 = a[i];
            end
        end
endmodule
```

Test Bench:

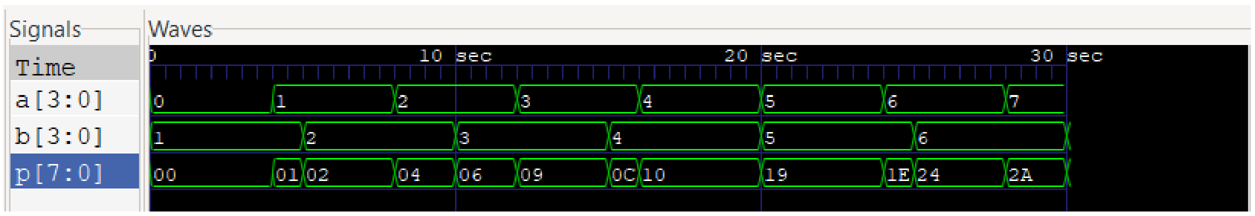
```
module booth_tb;
  // Inputs
  reg  [3:0] a;
  reg  [3:0] b;

  // Outputs
  wire [7:0] product;

  // Instantiate the Unit Under Test (UUT)
    booth_mul tb (.a(a),.b(b),.product(product));

  initial begin
    // Initialize Inputs
    #0 a=4'b0000;b=4'b0001;
    #30 $finish;
  end

  initial begin
    $dumpfile("booth.vcd");
    $dumpvars(0, booth_tb);
    $monitor("x = %d \ty = %d\t\tproduct = %d ", a, b, product);
  end

  always begin
    #4 a+=1;
  end
  always begin
    #5 b+=1;
  end

endmodule
```

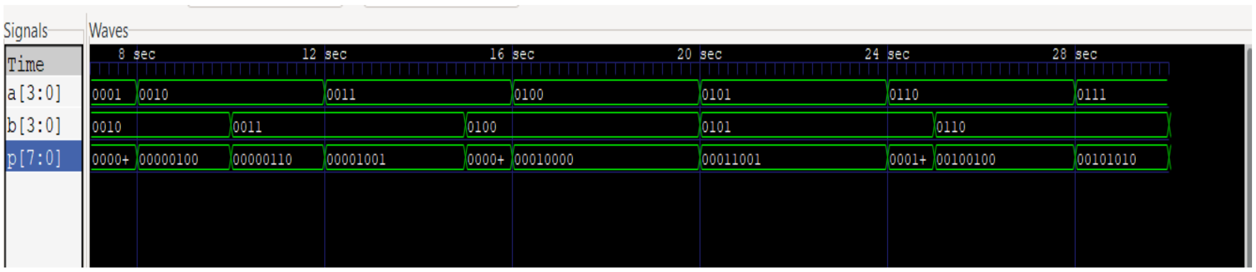
Output Terminal:



```
PS E:\Sem 5\VLSI\Lab\lab 3> iverilog booth.v
PS E:\Sem 5\VLSI\Lab\lab 3> vvp a.out
VCD info: dumpfile booth.vcd opened for output.
x =  0  y =  1         product =   0
x =  1  y =  1         product =   1
x =  1  y =  2         product =   2
x =  2  y =  2         product =   4
x =  2  y =  3         product =   6
x =  3  y =  3         product =   9
x =  3  y =  4         product =  12
x =  4  y =  4         product =  16
x =  5  y =  5         product =  25
x =  6  y =  5         product =  30
x =  6  y =  6         product =  36
x =  7  y =  6         product =  42
booth.v:47: $finish called at 30 (1s)
```

Wave Form:





Conclusion:
Thus, 4x4 booth multipliers have been implemented successfully using Verilog and simulation waveforms of more than 10 cases have been shown.

Other Questions

Q1)Loops are something handy in complex coding. Explore the loops (while, for, repeat, forever) in Verilog and their syntax. And write a small code for each loop and show the simulation outputs.

For loop: It is used to iterate a set of statements given within a loop as long as the given condition is true. An iterator is used upon which the condition depends.

While loop: Very similar to for loop, but there is no iterator here and the loop runs as long as the given condition is true.

Repeat loop: It runs a loop for a particular number of times without checking any condition.

Forever loop: It keeps running a loop as many times as possible unless we press ctrl+c.

For loop:

```
module forloop();
    integer i;
    initial begin
        $display("For loop:-");
        for(i=1;i<6;i=i+1) begin
            $display("%d",i);
        end
    end
endmodule
```

While loop:

```
module whileloop();
    integer j=1;
    initial begin
        $display("While loop:-");
        while(j<=5) begin
            $display("%d",j);
            j=j+1;
        end
    end
endmodule
```

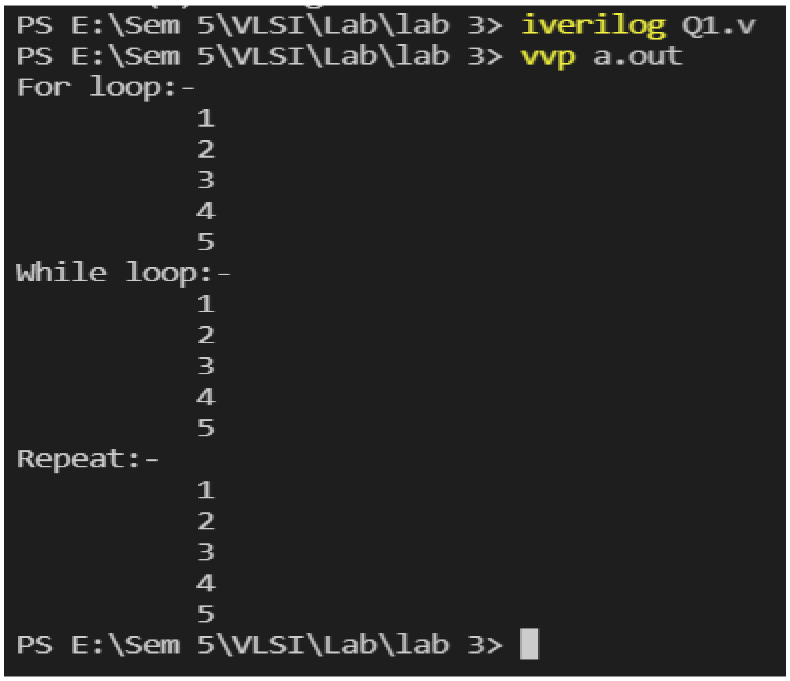
Repeat loop:

```
module repeatloop();
    integer i=1;

    initial begin
        $display("Repeat:-");
        repeat(5)
            #1 $display("%d",i++);
    end
endmodule
```

Forever loop:

```
module foreverloop ();
  initial
    begin
      integer i=1;
      forever
        #10 $display("%d",i++);
    end
endmodule
```

Output:

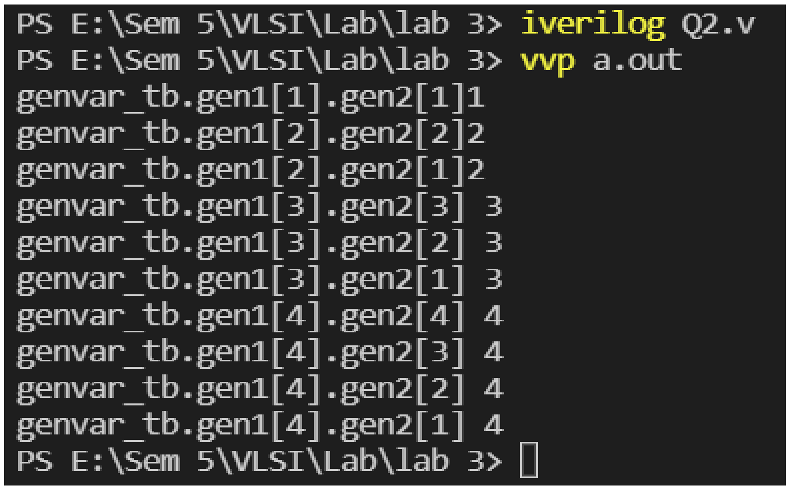


```
PS E:\Sem 5\VLSI\Lab\lab 3> iverilog Q1.v
PS E:\Sem 5\VLSI\Lab\lab 3> vvp a.out
For loop:-
          1
          2
          3
          4
          5
While loop:-
          1
          2
          3
          4
          5
Repeat:-
          1
          2
          3
          4
          5
PS E:\Sem 5\VLSI\Lab\lab 3>
```

Q2)Find the functionality of the keyword “genvar”? Write a small code using genvar.

A Genvar is a variable used in **a generate-for loop**. It stores positive integer values. It differs from other Verilog variables in that it can be assigned values and changed during compilation and elaboration time.

```
module genvar_tb();
generate
  genvar i;
  for (i = 0; i < 5; i = i + 1)
    begin : gen1
      genvar j;
      for (j = i; j >= 1; j = j - 1)
        begin : gen2
          reg [0:i] R;
          initial
            begin
              R = i;
              $display("%m", R);
            end
        end
    end
endgenerate
endmodule
```

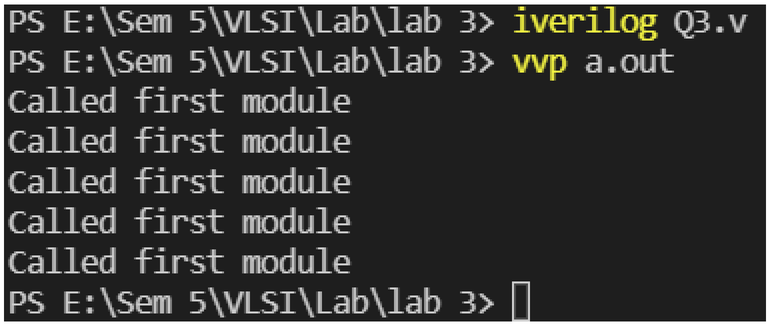


```
PS E:\Sem 5\VLSI\Lab\lab 3> iverilog Q2.v
PS E:\Sem 5\VLSI\Lab\lab 3> vvp a.out
genvar_tb.gen1[1].gen2[1]1
genvar_tb.gen1[2].gen2[2]2
genvar_tb.gen1[2].gen2[1]2
genvar_tb.gen1[3].gen2[3] 3
genvar_tb.gen1[3].gen2[2] 3
genvar_tb.gen1[3].gen2[1] 3
genvar_tb.gen1[4].gen2[4] 4
genvar_tb.gen1[4].gen2[3] 4
genvar_tb.gen1[4].gen2[2] 4
genvar_tb.gen1[4].gen2[1] 4
PS E:\Sem 5\VLSI\Lab\lab 3>
```

Q3)Find the functionality of the keyword “generate”? Write a small code using generate.

A generate for loop is used to create one or more instances of items that can be placed within a Verilog module. The loop is essentially the same as a regular Verilog HDL for loop, but with these limitations: The index loop variable must be a genvar.

```
module first();
    initial begin
        $display("Called first module");
    end
endmodule
module generatef();
    generate
        genvar i;
        for(i=1;i<=5;i=i+1) begin
            first f();
        end
    endgenerate
endmodule
module Ques3_tb();
    generatef g();
endmodule
```



```
PS E:\Sem 5\VLSI\Lab\lab 3> iverilog Q3.v
PS E:\Sem 5\VLSI\Lab\lab 3> vvp a.out
Called first module
Called first module
Called first module
Called first module
Called first module
PS E:\Sem 5\VLSI\Lab\lab 3> 
```