EE2016 : Microprocessor Theory and Lab

Lab Experiment # 1

Implementation and Performance Comparison of 4-bit Serial-Parallel Multiplier with Booth's Algorithm in FPGA

(Xilinx's Spartan 3E Board)

Batch 35

Sarathchandra Koppera EE21B073

Vikas Gollapalli EE21B051

Contents

1 Aim of the Experiment

1. To study the 4-bit serial-parallel multiplier and Booth’s algorithm for multiplication

2. To implement both the above in FPGA platform

3. To demonstrate its working given a test set, by writing a test bench code to display the output in LEDs

4. To compare the performance of both the algorithms in terms of number of clock cycles, given the same set of multiplicand and multiplier.

2 Code

Verilog code for Serial –Parallel Multiplier

```
module noradd(
  input [3:0]A,
  input [3:0]B,
  output reg [7:0]product
    );
always@(A or B)
begin

   product = 8'b00000000;
   if(B[0] == 1'b1)
     product = product + (A<<0);
   if(B[1] == 1'b1)
     product = product + (A<<1);
   if(B[2] == 1'b1)
     product = product + (A<<2);
   if(B[3] == 1'b1)
     product = product + (A<<3);

end
endmodule
```

Test Bench for Serial –Parallel Multiplier

```
module testi;

 // Inputs
 reg [3:0] A;
 reg [3:0] B;

 // Outputs
 wire [7:0] product;

 // Instantiate the Unit Under Test (UUT)
 noradd uut (
  .A(A),
  .B(B),
  .product(product)
 );

 initial begin
  // Initialize Inputs
  A = 0;
  B = 0;

  // Wait 100 ns for global reset to finish
  #100;

  // Add stimulus here
  A = 1;
  B = 1;
 end

endmodule
```

Verilog Code for Booth's Algorithm

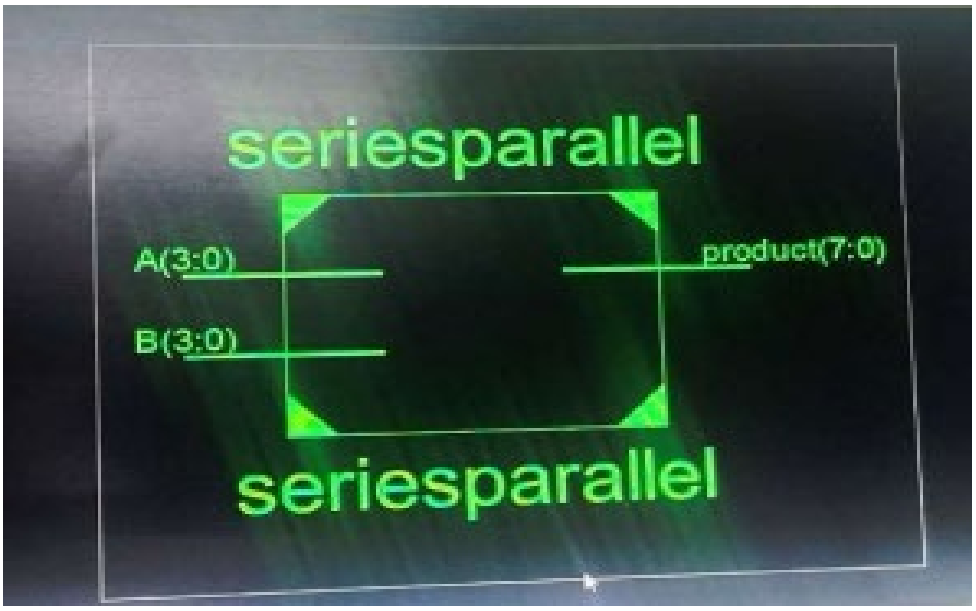
```
module BOOTHS_MULTIPLIER(
    output [7:0] prod,
    output busy,
    input [3:0] mc,
    input [3:0] mp,
    input clk,
    input start
    );
 reg [3:0] A, Q, M; // all registers are of 4 bits
 reg Q_1;
 reg [2:0] count;
 wire [3:0] sum, difference;

 always @(posedge clk)
 begin
  if (start)
  begin
   A <= 1'b0;
   M <= mc;
   Q <= mp;
   Q_1 <= 1'b0; // bit written to the left of lsb of number to be multiplied
   count <= 3'b0;
  end
  else
  begin
   case ({Q[0], Q_1})
    2'b0_1 : {A, Q, Q_1} <= {sum[3], sum, Q};
    2'b1_0 : {A, Q, Q_1} <= {difference[3], difference, Q};
    default: {A, Q, Q_1} <= {A[3], A, Q};
   endcase
   count <= count + 1'b1;
  end
 end
 alu adder(sum, A, M, 0); // adder
 alu subtracter(difference, A,~M, 1); //subtractor using 2's compliment
 assign prod = {A, Q}; // make it fill up the arguments
 assign busy = (count < 5);
endmodule

// The following is an alu.It is an adder, but capable of subtraction:
// Recall that subtraction means adding the two's complement-- a - b = a + (-b) = a
// + (inverted b + 1)
// The 1 will be coming in as cin (carry-in)
module alu(out, a, b, cin);
 output [3:0] out;
 input [3:0] a;
 input [3:0] b;
 input cin;
 assign out = a + b + cin;
endmodule
```

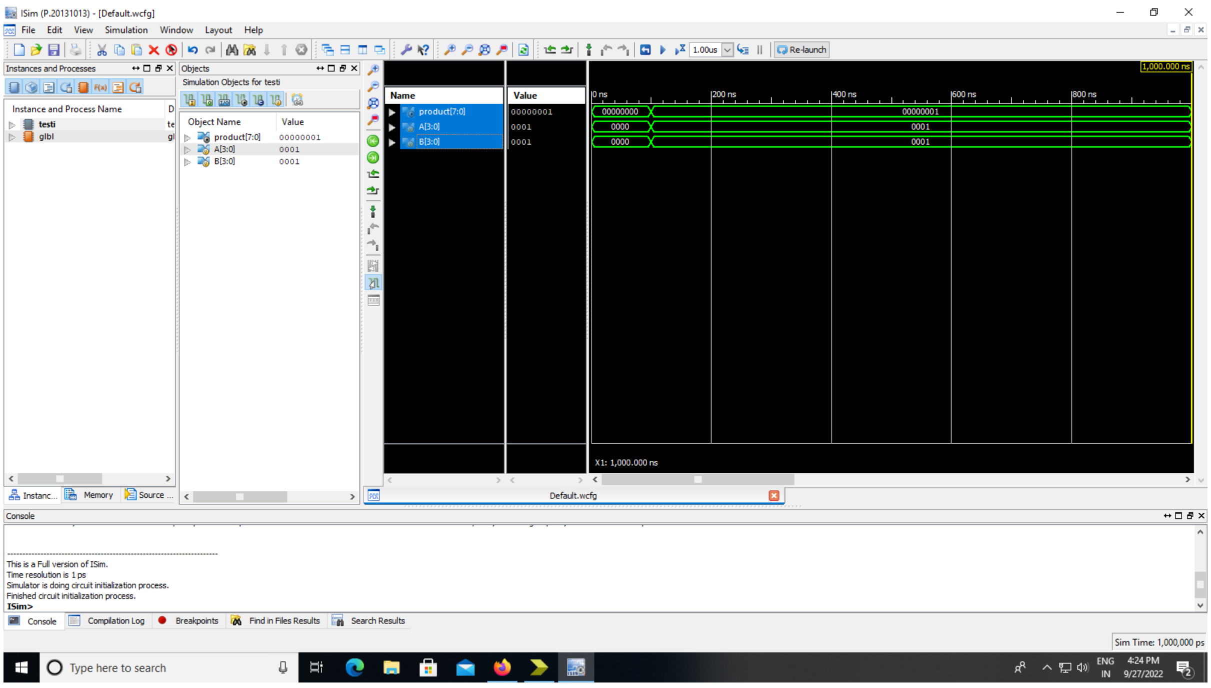
Test Bench for Booth’s Algorithm

```
module BOOTHS_MULTIPLIER_TB;

 // Inputs
 reg [3:0] mc;
 reg [3:0] mp;
 reg clk;
 reg start;

 // Outputs
 wire [7:0] prod;
 wire busy;

 // Instantiate the Unit Under Test (UUT)
 BOOTHS_MULTIPLIER uut (
  .prod(prod),
  .busy(busy),
  .mc(mc),
  .mp(mp),
  .clk(clk),
  .start(start)
 );

 initial begin
  // Initialize Inputs
  mc = 4'b0110;
  mp = 4'b0010;
  clk = 1;
        start = 1;
        #10 clk = ~clk;
  #10 clk = ~clk;
  start = 0;
  #10 clk = ~clk;
  #10 clk = ~clk;
  #10 clk = ~clk;
  #10 clk = ~clk;
  #10 clk = ~clk;
  #10 clk = ~clk;
  #10 clk = ~clk;
  #10 clk = ~clk;



        $finish;
 end

 initial begin
     $dumpfile("BOOTHS.vcd");
     $dumpvars(0,BOOTHS_MULTIPLIER_TB);
    end

endmodule
```

3 Results

Output for Serial –Parallel Multiplier





Instance and Process Name
testi
glbl

Object Name	Value
product[7:0]	00000001
A[3:0]	0001
B[3:0]	0001

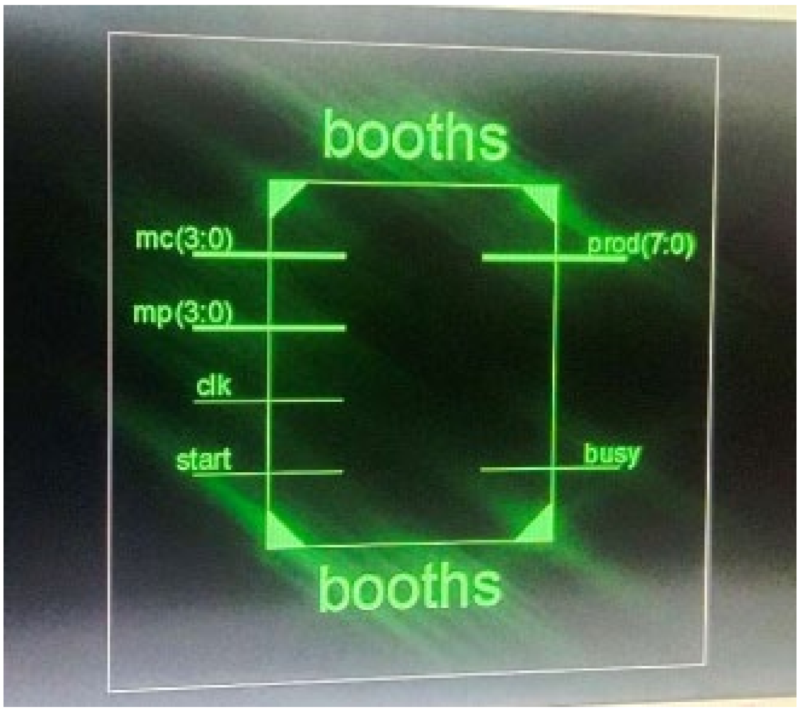
Name	Value	0 ns	1,000.000 ns
product[7:0]	00000001	00000000	00000001
A[3:0]	0001	0000	0001
B[3:0]	0001	0000	0001

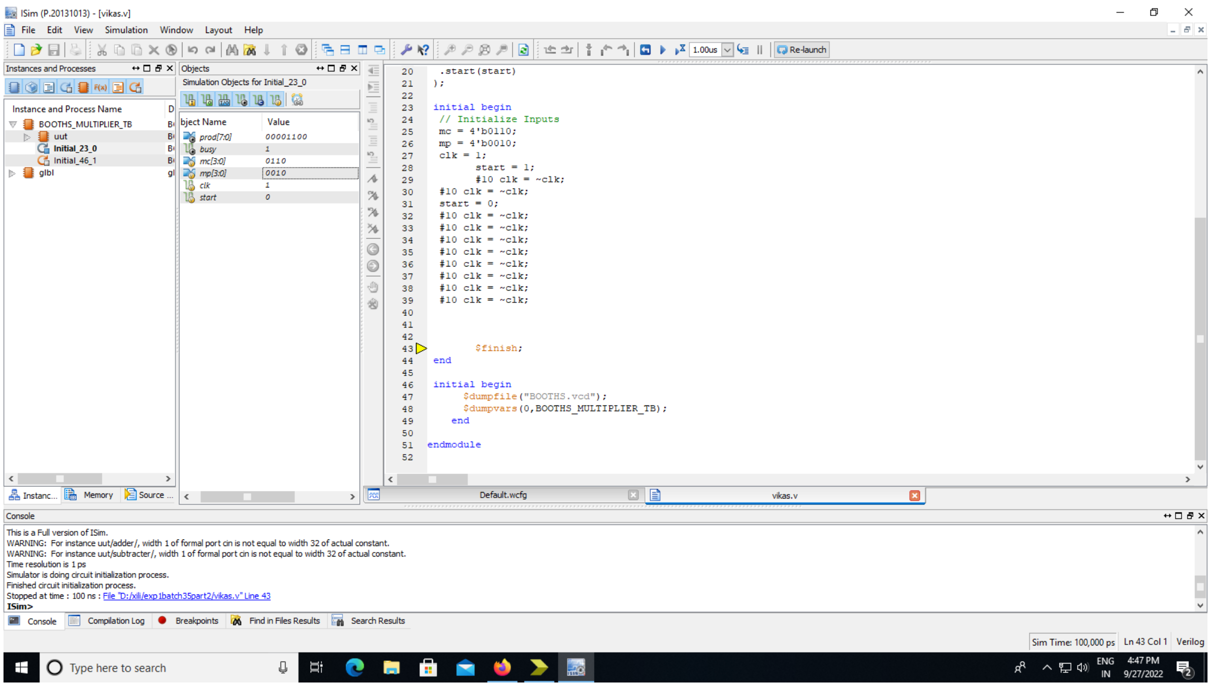
X1: 1,000.000 ns

```
---------------------------------------------------------------
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
ISim>
```

Sim Time: 1,000,000 ps

Output for Booth’s Multiplier





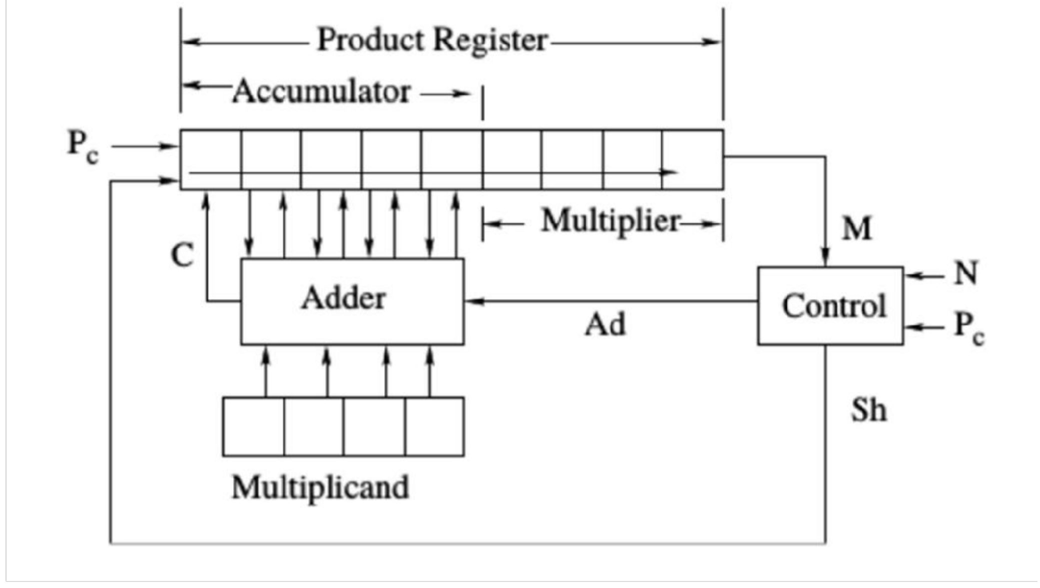
Object Name	Value
prod[7:0]	00001100
busy	1
mc[3:0]	0110
mp[3:0]	0010
clk	1
start	0

```
 .start(start)
);

initial begin
 // Initialize Inputs
 mc = 4'b0110;
 mp = 4'b0010;
 clk = 1;
        start = 1;
        #10 clk = ~clk;
 #10 clk = ~clk;
 start = 0;
 #10 clk = ~clk;
 #10 clk = ~clk;
 #10 clk = ~clk;
 #10 clk = ~clk;
 #10 clk = ~clk;
 #10 clk = ~clk;
 #10 clk = ~clk;
 #10 clk = ~clk;



        $finish;
end

initial begin
     $dumpfile("BOOTHS.vcd");
     $dumpvars(0,BOOTHS_MULTIPLIER_TB);
   end

endmodule

```

```
This is a Full version of ISim.
WARNING:  For instance uut/adder/, width 1 of formal port cin is not equal to width 32 of actual constant.
WARNING:  For instance uut/subtracter/, width 1 of formal port cin is not equal to width 32 of actual constant.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
Stopped at time : 100 ns : File "D:/xili/exp1batch35part2/vikas.v" Line 43
ISim>
```

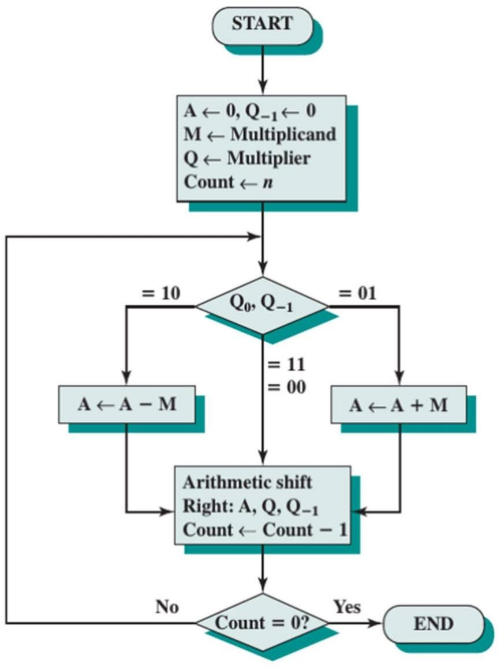
4 Explanation

Serial-Parallel Multiplier



In this algorithm , we multiply the multiplicand by least significant digit of the multiplier, shift right, then repeat for the next to accumulate the sum of product , keep doing till most significant digit and add them now. The only difference is that for each multiplication above, we use repeated addition and use the same accumulator to hold the product .

Booth's algorithm



With the help of the method of Booth's algorithm we can multiply two signed binary numbers in 2's complement notation. Booth's algorithm performs lesser number of additions and subtractions when compared to normal multiplication algorithm.

5 Learning Outcomes

1. We learnt the logic behind the Serial-Parallel and Booth’s algorithm and realization using the Verilog language
2. We also learnt how to implement a test bench and implement the multiplication of two numbers
3. We also learnt that booth’s algorithm is much faster than serial-parallel multiplier in terms of clock cycles. Theoretically booth’s algorithm takes n clock cycles to compute a n-bit multiplication while a serial-parallel multiplier has a time complexity of 2n. Booth’s algorithm gives correct result for signed bits while serial-parallel multiplier will give incorrect result for un signed bits.