# EE2016EXPRMNT1_FPGA
# MIRUDHULA
# EE23B046

## 1.OBJECTIVES:

1. To study the 4-bit serial-parallel multiplier and Booths 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 USED:

### 1)SERIAL PARALLEL MULTIPLIER:

**VERILOG CODE:**

```
module mult_4x4(
    input reset, clk,
    input [3:0] A, B,
    output [7:0] O, // 8-bits output
    output Finish
);
    reg [7:0] O;
    wire Finish;
    reg [3:0] State; // state machine
    reg [8:0] ACC; // Accumulator

    // logic to create 2 phase clocking when starting
    assign Finish = (State == 9) ? 1'b1 : 1'b0; // Finish Flag

    always @(posedge clk or posedge reset)
    begin
        if (reset)
        begin
            State <= 0;
            ACC <= 0;
            O <= 0;
        end
        else if (State == 0)
        begin
```

```
            ACC[8:4] <= 0; // begin cycle
            ACC[3:0] <= A; // Load A (one of our inputs)
            State <= 1;
        end
        else if (State == 1 || State == 3 || State == 5 || State == 7)
        // add/shift State
        begin
            if (ACC[0] == 1'b1)
            begin
                // add multiplicand
                ACC[8:4] <= {1'b0, ACC[7:4]} + B;
                State <= State + 1;
            end
            else
            begin
                ACC <= {1'b0, ACC[8:1]}; // shift right
                State <= State + 2;
            end
        end
        else if (State == 2 || State == 4 || State == 6 || State == 8)
        // shift State
        begin
            ACC <= {1'b0, ACC[8:1]}; // shift right
            State <= State + 1;
        end
        else if (State == 9)
        begin
            State <= 0;
            O <= ACC[7:0]; // loading data of accumulator in output
        end
    end
endmodule
```

**TEST BENCH:**

```
`timescale 100ns/100ps

module test;
// Signals
reg reset, clk;
reg [3:0] A, B; // 4-bit inputs for the multiplier
// Outputs
wire [7:0] O; // 8-bit output
wire Finish;
```

```
// Device Under Test (DUT)
mult_4x4 DUT (reset,clk,A,B,O,Finish);

// Clock generation
initial begin
   clk = 0;
end

always begin
   #10 clk = ~clk; // Toggle the clock every 10 ns
end

initial begin
   // Initialize signals
   reset = 1;
   A = 4'b0000;
   B = 4'b0000;

   // Apply reset
   #20 reset = 0; // Release reset after 20 ns

   // Set inputs
   #10 A = 4'b0011; // Load input A
   B = 4'b0111; // Load input B

   // Monitor output
   $monitor("At time %t: A = %b, B = %b, O = %b, Finish = %b", $time, A, B, O, Finish);

   // Wait for sufficient time to observe the results
   #400;

   // Finish the simulation
   $finish;
end
endmodule
```

**CONSTRAINTS:**

```
# Clock signal
set_property -dict { PACKAGE_PIN N11    IOSTANDARD LVCMOS33 } [get_ports { clk }];

# Switches
set_property -dict { PACKAGE_PIN L5    IOSTANDARD LVCMOS33 } [get_ports { A[0] }];#LSB
set_property -dict { PACKAGE_PIN L4    IOSTANDARD LVCMOS33 } [get_ports { A[1] }];
set_property -dict { PACKAGE_PIN M4    IOSTANDARD LVCMOS33 } [get_ports { A[2] }];
set_property -dict { PACKAGE_PIN M2    IOSTANDARD LVCMOS33 } [get_ports { A[3] }];
set_property -dict { PACKAGE_PIN M1    IOSTANDARD LVCMOS33 } [get_ports { reset }];
set_property -dict { PACKAGE_PIN N6    IOSTANDARD LVCMOS33 } [get_ports { B[0] }];
set_property -dict { PACKAGE_PIN T7    IOSTANDARD LVCMOS33 } [get_ports { B[1] }];
set_property -dict { PACKAGE_PIN P8    IOSTANDARD LVCMOS33 } [get_ports { B[2] }];
set_property -dict { PACKAGE_PIN M6    IOSTANDARD LVCMOS33 } [get_ports { B[3] }];#MSB

# LEDs
set_property -dict { PACKAGE_PIN J3    IOSTANDARD LVCMOS33 } [get_ports { O[0] }];#LSB
set_property -dict { PACKAGE_PIN H3    IOSTANDARD LVCMOS33 } [get_ports { O[1] }];
set_property -dict { PACKAGE_PIN J1    IOSTANDARD LVCMOS33 } [get_ports { O[2] }];
set_property -dict { PACKAGE_PIN K1    IOSTANDARD LVCMOS33 } [get_ports { O[3] }];
set_property -dict { PACKAGE_PIN L3    IOSTANDARD LVCMOS33 } [get_ports { O[4] }];
set_property -dict { PACKAGE_PIN L2    IOSTANDARD LVCMOS33 } [get_ports { O[5] }];
set_property -dict { PACKAGE_PIN K3    IOSTANDARD LVCMOS33 } [get_ports { O[6] }];
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { O[7] }];
set_property -dict { PACKAGE_PIN T9    IOSTANDARD LVCMOS33 } [get_ports { finish
}];#MSB
```

**2)BOOTH'S MULTIPLIER:**

**VERILOG CODE:**

```
module multiplier(prod, busy, mc, mp, clk, start, reset);
    output reg [7:0] prod;
    output reg busy;
    input [3:0] mc, mp;
    input clk, start, reset;
    reg [3:0] A, Q, M;
    reg Q_1;
    reg [2:0] count;
    wire [3:0] sum, difference;
```

```
    always @(posedge clk or posedge reset) begin
        if (reset) begin
            A <= 0;
            Q <= 0;
            Q_1 <= 1'b0;
            count <= 0;
            busy <= 1'b0;
        end
        else if (start) begin
            A <= 0;
            M <= mc;
            Q <= mp;
            Q_1 <= 1'b0;
            count <= 0;
            busy <= 1'b1;
        end
        else if (busy) begin
            case ({Q[0], Q_1})
                2'b01 : {A, Q, Q_1} <= {sum[3], sum, Q};
                2'b10 : {A, Q, Q_1} <= {difference[3], difference, Q};
                default: {A, Q, Q_1} <= {A[3], A, Q};
            endcase
            count <= count + 1'b1;
            if (count == 3'b100) begin
                busy <= 1'b0;
                prod <= {A, Q};
            end
        end
    end

    alu adder (sum, A, M, 1'b0);
    alu subtracter (difference, A, ~M, 1'b1);
endmodule

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:**

```
`timescale 1ns / 1ps

module tb_multiplier;

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

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

    // Instantiate the multiplier
    multiplier uut (
        .prod(prod),
        .busy(busy),
        .mc(mc),
        .mp(mp),
        .clk(clk),
        .start(start),
        .reset(reset)
    );

    // Clock generation
    always #5 clk = ~clk; // 100 MHz clock

    initial begin
        // Initialize Inputs
        mc = 0;
        mp = 0;
        clk = 0;
        start = 0;
        reset = 0;

        // Reset the system
        reset = 1;
        #10;
        reset = 0;
```

```
        // Apply inputs
        mc = 4'b0011; // Multiplicand
        mp = 4'b0111; // Multiplier
        start = 1;
        #10;
        start = 0;

        // Wait for the operation to complete
        wait (busy == 0);

        // Check the result
        $display("Product = %d", prod);

        // End simulation
        #100;
        $stop;
    end
endmodule
```

**CONSTRAINTS:**

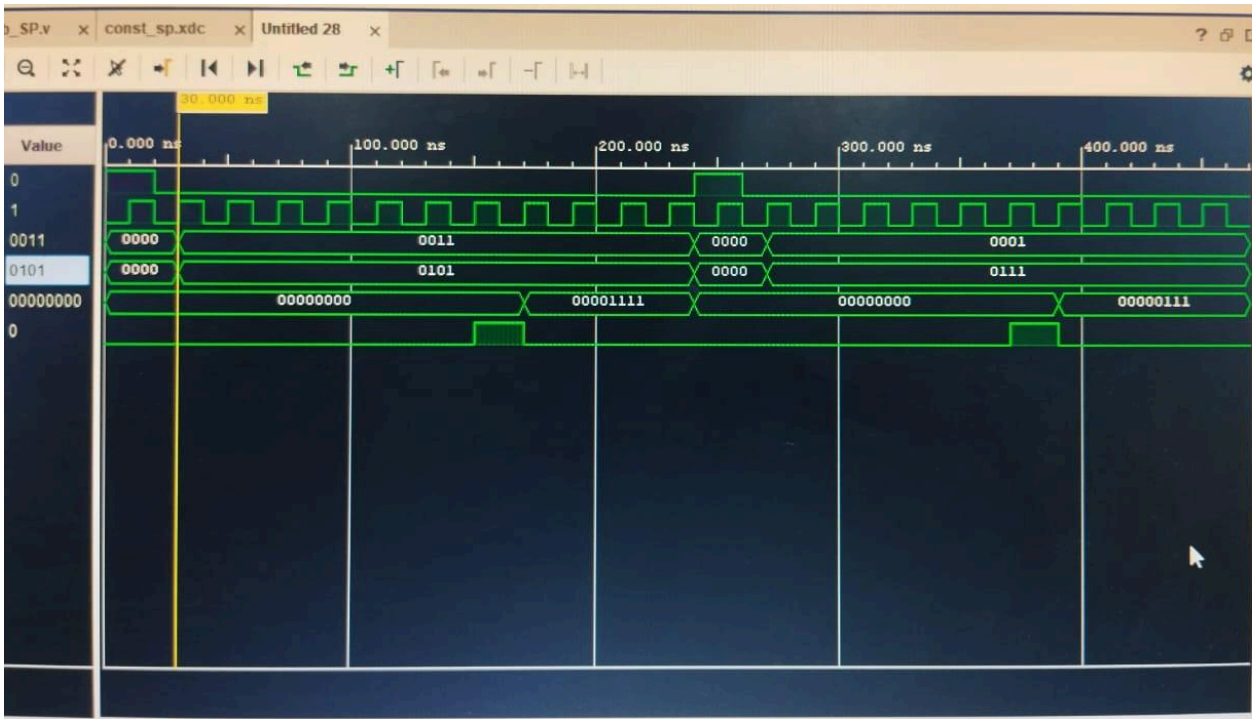
```
# Clock signal
set_property -dict { PACKAGE_PIN N11    IOSTANDARD LVCMOS33 } [get_ports { clk }];

# Switches
set_property -dict { PACKAGE_PIN L5    IOSTANDARD LVCMOS33 } [get_ports { mp[0] }];#LSB
set_property -dict { PACKAGE_PIN L4    IOSTANDARD LVCMOS33 } [get_ports { mp[1] }];
set_property -dict { PACKAGE_PIN M4    IOSTANDARD LVCMOS33 } [get_ports { mp[2] }];
set_property -dict { PACKAGE_PIN M2    IOSTANDARD LVCMOS33 } [get_ports { mp[3] }];
set_property -dict { PACKAGE_PIN M1    IOSTANDARD LVCMOS33 } [get_ports { start }];
set_property -dict { PACKAGE_PIN N3    IOSTANDARD LVCMOS33 } [get_ports { reset }];
set_property -dict { PACKAGE_PIN N6    IOSTANDARD LVCMOS33 } [get_ports { mc[0] }];
set_property -dict { PACKAGE_PIN T7    IOSTANDARD LVCMOS33 } [get_ports { mc[1] }];
set_property -dict { PACKAGE_PIN P8    IOSTANDARD LVCMOS33 } [get_ports { mc[2] }];
set_property -dict { PACKAGE_PIN M6    IOSTANDARD LVCMOS33 } [get_ports { mc[3]
}];#MSB

# LEDs
set_property -dict { PACKAGE_PIN J3    IOSTANDARD LVCMOS33 } [get_ports { prod[0]
}];#LSB
set_property -dict { PACKAGE_PIN H3    IOSTANDARD LVCMOS33 } [get_ports { prod[1] }];
set_property -dict { PACKAGE_PIN J1    IOSTANDARD LVCMOS33 } [get_ports { prod[2] }];
set_property -dict { PACKAGE_PIN K1    IOSTANDARD LVCMOS33 } [get_ports { prod[3] }];
```

```
set_property -dict { PACKAGE_PIN L3    IOSTANDARD LVCMOS33 } [get_ports { prod[4] }];
set_property -dict { PACKAGE_PIN L2    IOSTANDARD LVCMOS33 } [get_ports { prod[5] }];
set_property -dict { PACKAGE_PIN K3    IOSTANDARD LVCMOS33 } [get_ports { prod[6] }];
set_property -dict { PACKAGE_PIN K2    IOSTANDARD LVCMOS33 } [get_ports { prod[7] }];
set_property -dict { PACKAGE_PIN K5    IOSTANDARD LVCMOS33 } [get_ports { busy
}];}];#MSB
```

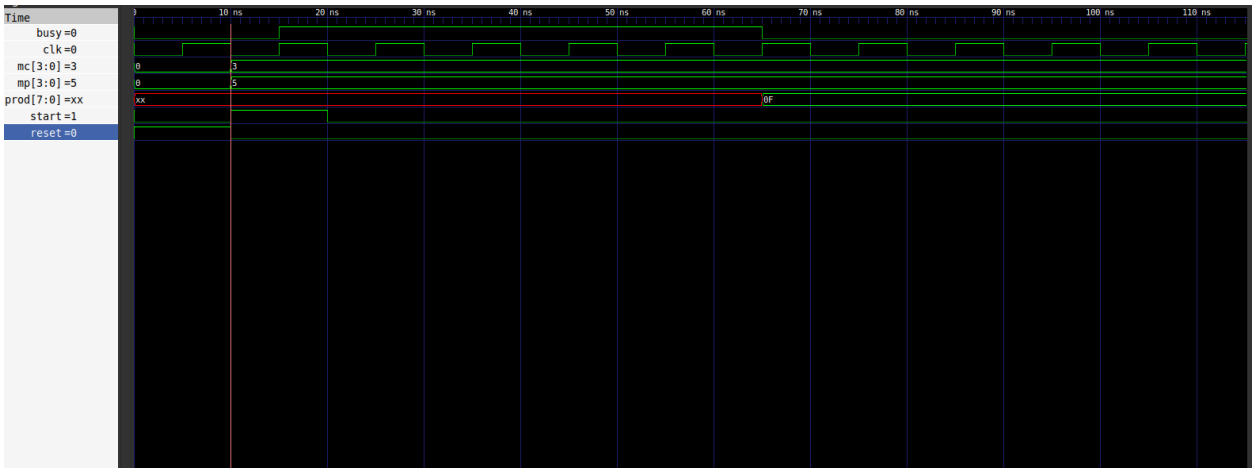
**PROCEDURE USED:**

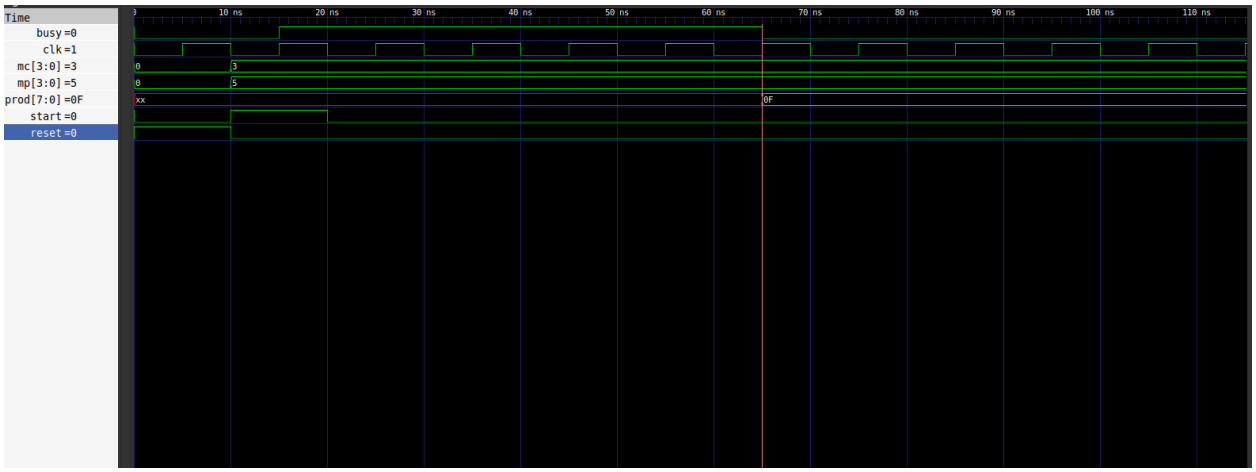
- After adding the source file (i.e, multiplier.v and the test_bench), we did run synthesis to get the result in gtkwave format.
- After analyzing this, we added the constraints file for the fpga demonstration.
- We started to generate bitStream and connected the fpga board to flash the program in it.
- Then demonstrated it to TA.

**OBSERVATION:**




For the inputs 0011 and 0101, the corresponding output is 00001111.
To compute this it took 7 clock cycles.i.e, 140.000 ns.




Here for the inputs 0011 and 0101, the corresponding outputs is 00010101.
To compute this it took around 6 clock cycles.i.e, 55.000 ns.

Thus, we can conclude that a booth's multiplier takes less number of clock cycles to compute the result than a serial parallel multiplier.