Experiment 1: Implementation and Performance Comparison of 4-bit Serial-Parallel Multiplier with Booth's Algorithm in FPGA (Xilinx's Spartan 3E Board)

Santhosh S P ee21b119 Batch 19

1 Serial-parallel multiplier

1.1 Verilog code for design of serial-parallel multiplier:

```
‘timescale 1ns / 1ps
module mul_4x4(
    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);

       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
```

1.2 Verilog test bench code:

```
module mult_4x4_TB;
    // Inputs
    reg [3:0] A;
    reg [3:0] B;

    // Outputs
    wire [7:0] Product;
    // Instantiate the Unit Under Test (UUT)
    mul_4x4 uut (
    .A(A),
    .B(B),
    .Product(Product)
    );
    initial
        begin
        // Initialize Inputs
        A = 4’b1101;
        B = 4’b1001;
        #100;
        end
endmodule
```

1.3 HDL schematic of the serial-parallel multiplier:

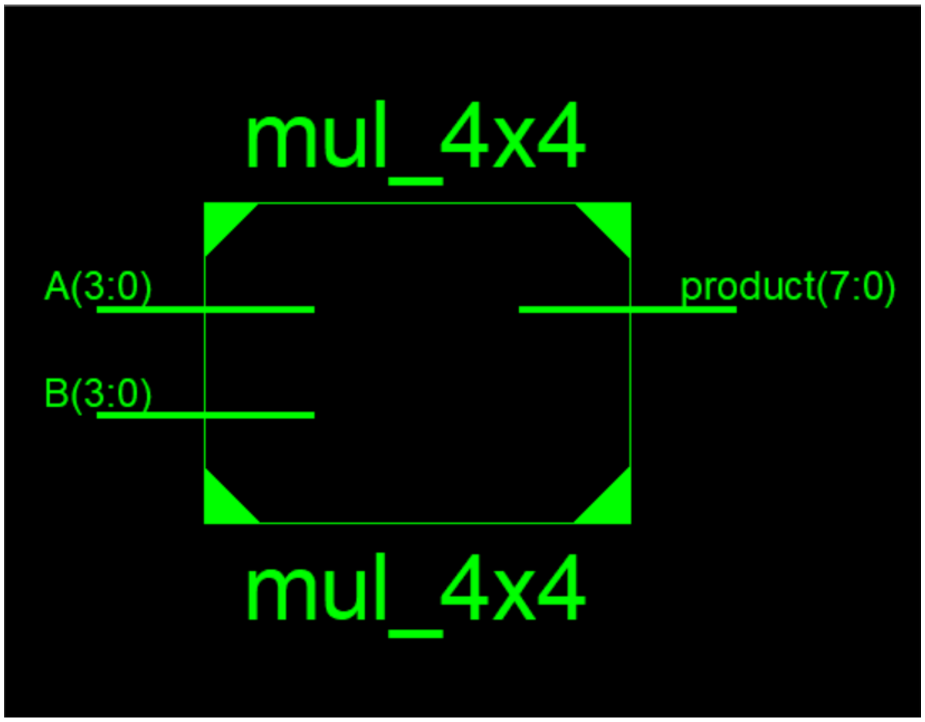
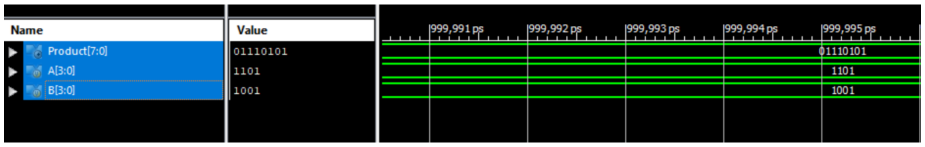


Figure 1: Serial-parallel multiplier

1.4 Output waveforms:



Name	Value
Product[7:0]	01110101
A[3:0]	1101
B[3:0]	1001

Figure 2: A=1101, B=1001

2 Booth's Algorithm:

2.1 Verilog Design Code for Booth's algorithm:

```
‘timescale 1ns / 1ps
module multiplier(prod, busy, mc, mp, clk, start);

    output [7:0] prod; // ouput 8 bits
    output busy;
    input [3:0] mc, mp; // input 4 bits
    input clk, 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 <= 4'b0000;
     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,1'b0); // adder
alu subtracter(difference, A, ~M, 1'b1); //subtractor using 2's        compliment
assign prod = {A, Q}; // make it fill up the arguments
assign busy = (count < 5);
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
```

2.2 Verilog test bench code for Booth's algorithm

```
‘timescale 1ns / 1ps

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'b1010;
mp = 4'b1110;
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
```

2.3 HDL schematic:

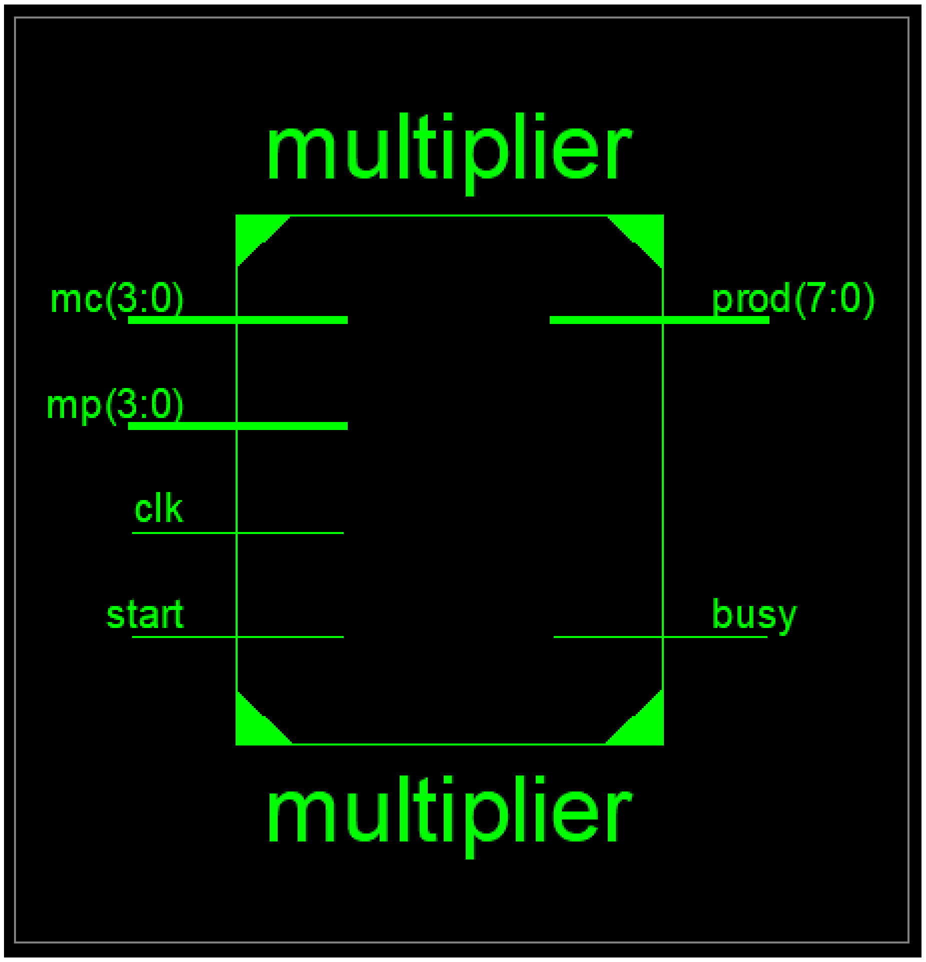


Figure 3: Schematic

2.4 Output waveforms:

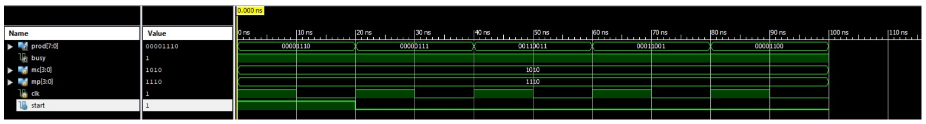


Figure 4: mc=1010, mp=1110