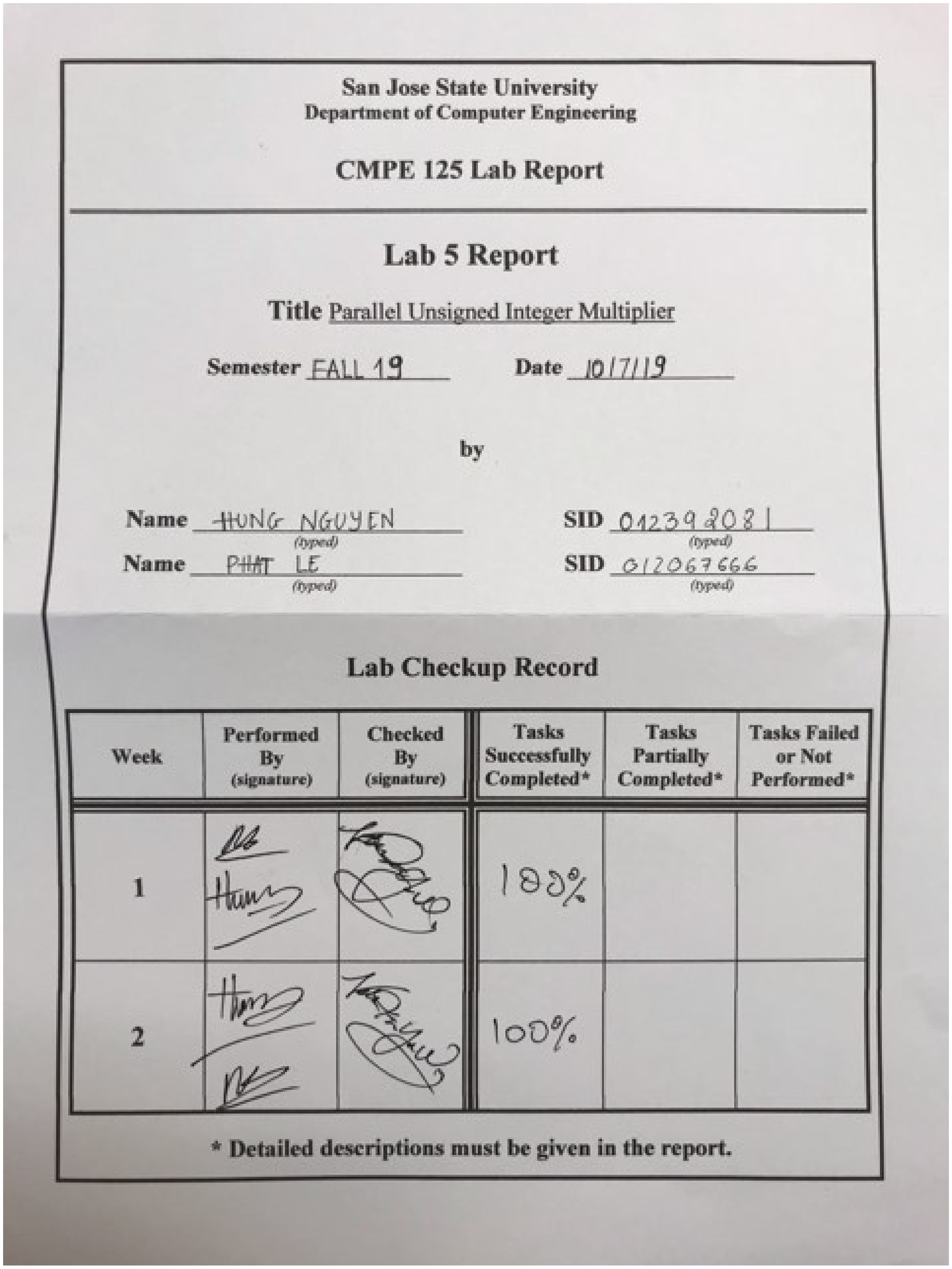

San Jose State University
Department of Computer Engineering

CMPE 125 Lab Report

Lab 5 Report

Title Parallel Unsigned Integer Multiplier

Semester FALL 19 **Date** 10/7/19

by

Name	HUNG NGUYEN *(typed)*	**SID**	012392081 *(typed)*
Name	PHAT LE *(typed)*	**SID**	012067666 *(typed)*

Lab Checkup Record

Week	Performed By (signature)	Checked By (signature)	Tasks Successfully Completed*	Tasks Partially Completed*	Tasks Failed or Not Performed*
1	[illegible]	[illegible]	100%		
2	[illegible]	[illegible]	100%		

*** Detailed descriptions must be given in the report.**

Introduction

The purpose of this lab is to get familiar with both combinational and sequential building blocks using flip flop registers and pipeline concept. Specifically, the main task is to create one combinational 4-bit multiplier and a sequential 4-bit multiplier using two-stage pipelined technique described in Lecture Slides 6.

Design Methodology

The first task is to design a multiplier that can multiply two 4-bit unsigned and write a self-checking testbench in Verilog to verify the design functionality. The design need to follow the parallel architecture described in Lecture Slides 6. Specifically, when multiplying A with B, we first AND A with every bit of B. There are four AND operations. Each AND operation need to be shifted to the left by 1 bit increment which produces four 8-bit numbers. The final result is the sum of four 8-bit numbers which can be done by using two slices of 4-bit CLA adders from previous lab.

Table 1 : List of modules for Task 1

Module	Function
unsigned_integer_multiplier	This is the top level module that requires two 4-bit inputs (A,B) and results 8-bit output (P).
AND	This module apply AND operation on A with each bit of B.
shift	This module shift left each output from AND module by 1 bit increment resulting 8-bit outputs.
CLA	This module is used to add the output produced after shift operation. Since they are 8-bit number and the CLA only do 4-bit operation, CLA module will add 4 bit at a time from left to right. Furthermore, the carry out from left CLA is the carry in from right CLA.
unsigned_integer_multiplier_tb	This module test all possible input cases and compare the output produced by the top level module with the result from Verilog multiplication operator *.

Table 2 : Shift operation for shift module

index	operation	pp[3:0]	out[7:0]
00	NO shift	a b c d	0 0 0 0 a b c d
01	Shift left 1 bit	a b c d	0 0 0 a b c d 0
10	Shift left 2 bit	a b c d	0 0 a b c d 0 0
11	Shift left 3 bit	a b c d	0 a b c d 0 0 0

The second task of the lab is to apply pipelining technique to Task 1 so that it becomes a two-stage pipelined integer multiplier. In other words, the operation of our multiplier is now sequential which would be controlled by clock input, reset input, and enable input. To make the design sequential, the multiplier is required to attach input / output registers and another stage registers in the middle of the operation so that it becomes a two-stage pipelined.

Table 2 : List of module for task 2

Module	Function
unsigned_integer_multiplier	This module have almost the same functionality as Task 1 except it also include flopenr module as insert stage registers.
flopenr	This module could be used as insert register since it has flip flop functionality (clk, en, reset). Use this for our insert stages
digit_seperator	Since the maximum decimal that the multiplication can get is 225, this module converts the decimal number into three separate numbers call ones, tenths, and hundreds. Use this to convert the result to BCD signal then to seven-segments converter module.
bcd_to_7seg	This module takes the outputs of digit_seperator signal and convert them to the seven-segments signal.
led_mux	This module helps choosing which seven-segment LED is on and the output signal for each LED. Since there are three

	signals (ones, tenths, hundredths) coming from the bcd_to_7seg module, we only need to choose LED0,1,2. LED3 become high(Vcc).
button_debuncer	This module acts as the clock control input
clk_gen	This module produce 5kHz clock signal for the button_debouncer module as well as led_mux module

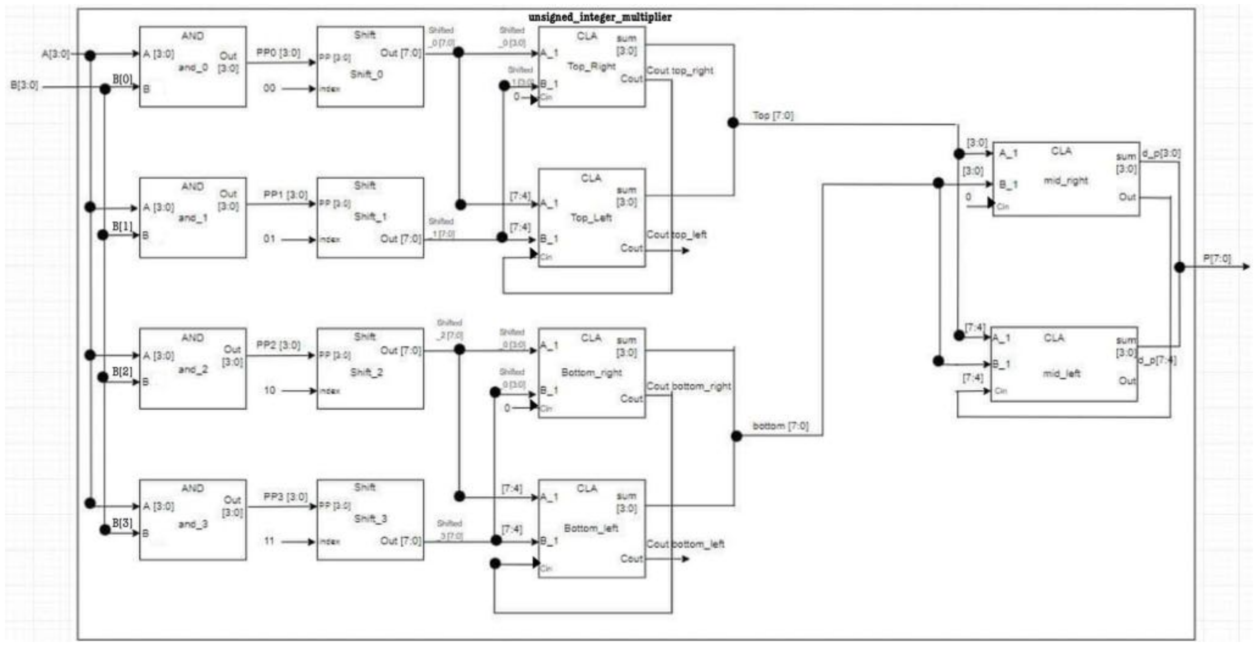

Figure 1: Block diagram for Task 1

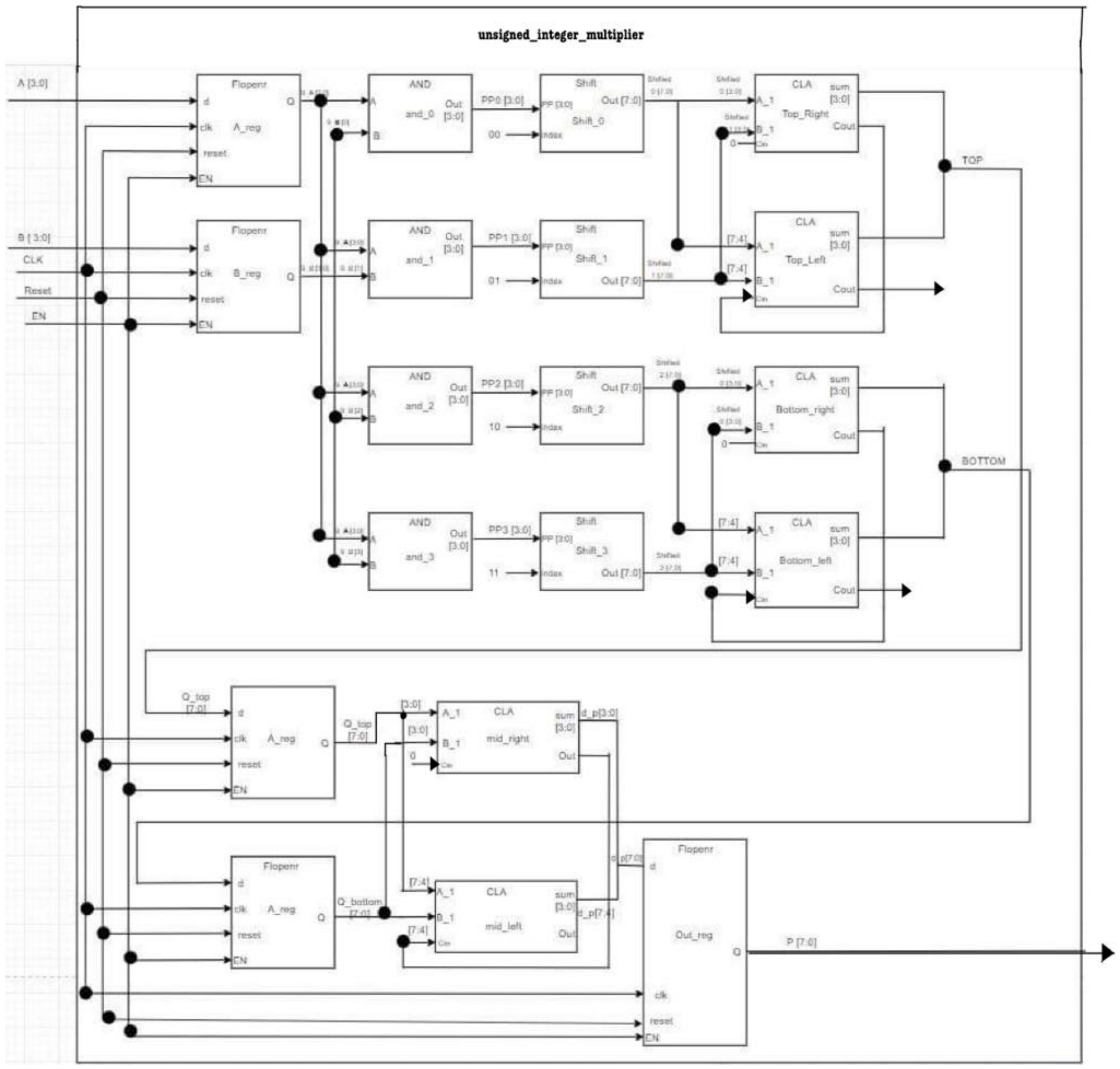

Figure 2:Block diagram for Task 2

Simulation Result

Task 1

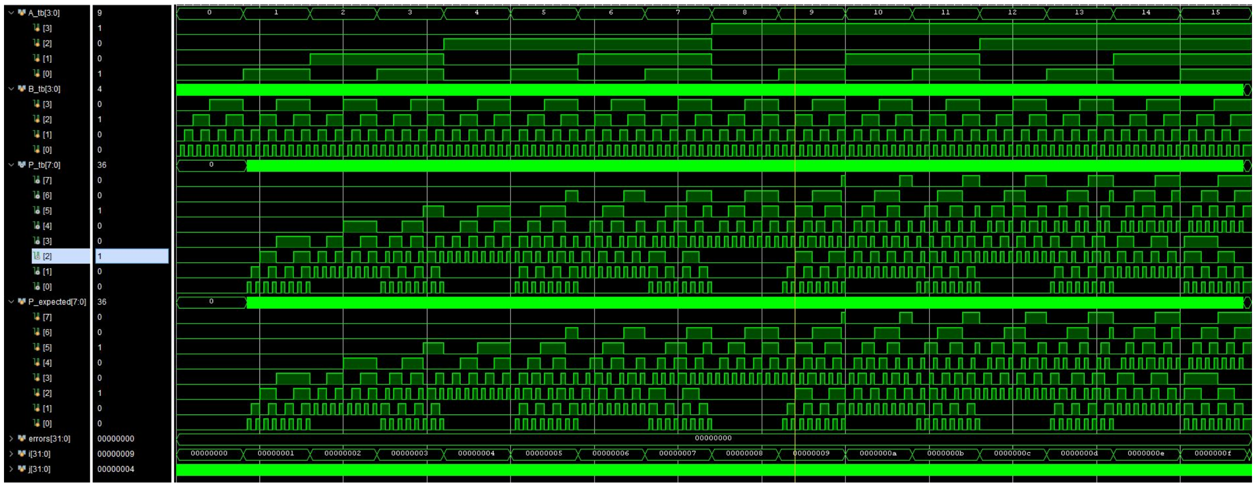

Figure 3: Simulation waveforms produced from Task 1

The testbench for this task would check every possible input cases and increment the errors variable if the testbench output is different from the expected output. According to *Figure 3* above, the cursor is at the input A(A_tb) of 9(1001), input B(B_tb) is 4(0100). We can observe that the output(P_tb) and matches the expected output (P_expected). Specifically, the expected result should be 36 because 9 * 4 = 36. Furthermore, we randomly check other cases and the result is as expected since the errors variable is 0. For this reason, the simulation process for Task 1 was successfully built.

Task 2

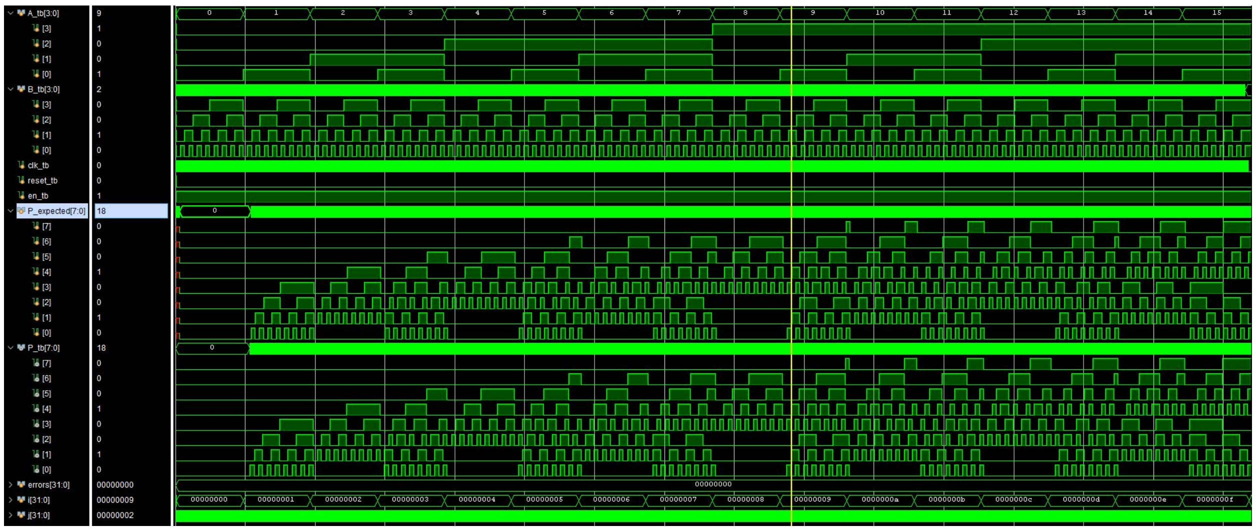

Figure 4: Simulation waveforms produced from Task 2

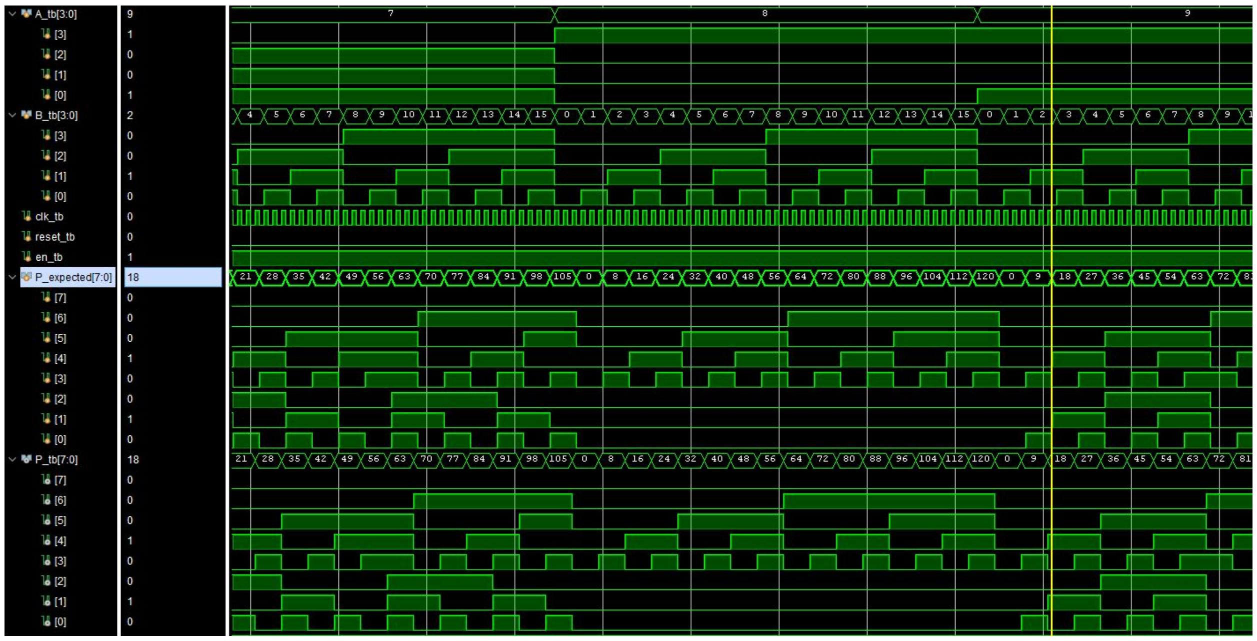

Figure 5: Zoom in version of simulation waveforms produced from Task 2

Similar to Task 1, Task 2 also test every possible input cases. The only difference between the testbenches of Task 1 and 2 is the clock(clk_tb), reset(reset_tb), enable(enable_tb) as shown in *Figure 4*. Since it is impossible to view all possible inputs and viewable clock signal at the same time in *Figure 5*, we decided to show a zoom in version of waveform in *Figure 5*. According to *Figure 5*, we can easily observe that the bit changes every three clock cycles which is correct. *Figure 5* also shows the expected result (P_expected) and actual result (P_tb) to be the same as well as the errors variable was 0. As a result, the functional verification for Task 2 was successful.

FPGA Validation

For the FPGA validation process,the rightmost four switches is four bits of A (Blue), the leftmost four switches is four bits of B(Purple), the button on the left is the clock (Green), and two switches in the middle are reset(Yellow) and enable(Red). Furthermore, the rightmost three seven-segments LEDs are the output (P). In addition, the LED above each input switch indicates whether they are on or off. For instance, the LED is lit when switch is on (1), and not lit when switch is off(0). According to *Figure 10* below, when A is 5(0101), B is 7(0111), and the enable switch is on(1), the result shown in the seven-segments LEDs is 35 after three times pressed clock button (three clock cycles). Moreover, when the reset switch is on and off one time, the output becomes 0. This result is as expected because only when it is enabled, the multiplication of 5 and 7 is 35. As soon as the reset switch is on, the output is 0. We also tested when the enable switch if off, the output remains unchanged for every clock cycle(Hold). Furthermore, we randomly try other inputs and the results turn out to be correct. For this reason, the FPGA validation process was successful.

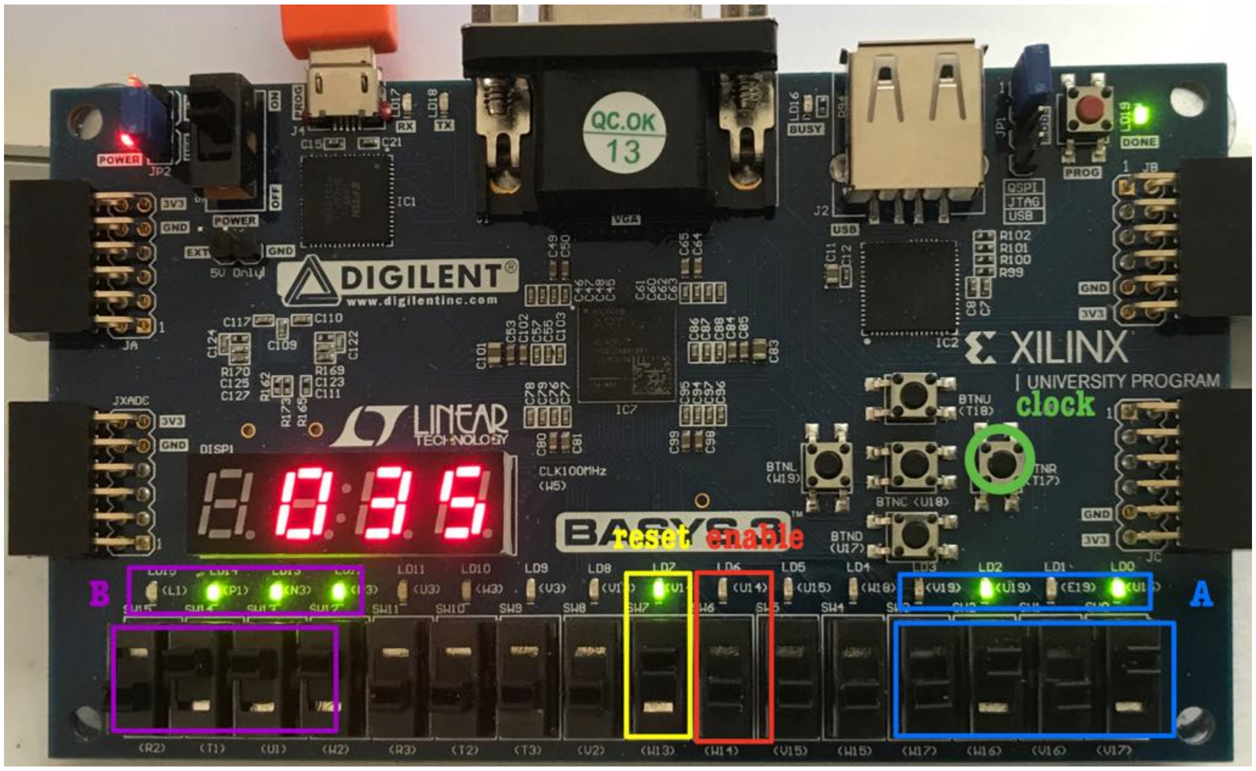

Figure 10: inputs and outputs layout for the FPGA validation process.

Conclusion

Overall, both the simulation and FPGA validation process for Task 1 and 2 was successfully performed since the result is as expected.. This lab experiment helps strengthen our knowledge on sequential and combinational functionality by closely observe the changes between reset, enable, and clock signal. We also learn that there is a minor delay when building complicated module. For instance, our clock initially only show two clock cycles for every bit changes in the simulated process; however, the actual hardware shows three clock cycle. Luckily, the TA helps us insert a delay time (#1) at the CLA adder so the result between two process would match three clock cycles.

Appendix

a. **Source Code**

Task 1

unsigned_integer_multiplier.v

```
module unsigned_integer_multiplier( input [3:0]A, input[3:0]B, output[7:0]P);
    wire[3:0] pp3,pp2, pp1, pp0;
    wire[7:0] shifted_3, shifted_2, shifted_1, shifted_0;
    wire[7:0] top, bottom, mid;
    wire Cout_top_left, Cout_top_right, Cout_bottom_left, Cout_bottom_right,
    Cout_mid_left, Cout_mid_right;
// use to and for the function (a3a2a1a0)*bi
    AND and_3(.A(A[3:0]), .B(B[3]), .out(pp3[3:0]));
    AND and_2(.A(A[3:0]), .B(B[2]), .out(pp2[3:0]));
    AND and_1(.A(A[3:0]), .B(B[1]), .out(pp1[3:0]));
    AND and_0(.A(A[3:0]), .B(B[0]), .out(pp0[3:0]));

//shift every pp by +1 bit
    shift shift_3(.pp(pp3[3:0]), .index(2'b11), .out(shifted_3));
    shift shift_2(.pp(pp2[3:0]), .index(2'b10), .out(shifted_2));
    shift shift_1(.pp(pp1[3:0]), .index(2'b01), .out(shifted_1));
    shift shift_0(.pp(pp0[3:0]), .index(2'b00), .out(shifted_0));

 //add all 4 together
 //top
    CLA top_right(.A_1(shifted_0[3:0]), .B_1(shifted_1[3:0]), .Cin(1'b0),
    .sum(top[3:0]), .Cout(Cout_top_right));
                 CLA    top_left(.A_1(shifted_0[7:4]),    .B_1(shifted_1[7:4]),
.Cin(Cout_top_right),
    .sum(top[7:4]), .Cout(Cout_top_left));
//bottom
    CLA bottom_right(.A_1(shifted_2[3:0]), .B_1(shifted_3[3:0]), .Cin(1'b0),
    .sum(bottom[3:0]), .Cout(Cout_bottom_right));
               CLA   bottom_left(.A_1(shifted_2[7:4]),   .B_1(shifted_3[7:4]),
.Cin(Cout_bottom_right),
    .sum(bottom[7:4]), .Cout(Cout_bottom_left));
//add top and bottom together
    CLA mid_right(.A_1(top[3:0]), .B_1(bottom[3:0]), .Cin(1'b0),
    .sum(P[3:0]), .Cout(Cout_mid_right));
    CLA mid_left(.A_1(top[7:4]), .B_1(bottom[7:4]), .Cin(Cout_mid_right),
    .sum(P[7:4]), .Cout(Cout_mid_left));

endmodule
```

AND.v

```
module AND(
    input [4:0]A,
    input B,
    output [3:0]out
    );
    assign out[3] = B & A[3];
    assign out[2] = B & A[2];
    assign out[1] = B & A[1];
    assign out[0] = B & A[0];
endmodule
```

shift.v

```
module shift(input [3:0]pp, input [1:0] index,
                output reg [7:0] out
    );
    always @(*)
        case (index)
            2'b00: out = {4'b0000,pp[3:0]};
            2'b01: out = {3'b000,pp[3:0],1'b0};
            2'b10: out = {2'b00,pp[3:0],2'b00};
            2'b11: out = {1'b0,pp[3:0],3'b000};
            default: out = {4'b0000,pp[3:0]};
        endcase
endmodule
```

CLA.v

```
module CLA(
                input[3:0]          A_1, B_1,
                input               Cin,
                output[3:0]         sum,
                output              Cout        );

    wire[3:0] P, G;
    wire[4:0] C;

    half_adder X0(.A(A_1[0]), .B(B_1[0]), .Y(P[0]), .carry(G[0]));
    half_adder X1(.A(A_1[1]), .B(B_1[1]), .Y(P[1]), .carry(G[1]));
    half_adder X2(.A(A_1[2]), .B(B_1[2]), .Y(P[2]), .carry(G[2]));
    half_adder X3(.A(A_1[3]), .B(B_1[3]), .Y(P[3]), .carry(G[3]));

    assign C[0] = Cin;
    assign C[1] = G[0]|(P[0]&Cin);
    assign C[2] = G[1]|(P[1]&G[0])|(P[1]&P[0]&C[0]);
    assign C[3] = G[2]|(P[2]&G[1])|(P[2]&P[1]&G[0])|(P[2]&P[1]&P[0]&C[0]);
                                            assign          C[4]            =
G[3]|(P[3]&G[2])|(P[3]&P[2]&G[1])|(P[3]&P[2]&P[1]&G[0])|(P[3]&P[2]&P[1]&P[0]
&C[0]);
```

```
    assign Cout = C[4];
    XOR2 Z3(.a(C[3]),.b(P[3]),.out(sum[3]));
    XOR2 Z2(.a(C[2]),.b(P[2]),.out(sum[2]));
    XOR2 Z1(.a(C[1]),.b(P[1]),.out(sum[1]));
    XOR2 Z0(.a(C[0]),.b(P[0]),.out(sum[0]));
endmodule
```

half_adder.v

```
module half_adder(
                input   A,B,
                output  Y,carry);

    assign Y = A^B;
    assign carry = (A&B);
endmodule
```

XOR2.v

```
module XOR2(input a,
            input b,
            output out
    );
    assign out=a^b;
endmodule
```

unsigned_integer_multiplier_tb.v

```
module unsigned_integer_multiplier_tb;
    reg [3:0] A_tb;
    reg [3:0] B_tb;
    wire [7:0] P_tb;

    unsigned_integer_multiplier DUT(.A(A_tb), .B(B_tb), .P(P_tb));
    integer i,j, errors;
    reg[7:0] P_expected;
    initial
    begin
        errors = 0;
        for(i=0; i< 16; i=i+1)
        begin
            A_tb = i;
            for(j=0; j< 16; j=j+1)
            begin
                B_tb = j;
                P_expected = A_tb*B_tb;
                #5;
                if(P_tb != P_expected)
                begin
                      $display("Failed when A=%b, B=%b, the P_expected is %d, but the
actual value P is %d",
                                A_tb, B_tb, P_expected, P_tb);
                    errors = errors+1;
                end
                #5;
            end
        end
        if(!errors) $display("Test finished with %0d errors", errors);
    #10 $finish;
    end
endmodule
```

Task 2

interger_multiplier_fpga.v

```
module integer_multiplier_fpga(
        input wire [3:0] A,
        input wire [3:0] B,
        input wire rst,
        input wire en,
        input wire clock,
        input wire clk100mHz,
        output wire [3:0] LEDSEL,
        output wire [7:0] LEDOUT,
        output wire [3:0] A_out,B_out,
        output wire rst_out, en_out
```

```
            );
            wire [7:0] P_out;
            wire DONT_USE;
            wire clk_5KHz;
            wire clock_button, debounced_clk_button;
            wire debounced_rst_button;
            wire [3:0] ones, tenths, hundredths;
            wire [7:0] LED2, LED1, LED0;
            supply1 [7:0] vcc;

            //set input LED
            assign A_out = A;
            assign B_out = B;
            assign rst_out = rst;
            assign en_out = en;

            clk_gen     clk(.clk100MHz(clk100mHz),
                        .rst(rst),
                        .clk_4sec(DONT_USE),
                        .clk_5KHz(clk_5KHz));

            button_debouncer    clk_button(.clk(clk_5KHz),
                                           .button(clock),

        .debounced_button(debounced_clk_button));
            //button_debouncer    rst_button(.clk(clk_5KHz),
              //                           .button(rst),
                                                                         //
        .debounced_button(debounced_rst_button));

            unsigned_integer_multiplier     UIM(.A(A),
                                                .B(B),
                                                .P(P_out),

        .Clk(debounced_clk_button),
                                                .Reset(rst),
                                                .En(en));

            digit_seperator DiSep       (.total(P_out),
                                         .ones(ones),
                                         .tenths(tenths),
                                         .hundredths(hundredths));

            bcd_to_7seg     hundreds(.BCD(hundredths),
                                     .s(LED2));
            bcd_to_7seg     tens(.BCD(tenths),
                                     .s(LED1));
            bcd_to_7seg     one(.BCD(ones),
                                     .s(LED0));

            led_mux         LED(.clk(clk_5KHz),
                                .rst(rst),
                                .LED3(vcc),
                                .LED2(LED2),
                                .LED1(LED1),
                                .LED0(LED0),
                                .LEDSEL(LEDSEL),
                                .LEDOUT(LEDOUT));

        endmodule
```

clk_gen.v

```
module clk_gen (
        input  wire clk100MHz,
        input  wire rst,
        output reg  clk_4sec,
        output reg  clk_5KHz
    );

    integer count1, count2;

    always @ (posedge clk100MHz) begin
        if (rst) begin
            count1 = 0;
            count2 = 0;
            clk_5KHz = 0;
            clk_4sec = 0;
        end
        else begin
            if (count1 == 200000000) begin
                clk_4sec = ~clk_4sec;
                count1 = 0;
            end

            if (count2 == 10000) begin
                clk_5KHz = ~clk_5KHz;
                count2 = 0;
            end

            count1 = count1 + 1;
            count2 = count2 + 1;
        end
    end

endmodule
```

button_debouncer.v

```
module button_debouncer #(parameter depth = 16) (
        input  wire clk,                 /* 5 KHz clock */
         input  wire button,               /* Input button from constraints
*/
        output reg  debounced_button
    );

    localparam history_max = (2**depth)-1;

    /* History of sampled input button */
    reg [depth-1:0] history;

    always @ (posedge clk) begin
        /* Move history back one sample and insert new sample */
        history <= { button, history[depth-1:1] };

          /* Assert debounced button if it has been in a consistent state
throughout history */
        debounced_button <= (history == history_max) ? 1'b1 : 1'b0;
    end

endmodule
```

unsigned_integer_multiplier.v

```
module     unsigned_integer_multiplier(     input     [3:0]A,     input[3:0]B,
output[7:0]P,
        input Clk,input Reset, input En);
    wire[3:0] pp3,pp2, pp1, pp0;
    wire[3:0] q_A, q_B;
    wire[7:0] q_top, q_bottom, d_P;
    wire[7:0] shifted_3, shifted_2, shifted_1, shifted_0;
    wire[7:0] top, bottom, mid;
               wire    Cout_top_left,    Cout_top_right,    Cout_bottom_left,
Cout_bottom_right,
    Cout_mid_left, Cout_mid_right;

 // insert input registers to the inputs
    flopenr#4 A_reg( .clk(Clk), .reset(Reset), .en(En), .d(A), .q(q_A));
    flopenr#4 B_reg(.clk(Clk), .reset(Reset), .en(En), .d(B), .q(q_B));
// use to and for the function (a3a2a1a0)*bi
    AND and_3(.A(q_A[3:0]), .B(q_B[3]), .out(pp3[3:0]));
    AND and_2(.A(q_A[3:0]), .B(q_B[2]), .out(pp2[3:0]));
    AND and_1(.A(q_A[3:0]), .B(q_B[1]), .out(pp1[3:0]));
    AND and_0(.A(q_A[3:0]), .B(q_B[0]), .out(pp0[3:0]));

//shift every pp by +1 bit
    shift shift_3(.pp(pp3[3:0]), .index(2'b11), .out(shifted_3));
    shift shift_2(.pp(pp2[3:0]), .index(2'b10), .out(shifted_2));
    shift shift_1(.pp(pp1[3:0]), .index(2'b01), .out(shifted_1));
    shift shift_0(.pp(pp0[3:0]), .index(2'b00), .out(shifted_0));

 //add all 4 together
 //top
    CLA top_right(.A_1(shifted_0[3:0]), .B_1(shifted_1[3:0]), .Cin(1'b0),
    .sum(top[3:0]), .Cout(Cout_top_right));
              CLA    top_left(.A_1(shifted_0[7:4]),    .B_1(shifted_1[7:4]),
.Cin(Cout_top_right),
    .sum(top[7:4]), .Cout(Cout_top_left));
//bottom
            CLA   bottom_right(.A_1(shifted_2[3:0]),   .B_1(shifted_3[3:0]),
.Cin(1'b0),
    .sum(bottom[3:0]), .Cout(Cout_bottom_right));
             CLA   bottom_left(.A_1(shifted_2[7:4]),   .B_1(shifted_3[7:4]),
.Cin(Cout_bottom_right),
    .sum(bottom[7:4]), .Cout(Cout_bottom_left));
//insert stage registers
        flopenr    top_reg(  .clk(Clk),  .reset(Reset),  .en(En),  .d(top),
.q(q_top));
        flopenr  bottom_reg(.clk(Clk),  .reset(Reset),  .en(En),  .d(bottom),
.q(q_bottom));
//add top and bottom together
    CLA mid_right(.A_1(q_top[3:0]), .B_1(q_bottom[3:0]), .Cin(1'b0),
    .sum(d_P[3:0]), .Cout(Cout_mid_right));
                  CLA    mid_left(.A_1(q_top[7:4]),    .B_1(q_bottom[7:4]),
.Cin(Cout_mid_right),
    .sum(d_P[7:4]), .Cout(Cout_mid_left));
////insert output registers
    flopenr  out_reg( .clk(Clk), .reset(Reset), .en(En), .d(d_P), .q(P));

endmodule
```

flopenr.v

```
module flopenr#(parameter WIDTH = 8)
    (input  clk, reset,
     input  en,
     input  [WIDTH-1:0] d,
     output  reg[WIDTH-1:0] q
    );
    always@(posedge clk, posedge reset)
        if (reset)q<=0;
        else if(en) q<=d;
        else q<=q;
endmodule
```

digit_seperator.v

```
module digit_seperator( input [7:0] total, output reg [3:0] ones, tenths,
hundredths
    );
    integer carry;
    always @(*)
        begin
        // to find the hundredths
            if((total >199) && (total <300))
            begin
                hundredths = 2;
                carry = total-200;
            end
            else if((total >99) && (total <200))
            begin
                hundredths = 1;
                carry = total-100;
            end
            else if(total <100)
            begin
                hundredths = 0;
                carry = total;
            end

            //to find the tenths
            if((carry >89) && (carry <100))
            begin
                tenths = 9;
                ones = carry-90;
            end
            else if((carry >79) && (carry <90))
            begin
                tenths = 8;
                ones = carry-80;
            end
            else if((carry >69) && (carry <80))
            begin
                tenths = 7;
                ones = carry-70;
            end
            else if((carry >59) && (carry <70))
            begin
                tenths = 6;
                ones = carry-60;
            end
            else if((carry >49) && (carry <60))
            begin
                tenths = 5;
                ones = carry-50;
            end
            else if((carry >39) && (carry <50))
            begin
                tenths = 4;
                ones = carry-40;
            end
            else if((carry >29) && (carry <40))
            begin
```

```
                tenths = 3;
                ones = carry-30;
            end
            else if((carry >19) && (carry <30))
            begin
                tenths = 2;
                ones = carry-20;
            end
            else if((carry >9) && (carry <20))
            begin
                tenths = 1;
                ones = carry-10;
            end
            else
            begin
                tenths = 0;
                ones = carry;
            end
            //to find the ones

        end
endmodule
```

bcd_to_7seg.v

```
module bcd_to_7seg (
        input  wire [3:0] BCD,
        output reg  [7:0] s
    );

    always @ (BCD) begin
        case (BCD)
                4'd0: s = 8'b11000000;
                4'd1: s = 8'b11111001;
                4'd2: s = 8'b10100100;
                4'd3: s = 8'b10110000;
                4'd4: s = 8'b10011001;
                4'd5: s = 8'b10010010;
                4'd6: s = 8'b10000010;
                4'd7: s = 8'b11111000;
                4'd8: s = 8'b10000000;
                4'd9: s = 8'b10010000;
             default: s = 8'b01111111;
        endcase
    end

endmodule
```

led_mux.v

```
module led_mux (
        input  wire       clk,
        input  wire       rst,
        input  wire [7:0] LED3,
        input  wire [7:0] LED2,
        input  wire [7:0] LED1,
        input  wire [7:0] LED0,
        output wire [3:0] LEDSEL,
        output wire [7:0] LEDOUT
    );

    reg [1:0]  index;
    reg [11:0] led_ctrl;

    assign {LEDSEL, LEDOUT} = led_ctrl;

    always @ (posedge clk) index <= (rst) ? 2'b0 : (index + 2'd1);

    always @ (index, LED0, LED1, LED2, LED3) begin
        case (index)
               2'd0: led_ctrl <= {4'b1110, LED0};
               2'd1: led_ctrl <= {4'b1101, LED1};
               2'd2: led_ctrl <= {4'b1011, LED2};
               2'd3: led_ctrl <= {4'b0111, LED3};
            default: led_ctrl <= {4'b1111, 8'hFF};
        endcase
    end

endmodule
```

multiplier_constrain.xdc

```
# Clock signal
set_property -dict {PACKAGE_PIN W5 IOSTANDARD LVCMOS33} [get_ports
{clk100mHz}];
create_clock -add -name sys_clk_pin -period 10.00 -waveform {0 5} [get_ports
{clk100mHz}];

# input switches
#A
set_property -dict {PACKAGE_PIN V17 IOSTANDARD LVCMOS33} [get_ports {A[0]}];
set_property -dict {PACKAGE_PIN V16 IOSTANDARD LVCMOS33} [get_ports {A[1]}];
set_property -dict {PACKAGE_PIN W16 IOSTANDARD LVCMOS33} [get_ports {A[2]}];
set_property -dict {PACKAGE_PIN W17 IOSTANDARD LVCMOS33} [get_ports {A[3]}];
#B
set_property -dict {PACKAGE_PIN W2 IOSTANDARD LVCMOS33} [get_ports {B[0]}];
set_property -dict {PACKAGE_PIN U1 IOSTANDARD LVCMOS33} [get_ports {B[1]}];
set_property -dict {PACKAGE_PIN T1 IOSTANDARD LVCMOS33} [get_ports {B[2]}];
set_property -dict {PACKAGE_PIN R2 IOSTANDARD LVCMOS33} [get_ports {B[3]}];

# input LED
#A
set_property -dict {PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports {A_out[0]}];
set_property -dict {PACKAGE_PIN E19 IOSTANDARD LVCMOS33} [get_ports {A_out[1]}];
set_property -dict {PACKAGE_PIN U19 IOSTANDARD LVCMOS33} [get_ports {A_out[2]}];
set_property -dict {PACKAGE_PIN V19 IOSTANDARD LVCMOS33} [get_ports {A_out[3]}];
#B
set_property -dict {PACKAGE_PIN P3 IOSTANDARD LVCMOS33} [get_ports {B_out[0]}];
set_property -dict {PACKAGE_PIN N3 IOSTANDARD LVCMOS33} [get_ports {B_out[1]}];
set_property -dict {PACKAGE_PIN P1 IOSTANDARD LVCMOS33} [get_ports {B_out[2]}];
set_property -dict {PACKAGE_PIN L1 IOSTANDARD LVCMOS33} [get_ports {B_out[3]}];
#rst
set_property -dict {PACKAGE_PIN V13 IOSTANDARD LVCMOS33} [get_ports {rst_out}];
#en
set_property -dict {PACKAGE_PIN V14 IOSTANDARD LVCMOS33} [get_ports {en_out}];

#clock button
set_property -dict {PACKAGE_PIN T17 IOSTANDARD LVCMOS33} [get_ports {clock}];

#reset s?itch
#set_property -dict {PACKAGE_PIN U18 IOSTANDARD LVCMOS33} [get_ports {rst}]; # Center
Button
set_property -dict {PACKAGE_PIN V2 IOSTANDARD LVCMOS33} [get_ports {rst}];
set_property -dict {PACKAGE_PIN W13 IOSTANDARD LVCMOS33} [get_ports {en}];

#LED selection
set_property -dict {PACKAGE_PIN U2 IOSTANDARD LVCMOS33} [get_ports {LEDSEL[0]}]; # AN0
set_property -dict {PACKAGE_PIN U4 IOSTANDARD LVCMOS33} [get_ports {LEDSEL[1]}]; # AN1
set_property -dict {PACKAGE_PIN V4 IOSTANDARD LVCMOS33} [get_ports {LEDSEL[2]}]; # AN2
set_property -dict {PACKAGE_PIN W4 IOSTANDARD LVCMOS33} [get_ports {LEDSEL[3]}]; # AN3

#LED output
set_property -dict {PACKAGE_PIN W7 IOSTANDARD LVCMOS33} [get_ports {LEDOUT[0]}]; # CA
set_property -dict {PACKAGE_PIN W6 IOSTANDARD LVCMOS33} [get_ports {LEDOUT[1]}]; # CB
set_property -dict {PACKAGE_PIN U8 IOSTANDARD LVCMOS33} [get_ports {LEDOUT[2]}]; # CC
set_property -dict {PACKAGE_PIN V8 IOSTANDARD LVCMOS33} [get_ports {LEDOUT[3]}]; # CD
set_property -dict {PACKAGE_PIN U5 IOSTANDARD LVCMOS33} [get_ports {LEDOUT[4]}]; # CE
set_property -dict {PACKAGE_PIN V5 IOSTANDARD LVCMOS33} [get_ports {LEDOUT[5]}]; # CF
set_property -dict {PACKAGE_PIN U7 IOSTANDARD LVCMOS33} [get_ports {LEDOUT[6]}]; # CG
set_property -dict {PACKAGE_PIN V7 IOSTANDARD LVCMOS33} [get_ports {LEDOUT[7]}]; # DP
```

unsigned_integer_multiplier_tb.v

```
module unsigned_integer_multiplier_tb;
    reg [3:0] A_tb;
    reg [3:0] B_tb;
    reg clk_tb;
    reg reset_tb;
    reg en_tb;
    wire [7:0] P_tb;
    integer i,j, errors;
    reg[7:0] P_expected;
          unsigned_integer_multiplier  DUT(.A(A_tb),  .B(B_tb),  .P(P_tb),
.Clk(clk_tb), .Reset(reset_tb), .En(en_tb));

    initial begin
        errors = 0;
        en_tb =1;
        reset_tb = 1;// start with new input
        clk_tb = 0;
        #5
        reset_tb = 0;
        for(i=0; i< 16; i=i+1)
        begin
            A_tb = i;
            for(j=0; j< 16; j=j+1)
            begin
                B_tb = j;

                clk_tb = !clk_tb;#5
                clk_tb = !clk_tb;#5
                clk_tb = !clk_tb;#5
                clk_tb = !clk_tb;#5
                clk_tb = !clk_tb;#5
                clk_tb = !clk_tb;
                P_expected = A_tb*B_tb;
                #5;
                if(P_tb != P_expected)
                begin
                     $display("Failed when A=%b, B=%b, the P_expected is %d,
but the actual value P is %d",
                                A_tb, B_tb, P_expected, P_tb);
                    errors = errors+1;
                end

            end
        end
        if(!errors) $display("Test finished with %0d errors", errors);
    #10 $finish;
    end
endmodule
```