

San Jose State University  
Department of Computer Engineering

CMPE 125 Lab Report

Lab 3 Report

Title Combinational Building Blocks

Semester Fall 2019

Date 9/23/2019

by

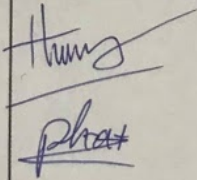
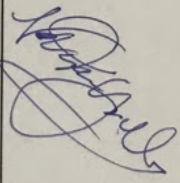
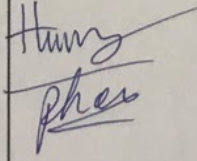
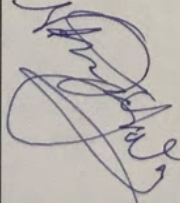
Name Hung Nguyen

SID 012392081

Name Phat Le

SID 012067666

Lab Checkup Record

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

\* Detailed descriptions must be given in the report.

## Introduction

The purpose of this lab is to get familiar with combinational building blocks and design, verification, validation flow, and the EDA tool. To make everything easier, the lab separated into 3 parts; 8-3 bit priority encoder, 4-bits right shifter/rotator, and 4 bit ALU.

## Design Methodology

The first task of the lab is to design an 8-3 bit priority encoder. The requirement for this design, we need to re-write the Verilog design code using *casez*, *if statement* and *for* loops; including *tb\_self\_checking*. Table 1 shows the truth table for 8-3 bit priority. Table 2 shows the I/O needed for the design to test all the case with Basys 3 FPGA board

Table 1: Truth Table for 8-3 bit

A[7]	A[6]	A[5]	A[4]	A[3]	A[2]	A[1]	A[0]	Y[2:0]	Valid
1	x	x	x	x	x	x	x	111	1
0	1	x	x	x	x	x	x	110	1
0	0	1	x	x	x	x	x	101	1
0	0	0	1	x	x	x	x	100	1
0	0	0	0	1	x	x	x	011	1
0	0	0	0	0	1	x	x	010	1
0	0	0	0	0	0	1	x	001	1
0	0	0	0	0	0	0	1	000	1
0	0	0	0	0	0	0	0	xxx	0

Table 2: I/O required for 8-3 design

Signal	Width	Type
A	A_width	Input
B	B_width	Output
valid	1	Output check for inputs value

*Table 3: List of modules and files needed for Task 1*

Module	Function
enc_case.v	This module will give 3-bit output 1-bit output. The 1-bit output will be a valid bit which is used to check the other 3-bit is match or not. This module is used "case" statement in priority encoder .
enc_if.v	This module will give 3-bit output 1-bit output. The 1-bit output will be a valid bit which is used to check the other 3-bit is match or not. This module is used "if" statement in priority encoder .
enc_for.v	This module will give 3-bit output 1-bit output. The 1-bit output will be a valid bit which is used to check the other 3-bit is match or not. This module is used "for" loop in priority encoder .
tb_encoder.v	This module includes all the test case for all "case, if, for loop" implement for priority encoder . This module will test all the possibility of priority encoder (I/O).
Priority_encoder_constrain.xdc	This module is set all the LED, I/O for priority encoder in Basys3 FPGA board.

For the second task, we need to design the 4-bit right shifter/rotator. The requirement for this task, we need to follow the given functional table (Table 3). After that, write a self\_testing Verilog test bench and validate the design by using Basys 3 FPGA board. Students can use a given source code on the lecture for a guideline.

*Table 4: 4-bit right shifter/rotator functional table*

<b>Ctrl [2:0]</b>	<b>Operation</b>	<b>Input</b>	<b>Output</b>
000	Pass	a b c d	a b c d
001	Shift right 1 bit	a b c d	0 a b c
010	Shift right 2 bit	a b c d	0 0 a b
011	Shift right 3 bit	a b c d	0 0 0 a
100	Shift right 4 bit	a b c d	0 0 0 0
101	Rotate right 1 bit	a b c d	d a b c
110	Rotate right 2 bit	a b c d	c d a b
111	Rotate right 3 bit	a b c d	b c d a

*Table 5: List of modules and files used for Task 2*

<b>Module</b>	<b>Function</b>
shift_rotator.v	This module will take 2 input signal and return one 4-bit output. The input signal will include one 3-bit control and one 4-bit input
tb_shift_rotator.v	This testbench will perform 4-bit right shifter/rotator and it should check all the possibility
shift_rotator.xdc	This module is set the LED, I/O and control for 4-bit shifter/rotator on Basys3 FPGA board.

For the last part, we need to design given 4 bit ALU source code on canvas, but enhance it with 2 more output functions overflow flag and zero flags. After that, write a self-testing Verilog testbench and functionally verify the ALU. Validate the output with Basys 3 FPGA board and make sure to use individual LED on board to display the output.

Table 6: ALU Function Table

Ctrl	Output
000	A & B
001	A  B
010	A+B
110	A-B
111	SLT

Table 7 : List of modules and files needed for Task 3

Module	Function
ALU.v	This module requires 3 input signal and return 3 output signal. The input signals will include 2 4-bit input (A,B) and 3-bit control signal. The output will include 3-bit result, 1 overflow and 1 zero.
tb_ALU.v	This is self tb_ALU.v and it should include all the function that can test all the possibilities .
ALU.xdc	This module is set the LED, I/O and control for 4-bit ALU on Basys3 FPGA board.

## Simulation Result

### Task 1

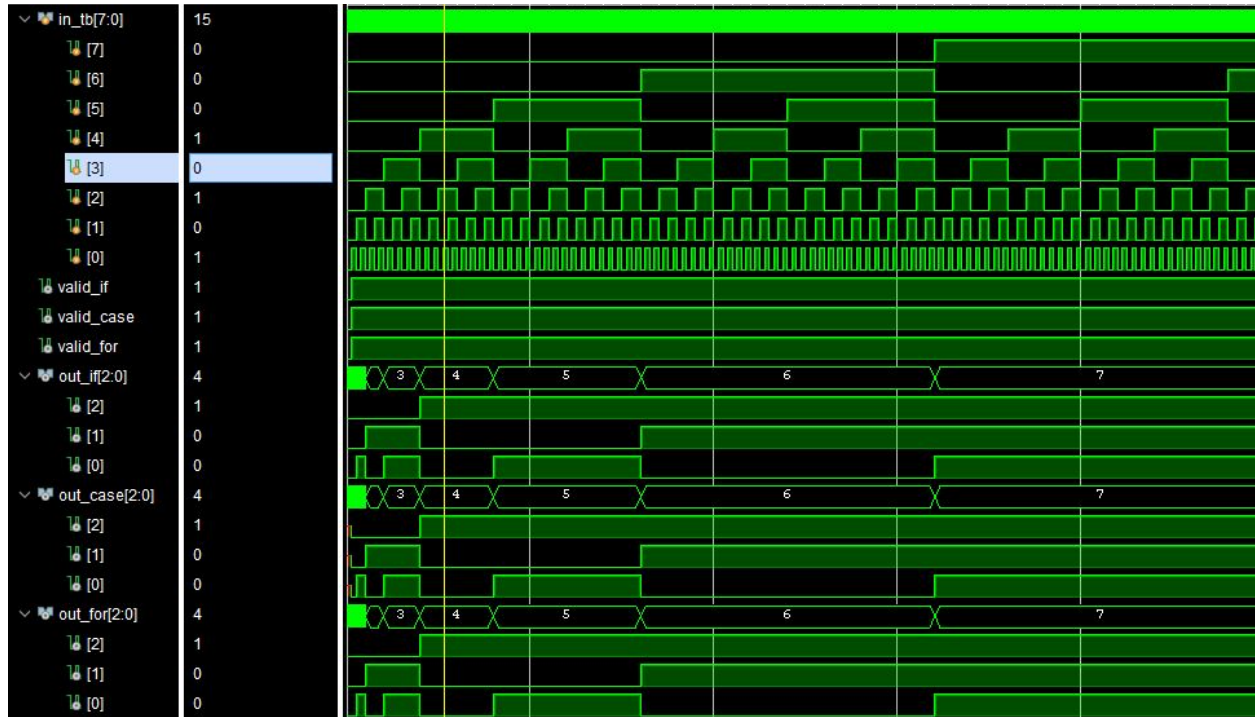


Figure 1: Simulation waveforms produced from Task 1

For this task, we try to test all the cases from 0 to 16 (`in_tb`). According to *Figure 1* above, the cursor is at the input (`in_tb`) of 00010101 and the output for all three modules if (`out_if`), casez (`out_case`), and for (`out_for`) are produced the same output of 100. This is correct because the 4th bit of the input is 1 so the output is also 4 (100). Furthermore, other input cases also correct. For this reason, we can conclude that the simulation process for Task 1 has been succeeded as all the outputs for casez, if, and for match the expected outcomes.

## Task 2

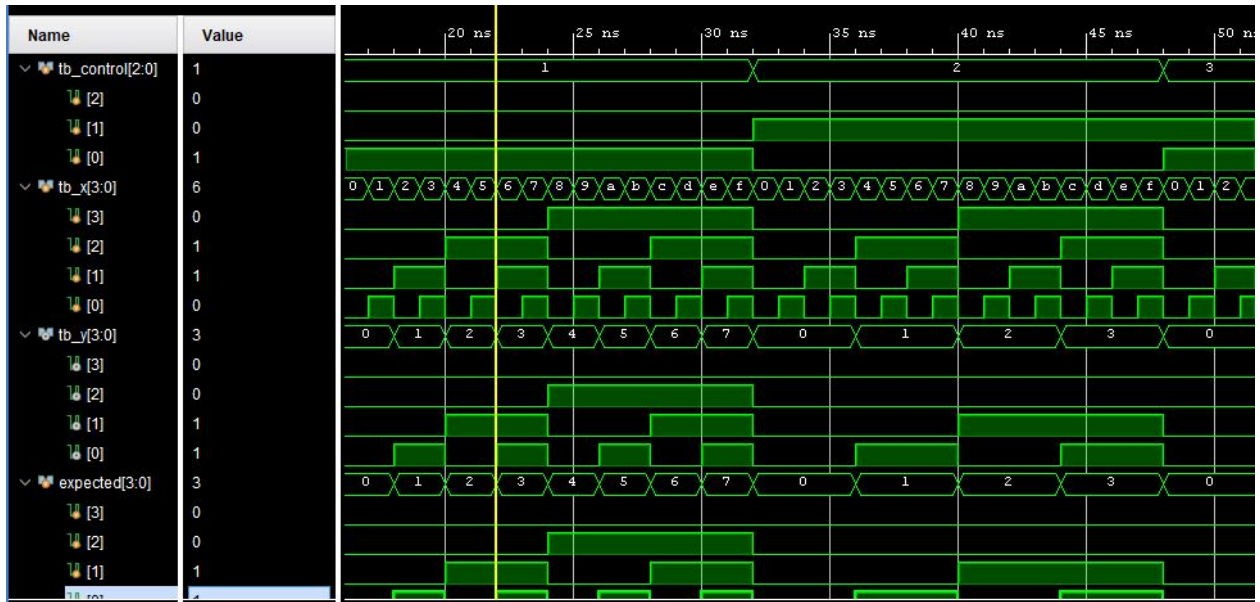


Figure 2: Simulation waveforms produced from Task 2

From Figure 2, we can observe that the output (`tb_y`) produces the same waveform as the expected result (`expected`) created by the testbench file. Specifically, in this case, the input (`tb_x`) is 0110 and control (`tb_control`) is 001. This gives the output to be 0011 since it shift right by 1 bit. Thus, the simulation process for Task 2 has been succeeded as the outcome is as expected.

## Task 3

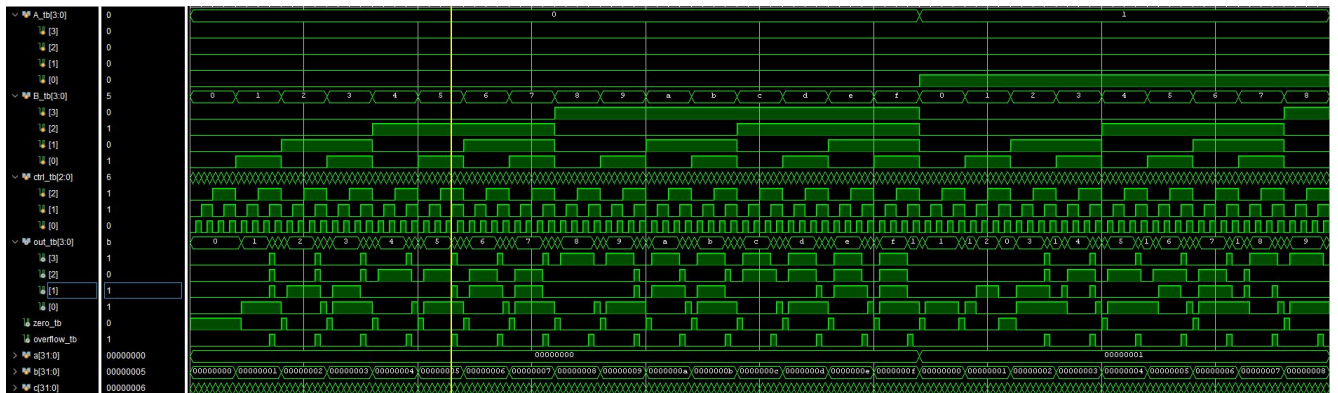


Figure 3: Simulation waveforms produced from Task 3

For Task 3 simulation process, we try to observe the output when  $A - B = 0 - 5 = -5$ . According to Figure 3, the outcome is as expected because when `A(A_tb)` is 0000, `B(B_tb)` is 0101, control (`ctrl_tb`) is 110, it gives the output (`out_tb`) to be 1011 and also the overflow signal (`overflow_tb`) is 1. In addition, we also randomly tested other input cases and the result matches the expected outcome. Thus, the simulation process for this task has been succeed.



## FPGA Validation Task 1

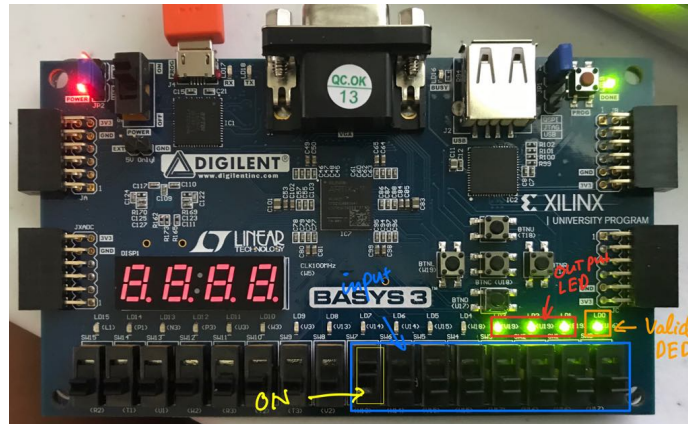


Figure 4: Photo of operation being (insert operation here is 10000000) tested on FPGA board.

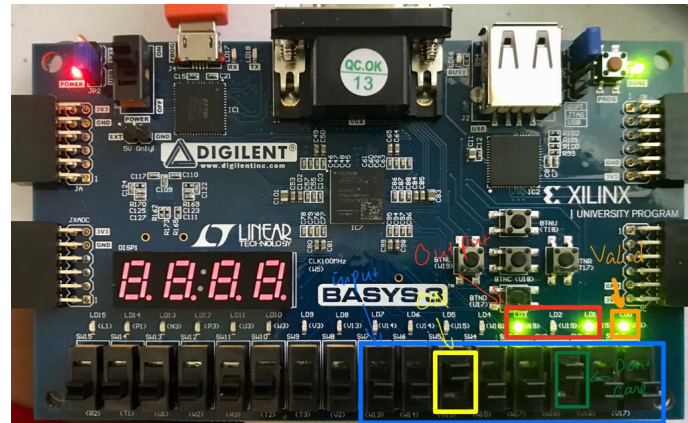


Figure 5: Photo of operation being (insert operation here is 00100100) tested on FPGA board.

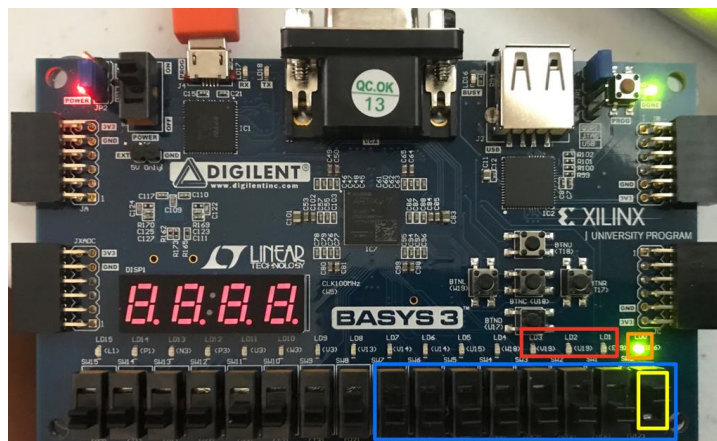
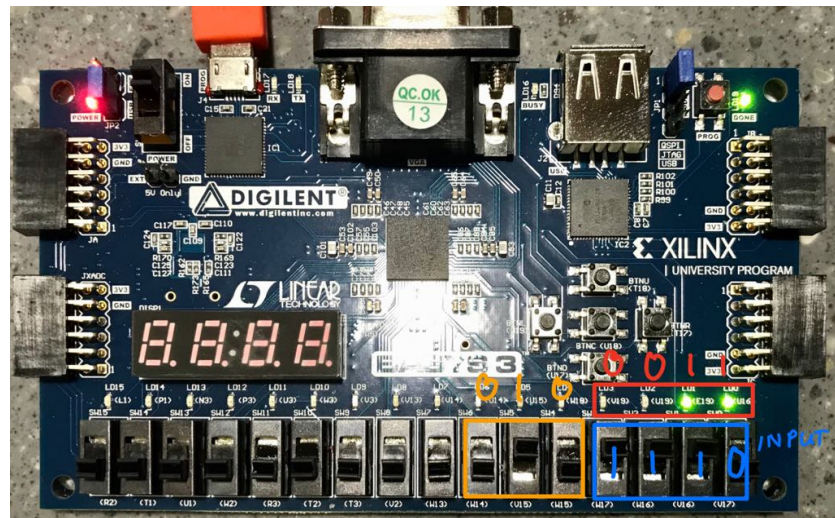


Figure 6: Photo of operation being (insert operation here is 00000001) tested on FPGA board.



For task 1, the 8-bit inputs are the first 8 switches (blue), the 3-bit outputs are 3 LEDs (red), and 1 testing valid output as an LED (orange). According to *Figure 4*, when the 8th switch is on (10000000), the output LED is 111 and the valid LED is 1. Furthermore, *Figure 5* shows when the input is 00100100, which mean the 6th and 3rd switches is on, the output is 101. When the 3rd switch is off, the output is still 101. This is correct because the design code only cares about the active level of the most significant bit. In this case, the MSB is the 6th bit. *Figure 6* shows when only the first switch is on, the output is 000 and it is still valid since valid LED is on. In conclusion, the FPGA validation has been succeeded as the outputs match the expected results described in *Table 1*.

## Task 2



*Figure 7: Photo of operation being (input is 1110 and ctrl is 010) tested on FPGA board.*

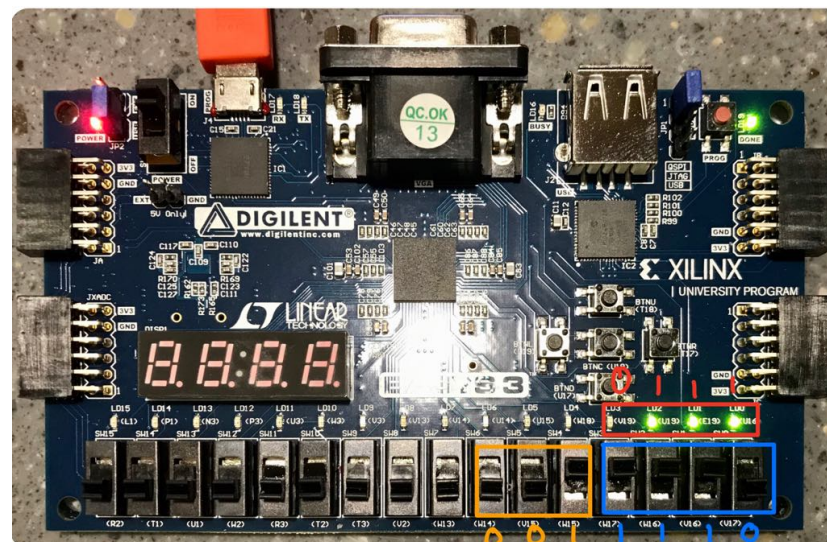


Figure 8: Photo of operation being (input is 1110 and ctrl is 001) tested on FPGA board.

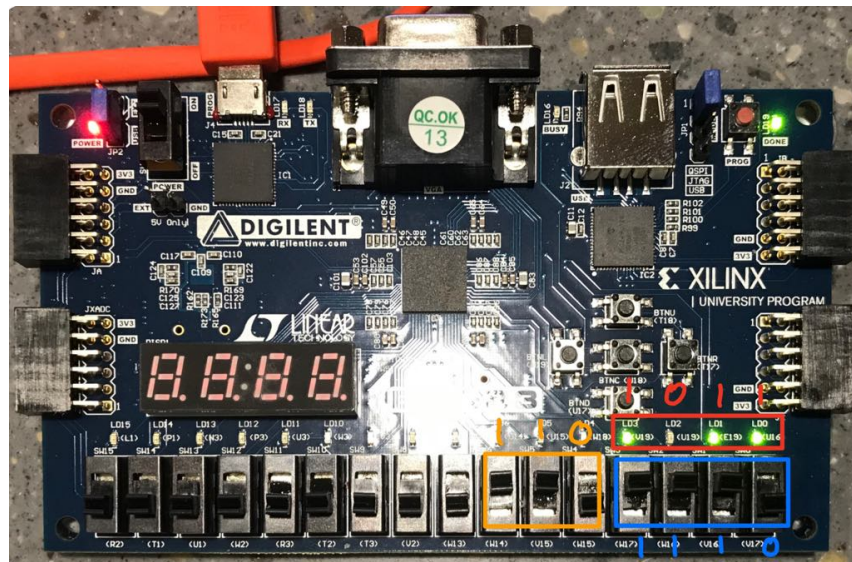
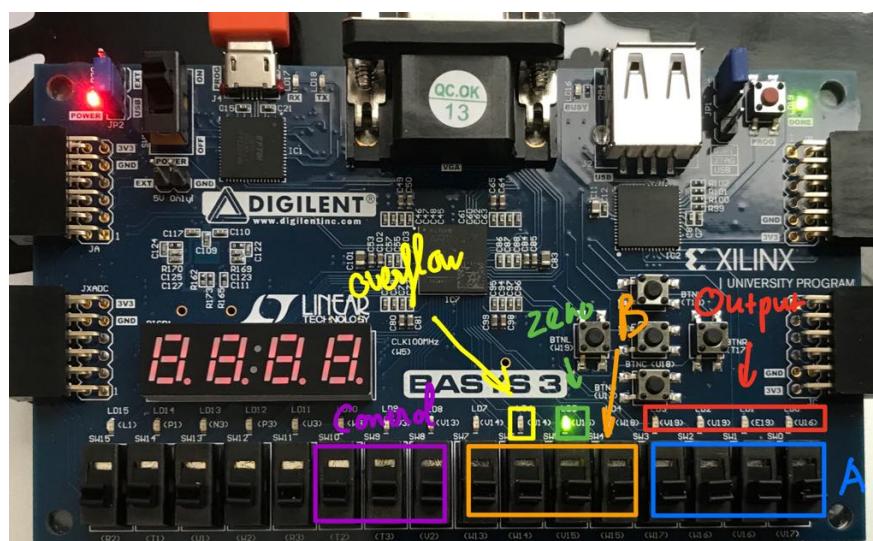


Figure 9: Photo of operation being (input is 1110 and ctrl is 110) tested on FPGA board.

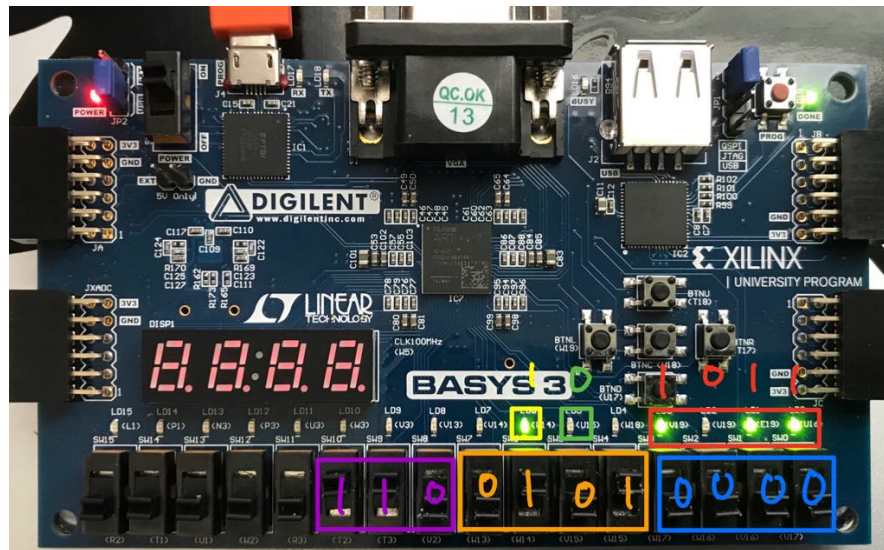
According to Figure 7, the control operation is 010 which is shift right 2 bit (Table 3). In this case, the input is 1110 and the output is 0011 which matches the expected result. For Figure 8, the control operation is 010 which shifts the input to the right by 1 bit. The result is as expected since the input is 1110 and the LED output is 0111. Figure 9 also shows the correct result of 2 bit rotation operation (110) as 1110 input becomes 1011. As a result, Task 2 FPGA validation has been successfully completed.

### Task 3





*Figure 10: Photo of operation being (input A is 0000, B is 0000, and Control is 0000) tested on FPGA board.*



*Figure 11: Photo of operation being (input A is 0000, B is 0101, and Control is 1100) tested on FPGA board.*

In this Task, the two input A(Blue) and B(Orange) will be used by the operation from Control signal (Purple). In *Figure 10*, the output(Red) is 0000 and the zero flag (Green) is 1 because input A and B is 0000 and performing AND operation (000). For *Figure 11*, A is 0(0000) , B is 5(0101), and perform subtraction operation (110). We can observe that the overflow flag(Yellow) is on because the output is -5 in the form of 2's complement of 1011. Ultimately, the FPGA validation process has been succeeded.

## Conclusion

Overall, both the simulation and FPGA validation has successfully completed since the outcome is as expected. For the simulation process, we try to test every input for every task. We realize that it is hard to prove all of the cases as the number of inputs is big. For this reason, we will try to generate random inputs in the testbench file to be more effective in the next lab. One thing we learn from this lab is that the value is in 2's complement format. Specifically, when we try to test overflow result in Task 3, 0 minus 5 is equal to -5 which will make the overflow flag to turn on and the value is 1011.

## Appendix

### a. Block Diagram

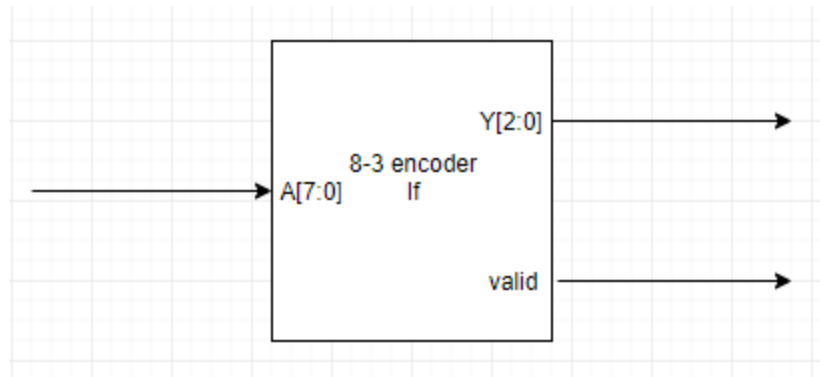


Figure 12: 8-3 bit priority encoder if function

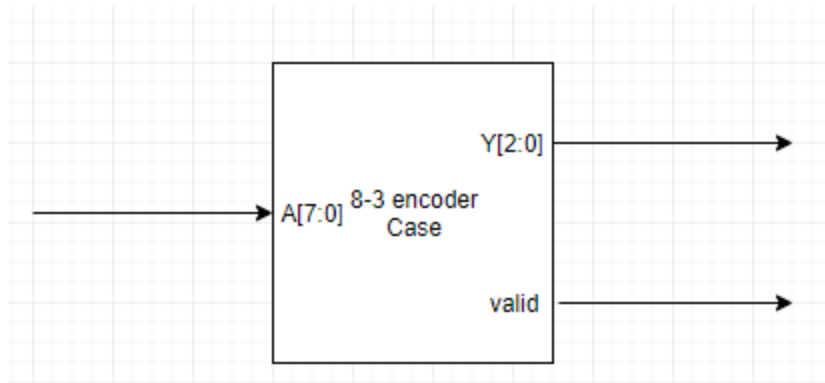


Figure 13: 8-3 bit priority encoder casez function

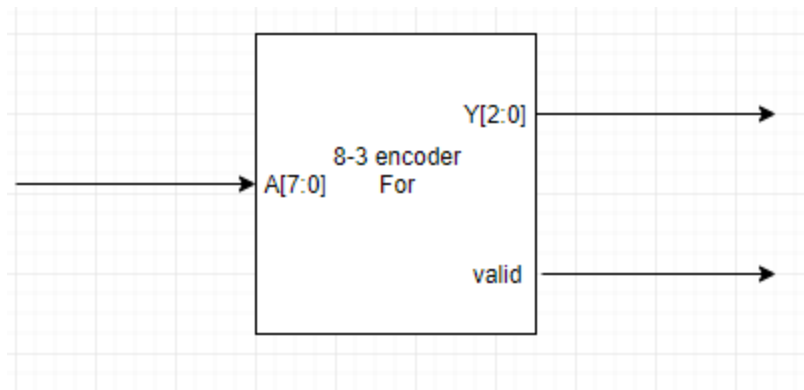


Figure 14: 8-3 bit priority encoder for loop

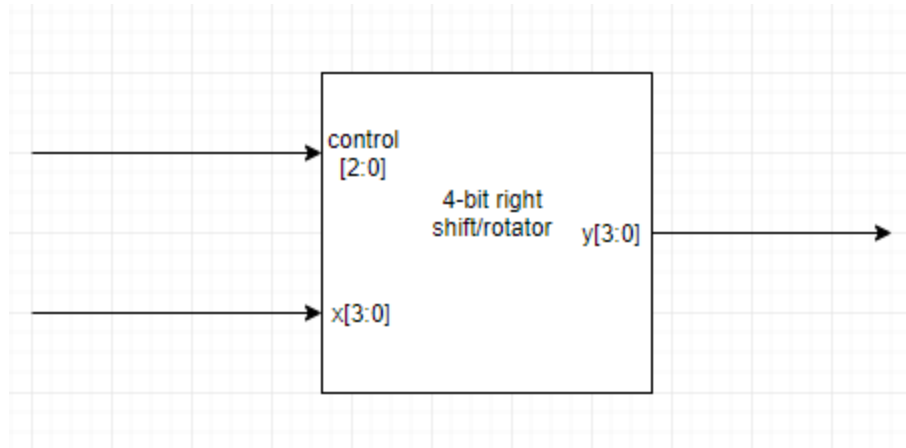


Figure 15: 4 bit right shifter/rotator

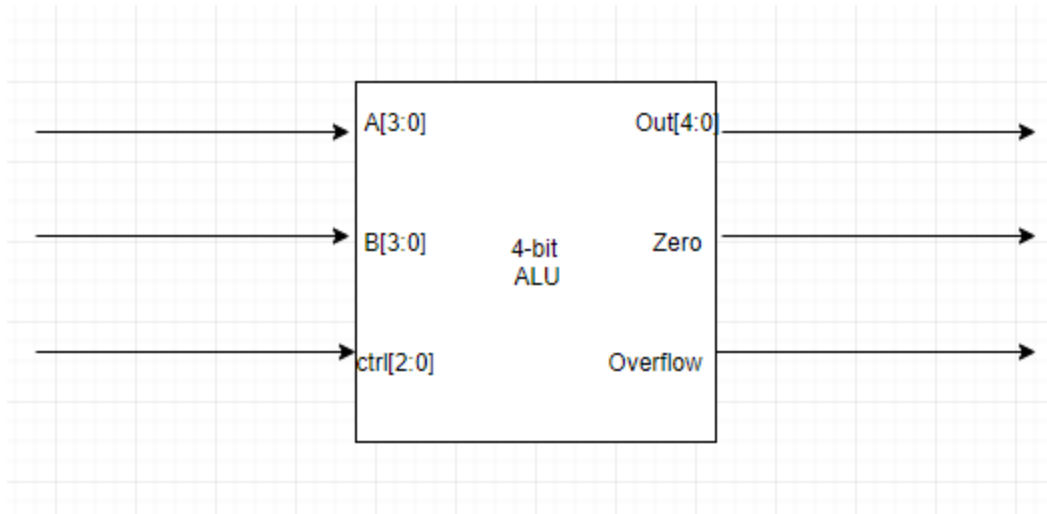


Figure 16: 4-bit ALU

## b. Source Code

### Task 1

#### enc\_if.v

```
module enc_if (input [7:0] A, output reg valid, reg[2:0] Y);
    always @ (A)
    begin
        valid=1;
        if (A[7])Y = 7;
        else if (A[6]) Y = 6;
        else if (A[5]) Y = 5;
        else if (A[4]) Y = 4;
        else if (A[3]) Y = 3;
        else if (A[2]) Y = 2;
        else if (A[1]) Y = 1;
        else if (A[0]) Y = 0;
        else begin
            valid=0;
            Y = 3'b000;
        end
    end
end
endmodule
```

#### enc\_case.v

```
module enc_case
(input [7:0] A, output reg valid, reg[2:0] Y);
    always @ (A)
begin
    valid = 1;
    casez (A)
        8'b1???_????: Y = 7;
        8'b01??_????: Y = 6;
        8'b001?_????: Y = 5;
        8'b0001_????: Y = 4;
        8'b0000_1??? : Y = 3;
        8'b0000_01?? : Y = 2;
        8'b0000_001? : Y = 1;
        8'b0000_0001: Y = 0;
        default: begin
            valid = 0;
            Y = 3'bxxx;
        end
    endcase
end
endmodule
```



## enc\_for.v

```
module enc_for #(parameter iwidth = 8, owidth = 3)
(input [iwidth-1:0] A, output reg valid, reg[owidth-1:0] Y);
    integer n;
    always @ (A)
    begin
        valid = 0;
        Y= 3'b000;
        for (n = 0; n < iwidth; n = n + 1)
        begin
            if (A[n])
            begin
                valid = 1;
                Y = n;
            end
        end
    end
end
endmodule
```

## tb\_encoder.v

```
module tb_encoder;
    reg [7:0] in_tb;
    wire      valid_if, valid_case, valid_for;
    wire [2:0] out_if, out_case, out_for;
    integer n;
    enc_if      DUT_IF (in_tb, valid_if, out_if);
    enc_case    DUT_CASE (in_tb, valid_case, out_case);
    enc_for     #(8,3) DUT_FOR (in_tb, valid_for, out_for);
    initial begin
        for (n = 0; n < 256; n = n + 1)
        begin
            in_tb = n; #5;
            if (!in_tb[out_if])      $display ("Encoder If Failed");
            if (!in_tb[out_case])    $display ("Encoder Case Failed");
            if (!in_tb[out_for])     $display ("Encoder For Failed");
            if (in_tb && !valid_if)   $display ("Valid If Failed");
            if (in_tb && !valid_case) $display ("Valid Case Failed");
            if (in_tb && !valid_for)  $display ("Valid For Failed");
        end
        $display ("Simulation Finished");
        $finish;
    end
endmodule
```

## priority\_encoder\_constains.xdc

```
#Set inputs
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]}};
set_property -dict {PACKAGE_PIN W15 IOSTANDARD LVCMOS33} [get_ports {A[4]}};
set_property -dict {PACKAGE_PIN V15 IOSTANDARD LVCMOS33} [get_ports {A[5]}};
set_property -dict {PACKAGE_PIN W14 IOSTANDARD LVCMOS33} [get_ports {A[6]}};
set_property -dict {PACKAGE_PIN W13 IOSTANDARD LVCMOS33} [get_ports {A[7]}};
#Set outputs
set_property -dict {PACKAGE_PIN E19 IOSTANDARD LVCMOS33} [get_ports {Y[0]}};
set_property -dict {PACKAGE_PIN U19 IOSTANDARD LVCMOS33} [get_ports {Y[1]}};
set_property -dict {PACKAGE_PIN V19 IOSTANDARD LVCMOS33} [get_ports {Y[2]}};
#Valid button
set_property -dict {PACKAGE_PIN U16 IOSTANDARD LVCMOS33} [get_ports {valid}};
```

## Task 2:

## shift\_rotator.v

```
module shift_rotator(
    input wire[2:0] control,
    input wire[3:0] x,
    output reg [3:0] y
);
    always @(control,x)
    begin
        case(control)
            3'b000: y=x;
            3'b001: y={1'b0,x[3:1]};
            3'b010: y={2'b00, x[3:2]};
            3'b011: y={3'b000,x[3:3]};
            3'b100: y=4'b0000;
            3'b101: y={x[0],x[3:1]};
            3'b110: y={x[1:0],x[3:2]};
            3'b111: y={x[2:0],x[3]};
            default: y=4'b0000;
        endcase
    end
endmodule
```

## tb\_shift\_rotator.v

```
module tb_shift_rotator;
    reg [2:0] tb_control;
    reg [3:0] tb_x;
    wire [3:0] tb_y;
    reg [3:0] expected;

    shift_rotator DUT(.control(tb_control),.x(tb_x),.y(tb_y));
    integer i,j;
    initial
    begin

        for(i=0;i<8;i=i+1)//control bit
        begin
            for(j=0;j<16;j=j+1)//4 bit input
            begin
                tb_control=i;
                tb_x=j;
                case (tb_control)
                    3'b000: expected = tb_x;
                    3'b001: expected = {1'b0, tb_x[3],tb_x[2],tb_x[1]};
                    3'b010: expected = {2'b00, tb_x[3],tb_x[2]};
                    3'b011: expected = {3'b000, tb_x[3]};
                    3'b100: expected = 4'b0000;
                    3'b101: expected = {tb_x[0], tb_x[3],tb_x[2],tb_x[1]};
                    3'b110: expected = {tb_x[1], tb_x[0], tb_x[3], tb_x[2]};
                    3'b111: expected = {tb_x[2], tb_x[1], tb_x[0], tb_x[3]};
                    default: expected = 4'b0000;
                endcase

                // Wait for inputs to propagate to output
                #1;

                // Check if output matches expected result
                if(tb_y != expected)
                begin
                    // Mismatch, print error message
                    $display("Error, output is %d, expected %d", tb_y, expected);
                    // Stop simulation on the first error we found
                    $stop;
                end

            end // End Assign loop x counts to inputs

        end // End Assign loop control counts to inputs

        // No errors, print success message
        $display("No Errors. Test was successful.");
        $stop;
    end
endmodule
```

## shift\_rotator.xdc

```
# Input signals
set_property -dict { PACKAGE_PIN V17 IOSTANDARD LVCMOS33 } [get_ports { x[0] }];
set_property -dict { PACKAGE_PIN V16 IOSTANDARD LVCMOS33 } [get_ports { x[1] }];
set_property -dict { PACKAGE_PIN W16 IOSTANDARD LVCMOS33 } [get_ports { x[2] }];
set_property -dict { PACKAGE_PIN W17 IOSTANDARD LVCMOS33 } [get_ports { x[3] }];

#control signals
set_property -dict { PACKAGE_PIN W15 IOSTANDARD LVCMOS33 } [get_ports { control[0] }];
set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { control[1] }];
set_property -dict { PACKAGE_PIN W14 IOSTANDARD LVCMOS33 } [get_ports { control[2] }];

# LEDs
set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { y[0] }];
set_property -dict { PACKAGE_PIN E19 IOSTANDARD LVCMOS33 } [get_ports { y[1] }];
set_property -dict { PACKAGE_PIN U19 IOSTANDARD LVCMOS33 } [get_ports { y[2] }];
set_property -dict { PACKAGE_PIN V19 IOSTANDARD LVCMOS33 } [get_ports { y[3] }];
```

### Task 3:

## ALU.v

```
module ALU( input wire[3:0]      A,
            input wire[3:0]      B,
            input wire[2:0]      ctrl,
            output reg[3:0]      out,
            output                zero,
            output reg           overflow
            );

always@(*)
    case(ctrl)
        3'b000: begin overflow = 0; out = A & B; end    //bitwise AND
        3'b001: begin overflow = 0; out = A | B; end    //bitwise OR
        3'b010: {overflow, out} = A + B;    //addition
        3'b110: {overflow, out} = A - B;    //subtraction
        3'b111: begin overflow = 0; out = (A < B)? 1:0; end    //SLT
    endcase

    //assign overflow = (out[4] == 5'b1);
    assign zero = (out == 4'b0);

endmodule
```

## tb\_ALU.v

```
module tb_ALU;

reg[3:0] A_tb;
reg[3:0] B_tb;
reg[2:0] ctrl_tb;
wire[3:0] out_tb;
wire zero_tb;
wire overflow_tb;

ALU DUT(.A(A_tb), .B(B_tb), .ctrl(ctrl_tb), .out(out_tb), .zero(zero_tb),
.overflow(overflow_tb));

integer a;
integer b;
integer c;

initial
begin

    $display("Testbench started.");

    for (a = 0; a < 16; a = a + 1)//input a
    begin
        A_tb = a;
        for (b = 0; b < 16; b = b + 1)//input b
        begin
            B_tb = b;
            for (c = 0; c < 8; c = c + 1)//control
            begin
                ctrl_tb = c; #5
                if (c == 0 && out_tb != (A_tb & B_tb))                $display("Control
signal 000: error at A = %0d, B = %0d", a, b);
                else if (c == 1 && out_tb != (A_tb | B_tb))          $display("Control
signal 001: error at A = %0d, B = %0d", a, b);
                else if (c == 2 && out_tb != (A_tb + B_tb))          $display("Control
signal 010: error at A = %0d, B = %0d", a, b);
                else if (c == 6 && out_tb != (A_tb - B_tb))          $display("Control
signal 110: error at A = %0d, B = %0d", a, b);
                else if (c == 7 && out_tb != (A_tb < B_tb)?1:0)      $display("Control
signal 111: error at A = %0d, B = %0d", a, b);
            end
        end
    end

    $display("Test was successful.");
    #10 $finish;

end
endmodule
```

## ALU.xdc

### #Switches

```
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] }];
set_property -dict { PACKAGE_PIN W15 IOSTANDARD LVCMOS33 } [get_ports { B[0] }];
set_property -dict { PACKAGE_PIN V15 IOSTANDARD LVCMOS33 } [get_ports { B[1] }];
set_property -dict { PACKAGE_PIN W14 IOSTANDARD LVCMOS33 } [get_ports { B[2] }];
set_property -dict { PACKAGE_PIN W13 IOSTANDARD LVCMOS33 } [get_ports { B[3] }];
```

### #control

```
set_property -dict { PACKAGE_PIN V2 IOSTANDARD LVCMOS33 } [get_ports { ctrl[0] }];
set_property -dict { PACKAGE_PIN T3 IOSTANDARD LVCMOS33 } [get_ports { ctrl[1] }];
set_property -dict { PACKAGE_PIN T2 IOSTANDARD LVCMOS33 } [get_ports { ctrl[2] }];
```

### # LEDs

```
set_property -dict { PACKAGE_PIN U16 IOSTANDARD LVCMOS33 } [get_ports { out[0] }];
set_property -dict { PACKAGE_PIN E19 IOSTANDARD LVCMOS33 } [get_ports { out[1] }];
set_property -dict { PACKAGE_PIN U19 IOSTANDARD LVCMOS33 } [get_ports { out[2] }];
set_property -dict { PACKAGE_PIN V19 IOSTANDARD LVCMOS33 } [get_ports { out[3] }];
set_property -dict { PACKAGE_PIN U15 IOSTANDARD LVCMOS33 } [get_ports { zero }];
set_property -dict { PACKAGE_PIN U14 IOSTANDARD LVCMOS33 } [get_ports { overflow }];
```