

Digital Systems Design

Engr 378

Lab 4

Downloading into A FPGA

Date:

4/6/21

Grade:

By:

Aaron Luong and Richard Couto

Problem Analysis

In this lab, we set out to design a Multi-Code Converter as well as install it onto an FPGA. This Multi-code converter will be able to take in a 4-bit binary number and output either binary, hexadecimal, or decimal numbers. We will show this on three digital displays, two for the resulting digits themselves and one to show the form of conversion

Hardware Design

We began with a general circuit outline, leaving the conversion sections as named modules that received our 4-bit binary number as an input, and fed out the 7-bit lines required to drive the digital display. Each module delivers this to a mux before the display. The mux is influenced by our three buttons, thus selecting the desired output and displaying only the appropriate conversion.

We coded the conversion by tying every input to the desired display option for every input scenario. Due to the display utilizing negative logic, we needed to invert our original pre-lab designs. Our pre-labs include the table we used to convert both the decimal and hexadecimal. The binary form was sent to both displays. However, it was reduced by 10 on its path to the second display to ensure correct representation

Verilog Modeling

For the lab, we will be making a device that has three buttons with negative logic to select the type of output and four switches to represent the input number. The first thing that is needed is the converter module.

Converter:

```
module Converter(in, out);
    input [3:0] in;
    output reg [3:0] out;

    always @(*) begin
        if (in > 4'b1001) begin
            out = 4'b1001;
        end
        else begin
            out = in;
        end
    end
end
endmodule
```

This module is to make sure that the switch that represents showing numbers from zero to nine works as intended since any input more than nine should be shown as a nine. When the number is less than nine it will show the current value.

Decimal:

```
module dec (in, out1, out2);
    input [3:0] in;
    output [3:0] out1, out2;
    reg [3:0] ten = 4'b1010;
    reg [3:0] lsb, msb;

    always @(*) begin
        if (in < 4'b1010) begin
            lsb = in;
            msb = 4'b0000;
        end
        else begin
            lsb = in - ten;
            msb = 4'b0001;
        end
    end

    Converter S0(lsb, out1);
    assign out2 = msb;
endmodule
```

What the Decimal module does is make sure that the displays can show numbers up to fifteen. When the value is less than ten then the rightmost display shows the number and the display left of that shows a zero. If the number is more than ten then the rightmost display shows the number of the input value minus ten and the left display shows a one.

Mux:

```
module mux(s0, s1, s2, hex, binary, baseten, out);
    input s0, s1, s2;
    input [3:0] hex, binary, baseten;
    output reg [3:0] out;
    wire [2:0] sel;

    assign sel = {s2, s1, s0};

    always @(*)begin
        if(sel == 3'b001)begin
```

```

        out = hex;
    end
    else if(sel == 3'b010 || sel == 3'b011)begin
        out = binary;
    end
    else if (sel == 3'b100 || sel == 3'b110 || sel == 3'b101 || sel == 3'b111)begin
        out = baseten;
    end
    else begin
        out = 4'b0000;
    end
end
endmodule

```

The purpose of this module is to select the correct type of result that is needed since how this entire program works are that it has all possible results computed before the program knows which result is needed. The reason why this is done is that it was easier to understand in our heads than picking the program after knowing what the user wants.

Encoder

```

module encoder (s0, s1, s2, s3, s4, s5, s6, in, enable);
    input enable;
    input [3:0] in;//declare as a 4bit binary
    output reg s0, s1, s2, s3, s4, s5, s6;

    always @(*) begin
        if(enable)begin
            if (in==4'b0000) begin
                s0=0;
                s1=0;
                s2=0;
                s3=0;
                s4=0;
                s5=0;
                s6=1;
            end
            else if (in==4'b0001) begin
                s0=1;
                s1=0;
                s2=0;
                s3=1;
                s4=1;
                s5=1;
                s6=1;
            end
        end
    end
endmodule

```

```
end
else if (in==4'b0010) begin
    s0=0;
    s1=0;
    s2=1;
    s3=0;
    s4=0;
    s5=1;
    s6=0;
end
else if (in==4'b0011) begin
    s0=0;
    s1=0;
    s2=0;
    s3=0;
    s4=1;
    s5=1;
    s6=0;
end
else if (in==4'b0100) begin
    s0=1;
    s1=0;
    s2=0;
    s3=1;
    s4=1;
    s5=0;
    s6=0;
end
else if (in==4'b0101) begin
    s0=0;
    s1=1;
    s2=0;
    s3=0;
    s4=1;
    s5=0;
    s6=0;
end
else if (in==4'b0110) begin
    s0=0;
    s1=1;
    s2=0;
    s3=0;
    s4=0;
    s5=0;
```

```
        s6=0;
end
else if (in==4'b0111) begin
    s0=0;
    s1=0;
    s2=0;
    s3=1;
    s4=1;
    s5=1;
    s6=1;
end
else if (in==4'b1000) begin
    s0=0;
    s1=0;
    s2=0;
    s3=0;
    s4=0;
    s5=0;
    s6=0;
end
else if (in==4'b1001) begin
    s0=0;
    s1=0;
    s2=0;
    s3=1;
    s4=1;
    s5=0;
    s6=0;
end
else if (in==4'b1010) begin
    s0=0;
    s1=0;
    s2=0;
    s3=1;
    s4=0;
    s5=0;
    s6=0;
end
else if (in==4'b1011) begin
    s0=1;
    s1=1;
    s2=0;
    s3=0;
    s4=0;
```

```
        s5=0;
        s6=0;
    end
    else if (in==4'b1100) begin
        s0=0;
        s1=1;
        s2=1;
        s3=0;
        s4=0;
        s5=0;
        s6=1;
    end
    else if (in==4'b1101) begin
        s0=1;
        s1=0;
        s2=0;
        s3=0;
        s4=0;
        s5=1;
        s6=0;
    end
    else if (in==4'b1110) begin
        s0=0;
        s1=1;
        s2=1;
        s3=0;
        s4=0;
        s5=0;
        s6=0;
    end
    else if (in==4'b1111) begin
        s0=0;
        s1=1;
        s2=1;
        s3=1;
        s4=0;
        s5=0;
        s6=0;
    end
    else begin
        s0=0;
        s1=0;
        s2=0;
        s3=1;
```

```

        s4=1;
        s5=1;
        s6=1;
    end
end
else begin
    s0 = 0;
    s1 = 0;
    s2 = 0;
    s3 = 0;
    s4 = 0;
    s5 = 0;
    s6 = 1;
end
end
endmodule

```

The purpose of this module is to translate the four-bit number to a seven-bit number that the seven-segment display can understand. The code above might be a lot, but all the code is necessary seeing that you need zero through nine have to be shown as well as A through F for the hex value.

Type:

```

module type (sw1, sw2, sw3, out);
    input sw1, sw2, sw3;
    output reg[6:0] out;

    always @(*)begin
        if(sw3 == 1) begin
            out = {1'b0,1'b1,1'b0,1'b0,1'b0,1'b0,1'b1}; //H
        end
        else if(sw2 == 1) begin
            out = {1'b0,1'b0,1'b0,1'b0,1'b0,1'b1,1'b1}; //b
        end
        else if(sw1 == 1) begin
            out = {1'b0,1'b0,1'b0,1'b1,1'b0,1'b0,1'b1}; //d
        end
        else begin
            out = {1'b0,1'b0,1'b0,1'b0,1'b1,1'b1,1'b0}; //E
        end
    end
end
endmodule

```


The purpose of this module is to translate the button pressed to the left-most seven-segment display. This part of the lab was not talked about much but since it's not much different than what we already did, we decided to include it.

“Parent” Module

```
module Lab4(in, outlsb, outmsb, outtype, sw1, sw2, sw3);
    input [3:0] in;
    input sw1, sw2, sw3;
    output [6:0] outlsb, outmsb, outtype;
    wire [4:0] hex, fourbit, baseten1, baseten2, decode;
    wire b1, b2, b3;

    assign b1 = ~sw1;
    assign b2 = ~sw2;
    assign b3 = ~sw3;

    assign hex = in;
    Converter bit(in, fourbit);
    dec base10(in, baseten1, baseten2);

    mux lsb(b1, b2, b3, hex, fourbit, baseten1, decode);

    encoder LSB(outlsb[0], outlsb[1], outlsb[2], outlsb[3], outlsb[4], outlsb[5], outlsb[6],
    decode, 1'b1);
    encoder MSB(outmsb[0], outmsb[1], outmsb[2], outmsb[3], outmsb[4], outmsb[5],
    outmsb[6], baseten2, b3);
    type display(b1, b2, b3, outtype);
endmodule
```

The purpose of this module is to be the highest-level module and to connect everything together. This module also converts the negative logic of the pushbuttons to positive logic because everything was written for positive logic before we found out that the buttons were negative logic. The first thing that this module does is to process all the possible results before using the mux and picking the user-selected result. Then after that, the input goes into the encoders where they are changed to seven-bits so the seven-segments can understand what to display.

Results/ Verification

ModelSim:

```
module Lab4Test();
    reg sw1, sw2, sw3;
```

```

reg [3:0] in;
wire [6:0] outlsb, outmsb, outtype;
parameter time_out = 100;

initial $monitor (in, sw1, sw2, sw3, outtype, outmsb, outlsb);
always
begin
    #0 in = 4'b0000; sw1 = 1; sw2 = 1; sw3 = 1;
    #5 in = 4'b1001; sw1 = 0;
    #5 sw1 = 1; sw2 = 0;
    #5 sw2 = 1; sw3 = 0;
    #5 sw1 = 0;
    #5 sw2 = 0;
    #5 sw3 = 1;
    #5 sw2 = 1;
    #5 sw1 = 1;
    #5 in = 4'b1111; sw1 = 0;
    #5 sw1 = 1; sw2 = 0;
    #5 sw2 = 1; sw3 = 0;
    #5 sw3 = 0;
end
Lab4 test(in, outlsb, outmsb, outtype, sw1, sw2, sw3);
endmodule

```

The testbench is based on previous test benches and had the parameters that need to be monitored changed to monitor the three switches, the four-bit input, and the three seven-segment displays.

▶	/Lab4Test/sw1	1
▶	/Lab4Test/sw2	0
▶	/Lab4Test/sw3	1
+	▶ /Lab4Test/in	1001
+	▶ /Lab4Test/outlsb	0011000
+	▶ /Lab4Test/outmsb	1000000
+	▶ /Lab4Test/outtype	0000011

0000	1001								1111			0000	1001
1000000	0011000							1000000	0001110	0011000	0010010	1000000	0011000
1000000											1111001	1000000	
0000110	0001001	0000011	0100001				0000011	0001001	0000110	0001001	0000011	0100001	0000110

Fig 1: Test Results of the Program

From the results, we can see that results for inputs less than ten and more than ten work for the hex, zero through nine, and base ten types. For the types of numbers that the switches represent; SW1 is for hex, while SW2 is for zero through nine, and SW3 is for base ten. When more than one button is pressed the smaller number of the switch, the high priority it has. The display for the type of switch pressed also seems to be working.

FPGA:

```
// Include the top-level pin ip file
```

```
`include "../hdl/pin_ip.v"
```

```
// GUI is DE10-Lite, but actual FPGA is DE0-CV
```

```
module top( input CYCLONEV_CLK_50 );
```

```
    // Peripheral interconnect wires
```

```
    // Use these wires to feed outputs to LED widgets
```

```
    wire [9:0]    LEDR;
```

```
    // Use these wires as 8-bit active-low 7-segment
```

```
    // outputs, where the 8th bit is the decimal point
```

```
    wire [7:0]    HEX0, HEX1, HEX2,  
                  HEX3, HEX4, HEX5;
```

```
    // Use these wires as slide switch inputs
```

```
    wire [9:0]    SW;
```

```
    // Use these wires as active low, push-button inputs
```

```
    wire [1:0]    KEY;
```

```
    // These function as programmable register inputs to a
```

```
    // design, useful for adjusting inputs using a keyboard
```

```
    wire [31:0] param1, param2, param3;
```

```
    // User instantiates design below
```

```
    Lab4 test({SW[3], SW[2], SW[1], SW[0]}, HEX0, HEX1, HEX5, KEY[0], KEY[1],  
SW[9]);
```

```
    /*
```

```
    **
```

```
    **      Instantiated design connects to pin_ip connected wires
```

```
    ** these wires function as memory-mapped i/o which is
```

```
    ** translated to widgets on the GUI
```

```
    **
```

```
    */
```

```
    // IP to allow simple user design interfacing with development kit
```

```
        pin_ip          platform_designer_pin_ip(
.CYCLONEV_CLK_50(CYCLONEV_CLK_50),

        .LEDR(LEDR),

        .HEX0(HEX0),

        .HEX1(HEX1),

        .HEX2(HEX2),

        .HEX3(HEX3),

        .HEX4(HEX4),

        .HEX5(HEX5),

        .SW(SW),

        .KEY(KEY),

        .param1(param1),

        .param2(param2),

        .param3(param3));

endmodule
```

This FPGA implementation code was mainly provided, by Intel and the only thing that has been changed is that the user-defined design was added. This is the single line of code that calls to use Lab4.v

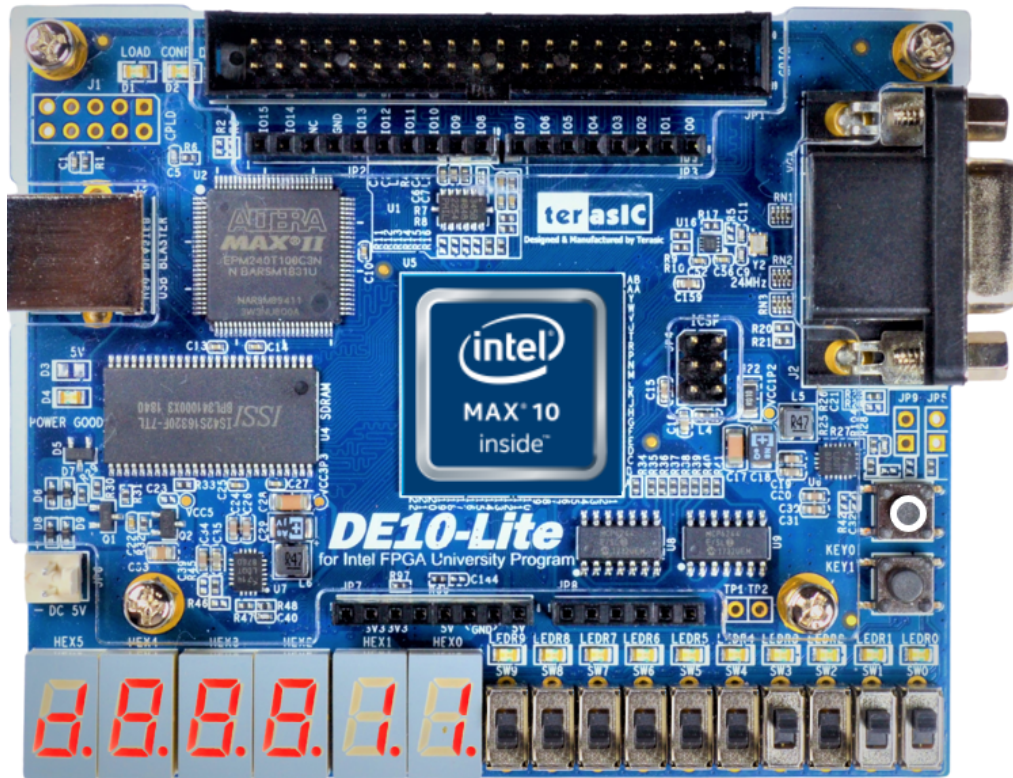


Fig 2: 0-15 option chosen as well as hex option

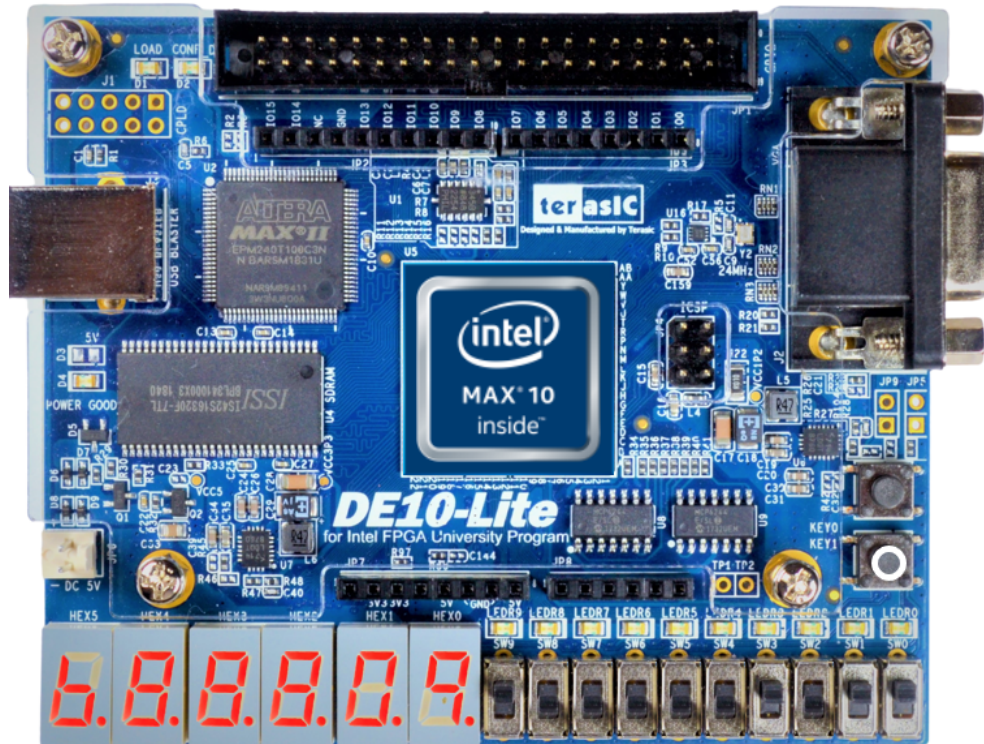


Fig 3: 0-9 option chosen

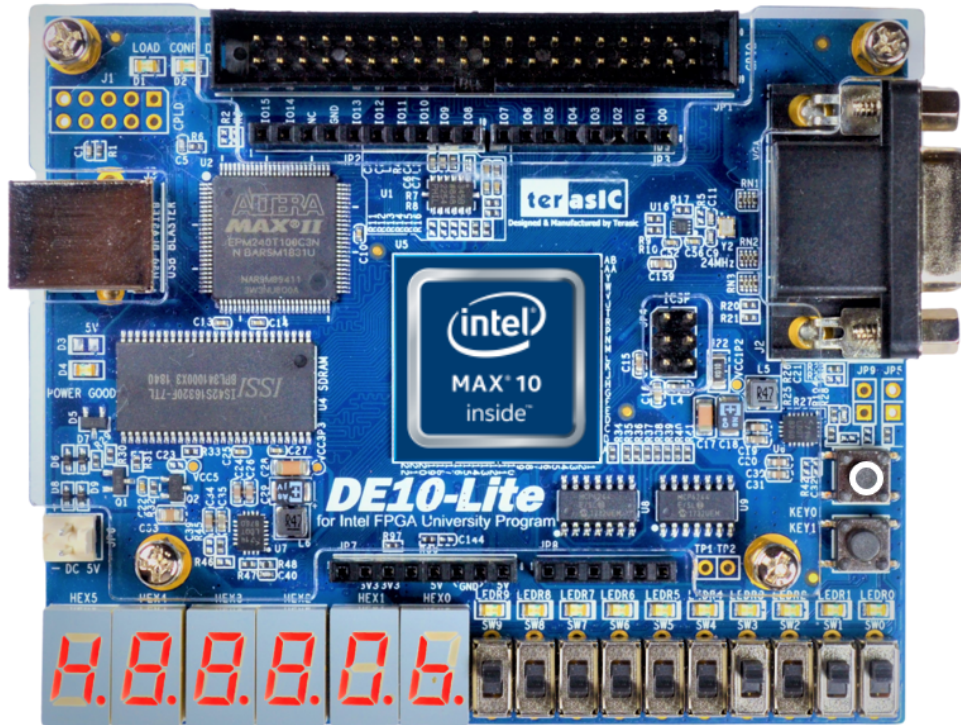


Fig 4: Hex option chosen

From the results of the FPGA, this is working as intended where three buttons with negative logic control how the input is displayed. With the binary number of eleven imputed, the 0-15 option displays eleven, the 0-9 option shows nine, and the hex option displays a “b”. There is also an example of what happens when two buttons are pressed at the same time. When the 0-15 option and hex options are pressed at the same time, the 0-15 has priority. This is because 0-15 has a higher priority than hex.

Conclusion and Discussion

When a button is pressed the input number is shown on two seven-segment displays and the type of number is shown on another display. SW1 does a hex output and uses one display and SW2 does a base-ten output, but also only uses one display. SW3 does a base-ten output on two displays and as a result, can display up to the number fifteen. Also, each time a button is pressed another display separate from the other two tells if the output is a hex, zero through nine, or a base-ten number with the letters H, B, D respectfully. If an error occurs then the letter E shows up on that display. As a group, we didn’t run into any problems other than finding out that almost everything was using negative logic and the modules that we wrote, for the most part, were using positive logic.

Work Breakdown

The base10, Converter, and type modules were done by Aaron. The encoder class and the mux module were done by Ricky. As for the highest level module, the testbench and the FPGA implementation was a joint effort of both members.

Prelabs

Aaron Luong:

What I would do to facilitate the signals to the outputs with the button press is with 3 different modules that do the different tasks needed when each button was pressed. If the buttons can only be pressed one at a time then a 4x1 mux can be used and if they can be pressed at the same time then an 8x1 mux can be used instead. Then after this, the output can then be converted to hex values and then can be decoded by a seven-segment decoder and output can be put into the seven-segment screen.

Display	Input	S_6	S_5	S_4	S_3	S_2	S_1	S_0
0	0000	1	1	1	1	1	0	1
1	0001	0	0	1	1	0	0	0
2	0010	1	0	0	1	1	1	1
3	0011	0	0	1	1	1	1	1
4	0100	0	1	1	1	0	1	0
5	0101	0	1	1	0	1	1	1
6	0110	1	1	1	0	1	1	1
7	0111	0	0	1	1	0	0	1
8	1000	1	1	1	1	1	1	1
9	1001	0	1	1	1	0	1	1
(A)	1010	1	1	1	1	0	1	1
(B)	1011	1	1	1	0	1	1	0
(C)	1100	1	1	0	0	1	0	1
(D)	1101	1	0	1	1	1	1	0
(E)	1110	1	1	0	0	1	1	1
(F)	1111	1	1	0	0	0	1	1

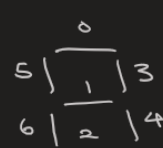


Fig 5:
Aaron
Luong's
Prelab

Ricky Couto:

Similar to Aaron, the best solution I could think of was to tie each input to its display code to ensure that all of the right LED would light. These seven-bit display outputs will be simultaneously sent to a 4x1 mux. This satisfies our 3 input scenarios as well as an error state.

3/9/21 Lab 4 Prelab Engr 378

Task 3)	Display	Input	s_0	s_1	s_2	s_3	s_4	s_5	s_6
	0	0000	1	1	1	1	1	0	1
	1	0001	0	0	1	1	0	0	0
s_6	2	0010	1	0	0	1	1	1	1
s_1, s_3, s_5	3	0011	0	1	1	1	0	1	0
s_0, s_2, s_4	4	0100	0	1	1	1	0	1	0
	5	0101	0	1	1	0	1	1	1
	6	0110	1	1	1	0	1	1	1
	7	0111	0	0	1	1	0	0	1
	8	1000	1	1	1	1	1	1	1
	9	1001	0	1	1	1	0	1	1
10	A	1010	1	1	1	1	0	1	1
	B	1011	1	1	1	0	1	1	1
12	C	1100	1	1	0	0	1	0	1
13	D	1101	1	1	1	1	1	1	0
14	E	1110	1	1	0	0	1	1	1
15	F	1111	1	1	0	0	0	1	1

Map

0 = 0
 6 = 1
 3 = 2
 1 = 3
 2 = 4
 5 = 5
 4 = 6

0 is high

Char	Input	s_0	s_1	s_2	s_3	s_4	s_5	s_6
0	0000	0	0	0	0	0	0	1
1	0001	1	0	0	1	1	1	1
2	0010	0	0	1	0	0	1	1
3	0011	0	0	0	0	1	1	1
4	0100	1	0	0	1	1	0	0
5	0101	0	1	0	0	1	0	0
6	0110	0	1	0	0	1	0	0
7	0111	0	0	0	1	1	1	1
8	1000	0	0	0	0	0	0	0
9	1001	0	0	0	1	1	0	0
A	1010	0	0	0	1	0	0	0
B	1011	1	1	0	0	0	0	0
C	1100	0	1	1	0	0	0	1
D	1101	1	0	0	0	0	1	0
E	1110	0	1	1	0	0	0	0
F	1111	0	1	1	0	0	0	0

Fig 6: Ricky Couto's Prelab

HDL Source Code

```
module Lab4Test();
    reg sw1, sw2, sw3;
    reg [3:0] in;
    wire [6:0] outlsb, outmsb, outtype;
    parameter time_out = 100;

    initial $monitor (in, sw1, sw2, sw3, outtype, outmsb, outlsb);
    always
    begin
        #0 in = 4'b0000; sw1 = 1; sw2 = 1; sw3 = 1;
        #5 in = 4'b1001; sw1 = 0;
        #5 sw1 = 1; sw2 = 0;
        #5 sw2 = 1; sw3 = 0;
        #5 sw1 = 0;
        #5 sw2 = 0;
        #5 sw3 = 1;
        #5 sw2 = 1;
        #5 sw1 = 1;
        #5 in = 4'b1111; sw1 = 0;
        #5 sw1 = 1; sw2 = 0;
        #5 sw2 = 1; sw3 = 0;
        #5 sw3 = 0;
    end
    Lab4 test(in, outlsb, outmsb, outtype, sw1, sw2, sw3);
endmodule
```

```
module Lab4(in, outlsb, outmsb, outtype, sw1, sw2, sw3);
    input [3:0] in;
    input sw1, sw2, sw3;
    output [6:0] outlsb, outmsb, outtype;
    wire [4:0] hex, fourbit, baseten1, baseten2, decode;
    wire b1, b2, b3;

    assign b1 = ~sw1;
    assign b2 = ~sw2;
    assign b3 = ~sw3;

    assign hex = in;
    Converter bit(in, fourbit);
    dec base10(in, baseten1, baseten2);

    mux lsb(b1, b2, b3, hex, fourbit, baseten1, decode);
endmodule
```

```

        encoder LSB(outlsb[0], outlsb[1], outlsb[2], outlsb[3], outlsb[4], outlsb[5], outlsb[6],
decode, 1'b1);
        encoder MSB(outmsb[0], outmsb[1], outmsb[2], outmsb[3], outmsb[4], outmsb[5],
outmsb[6], baseten2, b3);
        type display(b1, b2, b3, outtype);
endmodule

```

```

module type (sw1, sw2, sw3, out);
    input sw1, sw2, sw3;
    output reg[6:0] out;

    always @(*)begin
        if(sw3 == 1) begin
            out = {1'b0,1'b1,1'b0,1'b0,1'b0,1'b0,1'b1}; //H
        end
        else if(sw2 == 1) begin
            out = {1'b0,1'b0,1'b0,1'b0,1'b0,1'b1,1'b1}; //b
        end
        else if(sw1 == 1) begin
            out = {1'b0,1'b0,1'b0,1'b1,1'b0,1'b0,1'b1}; //d
        end
        else begin
            out = {1'b0,1'b0,1'b0,1'b0,1'b1,1'b1,1'b0}; //E
        end
    end
endmodule

```

```

module encoder (s0, s1, s2, s3, s4, s5, s6, in, enable);
    input enable;
    input [3:0] in;//declare as a 4bit binary
    output reg s0, s1, s2, s3, s4, s5, s6;

    always @(*) begin
        if(enable)begin
            if (in==4'b0000) begin
                s0=0;
                s1=0;
                s2=0;
                s3=0;
                s4=0;
                s5=0;
                s6=1;
            end
        end
    end
endmodule

```

```
else if (in==4'b0001) begin
    s0=1;
    s1=0;
    s2=0;
    s3=1;
    s4=1;
    s5=1;
    s6=1;
end
else if (in==4'b0010) begin
    s0=0;
    s1=0;
    s2=1;
    s3=0;
    s4=0;
    s5=1;
    s6=0;
end
else if (in==4'b0011) begin
    s0=0;
    s1=0;
    s2=0;
    s3=0;
    s4=1;
    s5=1;
    s6=0;
end
else if (in==4'b0100) begin
    s0=1;
    s1=0;
    s2=0;
    s3=1;
    s4=1;
    s5=0;
    s6=0;
end
else if (in==4'b0101) begin
    s0=0;
    s1=1;
    s2=0;
    s3=0;
    s4=1;
    s5=0;
    s6=0;
```

```
end
else if (in==4'b0110) begin
    s0=0;
    s1=1;
    s2=0;
    s3=0;
    s4=0;
    s5=0;
    s6=0;
end
else if (in==4'b0111) begin
    s0=0;
    s1=0;
    s2=0;
    s3=1;
    s4=1;
    s5=1;
    s6=1;
end
else if (in==4'b1000) begin
    s0=0;
    s1=0;
    s2=0;
    s3=0;
    s4=0;
    s5=0;
    s6=0;
end
else if (in==4'b1001) begin
    s0=0;
    s1=0;
    s2=0;
    s3=1;
    s4=1;
    s5=0;
    s6=0;
end
else if (in==4'b1010) begin
    s0=0;
    s1=0;
    s2=0;
    s3=1;
    s4=0;
    s5=0;
```

```
        s6=0;
end
else if (in==4'b1011) begin
    s0=1;
    s1=1;
    s2=0;
    s3=0;
    s4=0;
    s5=0;
    s6=0;
end
else if (in==4'b1100) begin
    s0=0;
    s1=1;
    s2=1;
    s3=0;
    s4=0;
    s5=0;
    s6=1;
end
else if (in==4'b1101) begin
    s0=1;
    s1=0;
    s2=0;
    s3=0;
    s4=0;
    s5=1;
    s6=0;
end
else if (in==4'b1110) begin
    s0=0;
    s1=1;
    s2=1;
    s3=0;
    s4=0;
    s5=0;
    s6=0;
end
else if (in==4'b1111) begin
    s0=0;
    s1=1;
    s2=1;
    s3=1;
    s4=0;
```

```

        s5=0;
        s6=0;
    end
    else begin
        s0=0;
        s1=0;
        s2=0;
        s3=1;
        s4=1;
        s5=1;
        s6=1;
    end
end
else begin
    s0 = 0;
    s1 = 0;
    s2 = 0;
    s3 = 0;
    s4 = 0;
    s5 = 0;
    s6 = 1;
end
end
endmodule

module mux(s0, s1, s2, hex, binary, baseten, out);
    input s0, s1, s2;
    input [3:0] hex, binary, baseten;
    output reg [3:0] out;
    wire [2:0] sel;

    assign sel = {s2, s1, s0};

    always @(*)begin
        if(sel == 3'b001)begin
            out = hex;
        end
        else if(sel == 3'b010 || sel == 3'b011)begin
            out = binary;
        end
        else if (sel == 3'b100 || sel == 3'b110 || sel == 3'b101 || sel == 3'b111)begin
            out = baseten;
        end
    end
endmodule

```

```

        out = 4'b0000;
    end
end
endmodule

```

```

module dec (in, out1, out2);
    input [3:0] in;
    output [3:0] out1, out2;
    reg [3:0] ten = 4'b1010;
    reg [3:0] lsb, msb;

    always @(*) begin
        if (in < 4'b1010) begin
            lsb = in;
            msb = 4'b0000;
        end
        else begin
            lsb = in - ten;
            msb = 4'b0001;
        end
    end
end

```

```

    Converter S0(lsb, out1);
    assign out2 = msb;
endmodule

```

```

module Converter(in, out);
    input [3:0] in;
    output reg [3:0] out;

    always @(*) begin
        if (in > 4'b1001) begin
            out = 4'b1001;
        end
        else begin
            out = in;
        end
    end
end
endmodule

```

```

// Include the top-level pin ip file
`include "../hdl/pin_ip.v"

```

```

// GUI is DE10-Lite, but actual FPGA is DE0-CV

```

```

module top( input CYCLONEV_CLK_50 );

    // Peripheral interconnect wires

    // Use these wires to feed outputs to LED widgets
    wire [9:0]    LEDR;
    // Use these wires as 8-bit active-low 7-segment
    // outputs, where the 8th bit is the decimal point
    wire [7:0]    HEX0, HEX1, HEX2,
                  HEX3, HEX4, HEX5;
    // Use these wires as slide switch inputs
    wire [9:0]    SW;
    // Use these wires as active low, push-button inputs
    wire [1:0]    KEY;
    // These function as programmable register inputs to a
    // design, useful for adjusting inputs using a keyboard
    wire [31:0]   param1, param2, param3;

    // User instantiates design below
    Lab4 test({SW[3], SW[2], SW[1], SW[0]}, HEX0, HEX1, HEX5, KEY[0], KEY[1],
SW[9]);

    /*
    **
    **      Instantiated design connects to pin_ip connected wires
    ** these wires function as memory-mapped i/o which is
    ** translated to widgets on the GUI
    **
    */

    // IP to allow simple user design interfacing with developent kit

    pin_ip          platform_designer_pin_ip(
.CYCLONEV_CLK_50(CYCLONEV_CLK_50),

        .LEDR(LEDR),

        .HEX0(HEX0),

        .HEX1(HEX1),

```



```
.HEX2(HEX2),  
  
.HEX3(HEX3),  
  
.HEX4(HEX4),  
  
.HEX5(HEX5),  
  
.SW(SW),  
  
.KEY(KEY),  
  
.param1(param1),  
  
.param2(param2),  
  
.param3(param3));
```

```
endmodule
```