

Lebanese American University



Digital Systems Lab

Section 33

Laboratory manual 6

Nour Kachmar: 201902597

Youssef Kourani: 202004265

Hussein Nasser: 201905733

Mostafa Rammal: 202003253

Jana Loubani: 201904551

12/01/2023

Table of Contents

List of Figures	2
List of Tables	3
Introduction	4
Experiment.....	5
State Diagram.....	5
Ps2 Keyboard Module	7
Converter Module.....	8
Password Module	9
LEDToggle Module	13
New Frequencies Module	15
LED TEST Module	16
Pin Assignment.....	19
Conclusion.....	21

List of Figures

Figure 1: State Diagram of the Safeguard	5
Figure 2: Keycode Values.....	7
Figure 3: Converter Module.....	8
Figure 4: Design of Password Module (1)	9
Figure 5: Design of Password Module (2)	10
Figure 6: Design of Password Module (3)	11
Figure 7: Design of Password Module (4)	12
Figure 8: LEDToggle Module.....	13
Figure 9: 2Hz and 8Hz Blinking Clocks.....	15
Figure 10: LCD Test Module (1)	16
Figure 11: LCD Test Module (2)	16
Figure 12: LCD Test Module (3)	17
Figure 13: LCD Test Module (4)	17
Figure 14: LCD Test Module (5)	18
Figure 15: LCD Test Module (6)	18
Figure 16: Pin Assignment (1)	19
Figure 17: Pin Assignment (2)	20

List of Tables

No table of figures entries found.

Introduction

During this lab session, we will delve into comprehending the functionality of the PS/2 keyboard, utilizing both the DE2-115 Manual and online references. The process involves reading documentation, crafting Verilog code, and establishing the design to ensure proper functionality. Quartus will be the primary tool employed throughout the lab, enabling us to build the required circuitry and conduct simulations to verify the logical coherence of the design.

Experiment

State Diagram

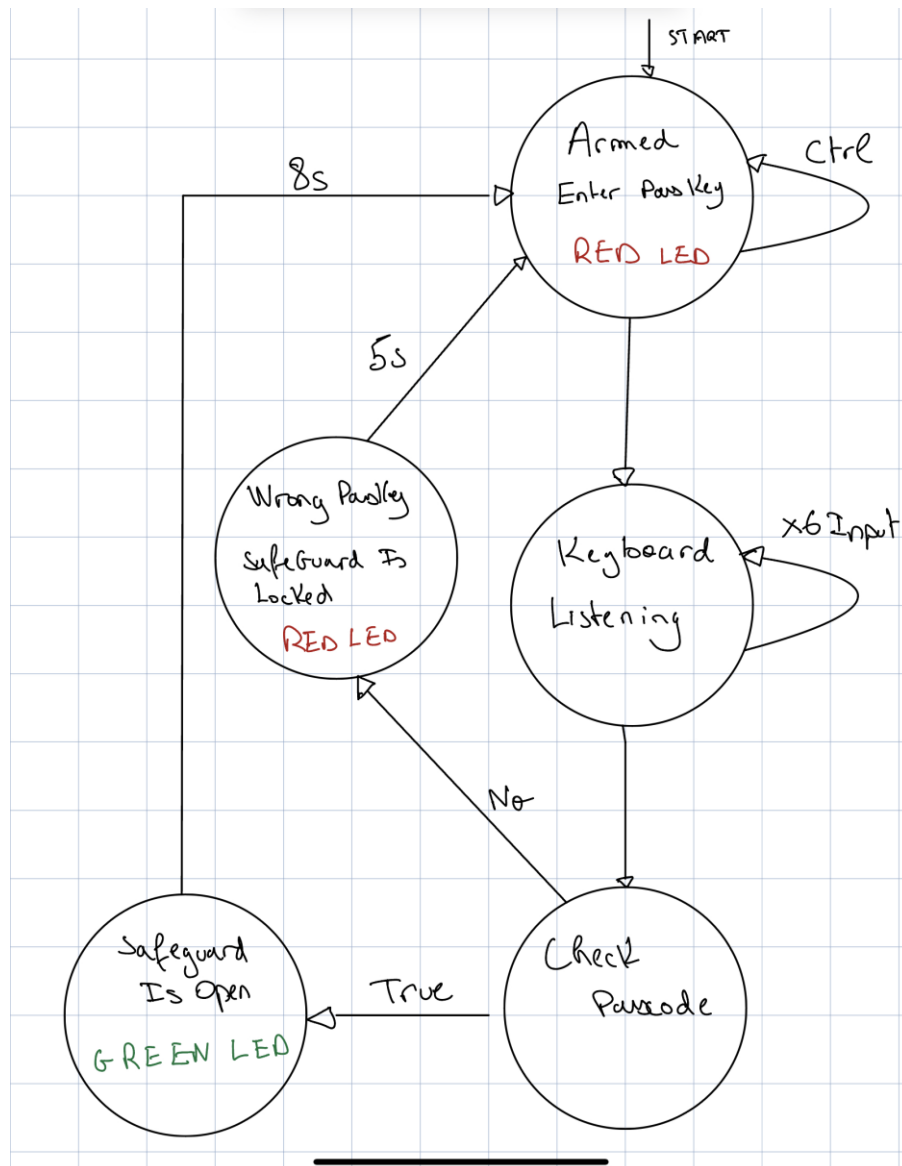


Figure 1: State Diagram of the Safeguard

Upon startup, the LCD will show "Armed Enter Passkey," accompanied by a red LED. The board awaits a sequence of six numeric inputs from the user, each spaced 2 seconds apart. Afterward, it checks if the input matches the stored value. If affirmative, "Safeguard is Open" appears on the LCD, and a Green LED lights up. After 8 seconds, the board reverts to the initial state. If the input is incorrect, "Wrong Passkey

Safeguard is locked" is displayed, and the red LED blinks for 5 seconds. The board then returns to the locked safeguard state.

Ps2 Keyboard Module

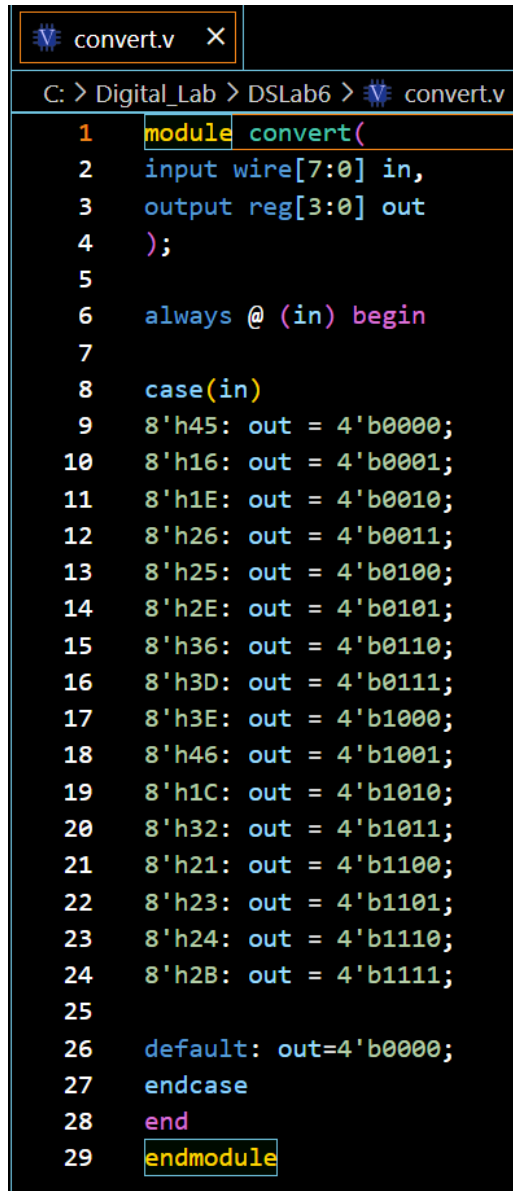
```
ps2_keyboard.v X
C: > Digital_Lab > DSLab6 > ps2_keyboard.v
82      (keycode_o==8'h52)?1:( // '
83      (keycode_o==8'h5b)?1:( // ]
84      (keycode_o==8'h4d)?1:( // P
85      (keycode_o==8'h44)?1:( // O
86      (keycode_o==8'h43)?1:( // I
87      (keycode_o==8'h35)?1:( // Y
88      (keycode_o==8'h2c)?1:( // T
89      (keycode_o==8'h24)?1:( // E
90      (keycode_o==8'h1d)?1:( // W
91      (keycode_o==8'h76)?1:( // esc
92      (keycode_o==8'h5A)?1:( // enter
93      (keycode_o==8'h29)?1:( // space
94      (keycode_o==8'h32)?1:( // B
95      (keycode_o==8'h21)?1:( // C
96      (keycode_o==8'h2A)?1:( // V
97      (keycode_o==8'h16)?1:( // 1
98      (keycode_o==8'h1E)?1:( // 2
99      (keycode_o==8'h26)?1:( // 3
100     (keycode_o==8'h25)?1:( // 4
101     (keycode_o==8'h2E)?1:( // 5
102     (keycode_o==8'h36)?1:( // 6
103     (keycode_o==8'h3d)?1:( // 7
104     (keycode_o==8'h3E)?1:( // 8
105     (keycode_o==8'h46)?1:( // 9
106     (keycode_o==8'h45)?1:( // 0
107     (keycode_o==8'h66)?1:( // back
108     (keycode_o==8'h14)?1:( // ctrl
109     (keycode_o==8'h15)?1:0 // Q
110     )))))))
111 );
```

Figure 2: Keycode Values

The ps2_keyboard receives input from the keyboard and transforms it into Hexadecimal value. In order to use the numbers 0 till 9 we added them to be recognized as keys in our keyboard module as shown above.

Note that we added letters and other buttons for the final project so that we don't re add them and for testing purposes.

Converter Module



```
1 module convert(  
2   input wire[7:0] in,  
3   output reg[3:0] out  
4 );  
5  
6 always @ (in) begin  
7  
8   case(in)  
9     8'h45: out = 4'b0000;  
10    8'h16: out = 4'b0001;  
11    8'h1E: out = 4'b0010;  
12    8'h26: out = 4'b0011;  
13    8'h25: out = 4'b0100;  
14    8'h2E: out = 4'b0101;  
15    8'h36: out = 4'b0110;  
16    8'h3D: out = 4'b0111;  
17    8'h3E: out = 4'b1000;  
18    8'h46: out = 4'b1001;  
19    8'h1C: out = 4'b1010;  
20    8'h32: out = 4'b1011;  
21    8'h21: out = 4'b1100;  
22    8'h23: out = 4'b1101;  
23    8'h24: out = 4'b1110;  
24    8'h2B: out = 4'b1111;  
25  
26    default: out=4'b0000;  
27  endcase  
28 end  
29 endmodule
```

Figure 3: Converter Module

Figure 3 shows that the hexadecimal value is then converted into binary value.

Password Module

```

Password.v X
C: > Digital_Lab > DSLab6 > Password.v
1  module Password(
2      input wire      resetn,
3      input wire      clock_50,
4      input wire [3:0] keycode_1_bcd,
5      input wire [7:0] keycode_1,
6
7      input wire      key1_active,
8      input wire      clk_1s,
9      input wire      flag_4s,
10     output reg [3:0]  lcd_state,
11     output reg        correct,
12     output reg        wrong,
13     output reg [15:0] first_num,
14     output reg [15:0] second_num,
15     output reg [15:0] third_num,
16     output reg [15:0] fourth_num,
17     output reg [15:0] fifth_num,
18     output reg [15:0] sixth_num
19 );
20
21 parameter INITIAL_STATE = 0;
22 parameter ENTER = 1;
23 parameter CHECK_VALS = 2;
24 parameter CTRL = 3;
25 parameter CHECK_VALS_CTRL = 4;
26
27
28 parameter FIRST_NUMBER = 0;
29 parameter SECOND_NUMBER = 1;
30 parameter THIRD_NUMBER = 2;
31 parameter FOURTH_NUMBER = 3;
32 parameter FIFTH_NUMBER = 4;
33 parameter SIXTH_NUMBER = 5;
34
```

Figure 4: Design of Password Module (1)

In the beginning, we defined our states and specified the password. We introduced a counter that increases by 1, synchronized with a 1-second clock. This counter assists in transitioning between digits and returning to the initial state after 8 seconds. We outlined two main situations: the regular process and the one initiated by pressing CTRL. For each digit in both scenarios, six unique cases were established. Finally, we added a 'check value' case to validate the correctness of the entered digits. The code below encapsulates the logic of our finite state machine:

```
always @(posedge clk_1s) begin
    if (~resetn) begin
        counter <= 0;
        counter10s <= 0;
    end
    else begin
        if(flag2==1)
            counter <= counter+1;
        if(flag==1)
            counter10s<=counter10s+1;
        if(flag==2)
            counter10s<=0;
    end
end

always @(posedge clock_50)
begin
    if (!resetn)
    begin
        state <= INITIAL_STATE;
        number <= FIRST_NUMBER;
        key1_active_reg <= 0;
        lcd_state <= 0;
        correct <= 0;
        wrong <= 0;
        flag2 <= 0;
        flag <= 0;
    end else begin
        key1_active_reg <= key1_active;
        if(flag2case2==1)
            countercase2 <= countercase2+1;
        if(flagcase2==1)
            counter10scase2<=counter10scase2+1;
        if(flagcase2==2)
            counter10scase2<=0;
    end
end
```

Figure 5: Design of Password Module (2)

```

Password.v X
C: > Digital_Lab > DSLab6 > Password.v
106 case (state)
107 INITIAL_STATE :
108 begin
109     lcd_state    <= 1;
110     correct      <= 0;
111     wrong        <= 0;
112     flag2        <=1;
113     if(keycode_1 == 8'h5A)begin
114         state    <= ENTER;
115         first_num<=0;
116         second_num<=0;
117         third_num<=0;
118         fourth_num<=0;
119         fifth_num<=0;
120         sixth_num<=0;
121         counter10scase2<=0;
122         number<=FIRST_NUMBER;
123     end
124     if(keycode_1 == 8'h14)begin
125         state    <= CTRL;
126         first_num<=0;
127         second_num<=0;
128         third_num<=0;
129         fourth_num<=0;
130         fifth_num<=0;
131         sixth_num<=0;
132         first_num_ctrl<=0;
133         second_num_ctrl<=0;
134         third_num_ctrl<=0;
135         fourth_num_ctrl<=0;
136         fifth_num_ctrl<=0;
137         sixth_num_ctrl<=0;
138         counter10scase2<=0;
139         number<=FIRST_NUMBER;
140         flagcase2<=1;
141     end
142 end
143

```

Figure 6: Design of Password Module (3)

```

Password.v
C:\Digital Lab > DSIab6 > Password.v

359 CHECK_VALS:
360 begin
361   flag<=1;
362   if (first_num == PASSWORD1 && second_num == PASSWORD2 && third_num == PASSWORD3 && fourth_num == PASSWORD4 && fifth_num == PASSWORD5 && sixth_num == PASSWORD6) begin
363     correct<=1;
364     wrong<=0;
365     lcd_state<=5;
366     if (counter10s==8) begin
367       flag<=2;
368       flagcase2<=2;
369       state <=INITIAL_STATE;
370     end
371   end
372 end
373 else begin
374   wrong<=1;
375   correct<=0;
376   lcd_state<=6;
377   if (counter10s==8) begin
378     flag<=2;
379     flagcase2<=2;
380     state <=INITIAL_STATE;
381   end
382 end
383 end
384
385
386 CHECK_VALS_CTRL:
387 begin
388   flag<=1;
389   if (first_num_ctrl == PASSWORD1 && second_num_ctrl == PASSWORD2 && third_num_ctrl == PASSWORD3 && fourth_num_ctrl == PASSWORD4 && fifth_num_ctrl == PASSWORD5 && sixth_num_ctrl == PASSWORD6) begin
390     correct<=1;
391     wrong<=0;
392     lcd_state<=5;
393     if (counter10s==8) begin
394       flag<=2;
395       flagcase2<=2;
396       state <=INITIAL_STATE;
397     end
398   end
399 else begin
400   wrong<=1;
401   correct<=0;
402   lcd_state<=6;
403   if (counter10s==8) begin
404     flag<=2;
405     flagcase2<=2;
406     state <=INITIAL_STATE;
407   end
408 end
409 end
410
411 endcase
412
413 end

```

Figure 7: Design of Password Module (4)

In order to showcase a "-" in the Binary-Coded Decimal (BCD) during the control case, it is necessary to assign a particular BCD value to symbolize the "-" character. Once this value is identified, it can be stored in a register whenever the control case is identified. As a result, we have two separate scenarios for password verification: one for the regular process and another for when the CTRL key is activated.

LEDToggle Module

```
LEDToggle.v X
C: > Digital_Lab > DSLab6 > LEDToggle.v
1  module LEDToggle(
2  input clk_2Hz,clk_8Hz,reset,correct, wrong,
3  output reg led_g,led_r);
4
5
6  initial begin
7  led_g=0;
8  led_r=0;
9  end
10
11 always @(posedge clk_2Hz)
12 begin
13     if(~reset)
14         led_g = 0;
15     else begin
16         if(correct==1)
17             led_g = ~led_g;
18         else
19             led_g = 0;
20     end
21 end
22
23 always @(posedge clk_8Hz)
24 begin
25     if(~reset)
26         led_r = 0;
27     else begin
28         if(wrong==1)
29             led_r = ~led_r;
30         else
31             led_r = 0;
32     end
33 end
34
35 endmodule
36
```

Figure 8: LEDToggle Module

In figure 8, we incorporated the logic to toggle the LED based on the correct or incorrect flags obtained from the state machine block. The green LED was configured to toggle at a frequency of 2Hz (slower than RED for success) , while the red LED was set to toggle at a frequency of 8Hz (fast blinking for failure). To achieve these frequencies, we introduced a new clock divider block

New Frequencies Module

```
clock_divider_8Hz_2Hz.v X
C: > Digital_Lab > DSLab6 > clock_divider_8Hz_2Hz.v
1 module clock_divider_8Hz_2Hz ( input clk_50MHz,
2                               input reset,
3                               output reg clk_8Hz ,
4                               output reg clk_2Hz
5                               );
6
7
8   reg [31:0] counter_8Hz;
9   reg [31:0] counter_2Hz;
10
11   always @ (posedge clk_50MHz or negedge reset)
12   begin
13       if (~reset) begin
14           counter_2Hz <= 32'd0;
15           clk_2Hz <= 1'b0;
16           counter_8Hz <= 32'd0;
17           clk_8Hz <= 1'b0;
18       end else begin
19           counter_2Hz <= counter_2Hz + 32'd1;
20           counter_8Hz <= counter_8Hz + 32'd1;
21
22           if (counter_2Hz == 32'd12500000)
23           begin
24               clk_2Hz <= ~clk_2Hz;
25               counter_2Hz <= 32'd0;
26           end
27
28           if (counter_8Hz == 32'd3125000)
29           begin
30               clk_8Hz <= ~clk_8Hz;
31               counter_8Hz <= 32'd0;
32           end
33       end
34   end
35 end
36
37
38
39 endmodule
```

Figure 9: 2Hz and 8Hz Blinking Clocks

Figure 9 shows that we have a 1 second counter that we reset when 1 second passes. We also open the door and send back the system to the waiting for valid state.

LED TEST Module

```

LCD_TEST.v X
C: > Digital_Lab > DSIab6 > LCD_TEST.v
77  if(lcd_state == 1) begin
78      case(LUT_INDEX)
79          // Initial
80          LCD_INTIAL+0: LUT_DATA    <= 9'h038;
81          LCD_INTIAL+1: LUT_DATA    <= 9'h00C;
82          LCD_INTIAL+2: LUT_DATA    <= 9'h001;
83          LCD_INTIAL+3: LUT_DATA    <= 9'h006;
84          LCD_INTIAL+4: LUT_DATA    <= 9'h080;
85          // Line 1
86
87          LCD_LINE1+0:  LUT_DATA    <= 9'h141; //Armed
88          LCD_LINE1+1:  LUT_DATA    <= 9'h172;
89          LCD_LINE1+2:  LUT_DATA    <= 9'h16D;
90          LCD_LINE1+3:  LUT_DATA    <= 9'h165;
91          LCD_LINE1+4:  LUT_DATA    <= 9'h164;
92          LCD_LINE1+5:  LUT_DATA    <= 9'h120;
93          LCD_LINE1+6:  LUT_DATA    <= 9'h120;
94          LCD_LINE1+7:  LUT_DATA    <= 9'h120;
95          LCD_LINE1+8:  LUT_DATA    <= 9'h120;
96          LCD_LINE1+9:  LUT_DATA    <= 9'h120;
97          LCD_LINE1+10: LUT_DATA    <= 9'h120;
98          LCD_LINE1+11: LUT_DATA    <= 9'h120;
99          LCD_LINE1+12: LUT_DATA    <= 9'h120;
100         LCD_LINE1+13: LUT_DATA    <= 9'h120;
101         LCD_LINE1+14: LUT_DATA    <= 9'h120;
102         LCD_LINE1+15: LUT_DATA    <= 9'h120;
103         // Change Line
104         LCD_CH_LINE:   LUT_DATA    <= 9'h0C0;
105
106         LCD_LINE2+0:   LUT_DATA    <= 9'h145; //Enter PassKey

```

Figure 10: LCD Test Module (1)

```

103         // Change Line
104         LCD_CH_LINE:   LUT_DATA    <= 9'h0C0;
105
106         LCD_LINE2+0:   LUT_DATA    <= 9'h145; //Enter PassKey
107         LCD_LINE2+1:   LUT_DATA    <= 9'h16E;
108         LCD_LINE2+2:   LUT_DATA    <= 9'h174;
109         LCD_LINE2+3:   LUT_DATA    <= 9'h165;
110         LCD_LINE2+4:   LUT_DATA    <= 9'h172;
111         LCD_LINE2+5:   LUT_DATA    <= 9'h120;
112         LCD_LINE2+6:   LUT_DATA    <= 9'h150;
113         LCD_LINE2+7:   LUT_DATA    <= 9'h161;
114         LCD_LINE2+8:   LUT_DATA    <= 9'h173;
115         LCD_LINE2+9:   LUT_DATA    <= 9'h173;
116         LCD_LINE2+10:  LUT_DATA    <= 9'h148;
117         LCD_LINE2+11:  LUT_DATA    <= 9'h165;
118         LCD_LINE2+12:  LUT_DATA    <= 9'h179;
119         LCD_LINE2+13:  LUT_DATA    <= 9'h120;
120         LCD_LINE2+14:  LUT_DATA    <= 9'h120;
121         LCD_LINE2+15:  LUT_DATA    <= 9'h120;
122         default:       LUT_DATA    <= 9'h120;
123     endcase
124 end

```

Figure 11: LCD Test Module (2)

In figure 10 and 11, when the lcd_state is 1, we print “Armed Enter Passkey” so that the user has the option to enter the passkey.

```

LCD_TEST.v X
C: > Digital_Lab > DSIab6 > LCD_TEST.v
125
126 if(lcd_state == 5) begin
127     case(LUT_INDEX)
128         // Initial
129         LCD_INTIAL+0: LUT_DATA    <= 9'h038;
130         LCD_INTIAL+1: LUT_DATA    <= 9'h00C;
131         LCD_INTIAL+2: LUT_DATA    <= 9'h001;
132         LCD_INTIAL+3: LUT_DATA    <= 9'h006;
133         LCD_INTIAL+4: LUT_DATA    <= 9'h080;
134         // Line 1
135
136         LCD_LINE1+0:  LUT_DATA    <= 9'h153; //Safe
137         LCD_LINE1+1:  LUT_DATA    <= 9'h161;
138         LCD_LINE1+2:  LUT_DATA    <= 9'h166;
139         LCD_LINE1+3:  LUT_DATA    <= 9'h165;
140         LCD_LINE1+4:  LUT_DATA    <= 9'h167;
141         LCD_LINE1+5:  LUT_DATA    <= 9'h175;
142         LCD_LINE1+6:  LUT_DATA    <= 9'h161;
143         LCD_LINE1+7:  LUT_DATA    <= 9'h172;
144         LCD_LINE1+8:  LUT_DATA    <= 9'h164;
145         LCD_LINE1+9:  LUT_DATA    <= 9'h120;
146         LCD_LINE1+10: LUT_DATA    <= 9'h120;
147         LCD_LINE1+11: LUT_DATA    <= 9'h120;
148         LCD_LINE1+12: LUT_DATA    <= 9'h120;
149         LCD_LINE1+13: LUT_DATA    <= 9'h120;
150         LCD_LINE1+14: LUT_DATA    <= 9'h120;
151         LCD_LINE1+15: LUT_DATA    <= 9'h120;
152         // Change Line
153         LCD_CH_LINE:  LUT_DATA    <= 9'h0C0;
154         // Line 2

```

Figure 12: LCD Test Module (3)

```

152         // Change Line
153         LCD_CH_LINE:  LUT_DATA    <= 9'h0C0;
154         // Line 2
155         LCD_LINE2+0:  LUT_DATA    <= 9'h149; // Is OPened
156         LCD_LINE2+1:  LUT_DATA    <= 9'h173;
157         LCD_LINE2+2:  LUT_DATA    <= 9'h120;
158         LCD_LINE2+3:  LUT_DATA    <= 9'h14F;
159         LCD_LINE2+4:  LUT_DATA    <= 9'h170;
160         LCD_LINE2+5:  LUT_DATA    <= 9'h165;
161         LCD_LINE2+6:  LUT_DATA    <= 9'h16E;
162         LCD_LINE2+7:  LUT_DATA    <= 9'h165;
163         LCD_LINE2+8:  LUT_DATA    <= 9'h164;
164         LCD_LINE2+9:  LUT_DATA    <= 9'h120;
165         LCD_LINE2+10: LUT_DATA    <= 9'h120;
166         LCD_LINE2+11: LUT_DATA    <= 9'h120;
167         LCD_LINE2+12: LUT_DATA    <= 9'h120;
168         LCD_LINE2+13: LUT_DATA    <= 9'h120;
169         LCD_LINE2+14: LUT_DATA    <= 9'h120;
170         LCD_LINE2+15: LUT_DATA    <= 9'h120;
171         default:      LUT_DATA    <= 9'h120;
172     endcase
173 end

```

Figure 13: LCD Test Module (4)

In figure 12 and 13, when lcd_state is 5, we print “Safe Is Opened” on the LCD screen so that we know that the Safe has been opened successfully.

```

LCD_TEST.v X
C: > Digital_Lab > DSIab6 > LCD_TEST.v

175 if(lcd_state == 6) begin
176     case(LUT_INDEX)
177         // Initial
178         LCD_INTIAL+0: LUT_DATA    <= 9'h038;
179         LCD_INTIAL+1: LUT_DATA    <= 9'h00C;
180         LCD_INTIAL+2: LUT_DATA    <= 9'h001;
181         LCD_INTIAL+3: LUT_DATA    <= 9'h006;
182         LCD_INTIAL+4: LUT_DATA    <= 9'h080;
183         // Line 1
184         //57 72 6F 6E 67 20 41 6E 73 77 65 72 21
185         LCD_LINE1+0: LUT_DATA    <= 9'h157; //Wrong Answer!
186         LCD_LINE1+1: LUT_DATA    <= 9'h172;
187         LCD_LINE1+2: LUT_DATA    <= 9'h16F;
188         LCD_LINE1+3: LUT_DATA    <= 9'h16E;
189         LCD_LINE1+4: LUT_DATA    <= 9'h167;
190         LCD_LINE1+5: LUT_DATA    <= 9'h120;
191         LCD_LINE1+6: LUT_DATA    <= 9'h150;
192         LCD_LINE1+7: LUT_DATA    <= 9'h161;
193         LCD_LINE1+8: LUT_DATA    <= 9'h173;
194         LCD_LINE1+9: LUT_DATA    <= 9'h173;
195         LCD_LINE1+10: LUT_DATA   <= 9'h120;
196         LCD_LINE1+11: LUT_DATA   <= 9'h148;
197         LCD_LINE1+12: LUT_DATA   <= 9'h165;
198         LCD_LINE1+13: LUT_DATA   <= 9'h179;
199         LCD_LINE1+14: LUT_DATA   <= 9'h120;
200         LCD_LINE1+15: LUT_DATA   <= 9'h120;
201         // Change Line
202         LCD_CH_LINE: LUT_DATA    <= 9'h0C0;

```

Figure 14: LCD Test Module (5)

```

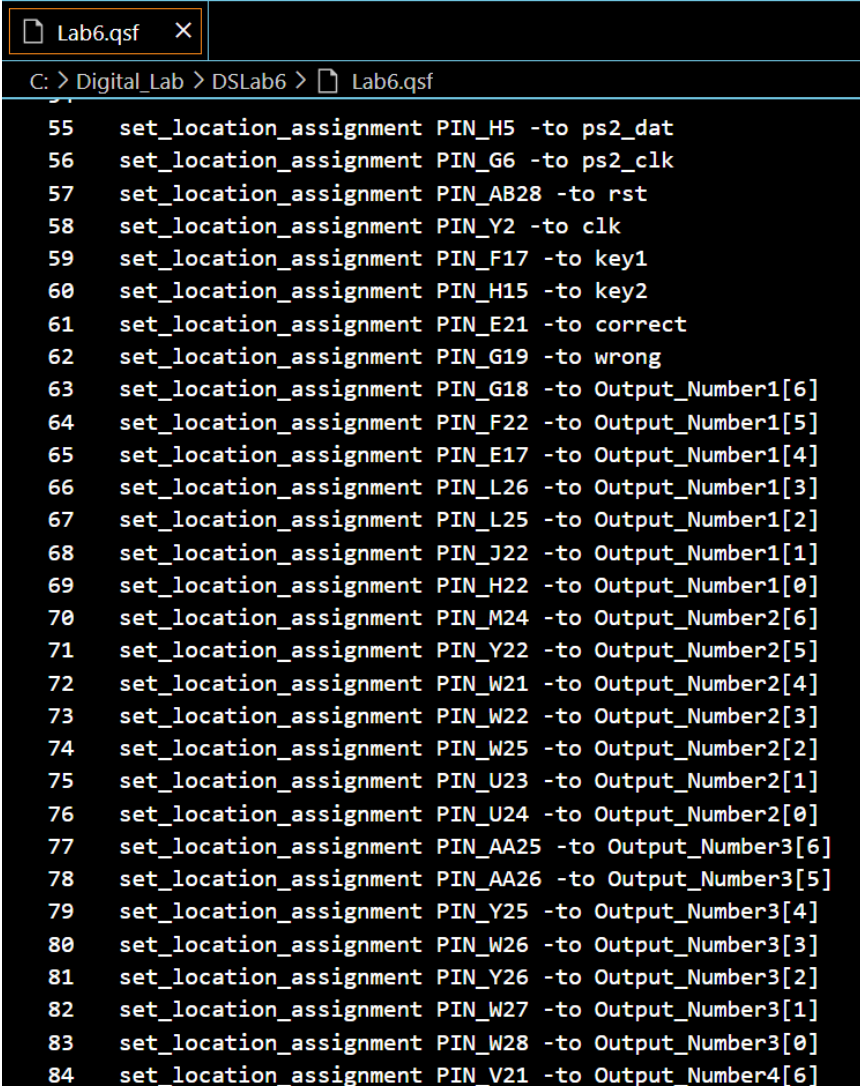
202     LCD_CH_LINE: LUT_DATA    <= 9'h0C0;
203     // Line 2
204     LCD_LINE2+0: LUT_DATA    <= 9'h153; //Safe
205     LCD_LINE2+1: LUT_DATA    <= 9'h161;
206     LCD_LINE2+2: LUT_DATA    <= 9'h166;
207     LCD_LINE2+3: LUT_DATA    <= 9'h165;
208     LCD_LINE2+4: LUT_DATA    <= 9'h167;
209     LCD_LINE2+5: LUT_DATA    <= 9'h175;
210     LCD_LINE2+6: LUT_DATA    <= 9'h161;
211     LCD_LINE2+7: LUT_DATA    <= 9'h172;
212     LCD_LINE2+8: LUT_DATA    <= 9'h164;
213     LCD_LINE2+9: LUT_DATA    <= 9'h120;
214     LCD_LINE2+10: LUT_DATA   <= 9'h16C;
215     LCD_LINE2+11: LUT_DATA   <= 9'h16F;
216     LCD_LINE2+12: LUT_DATA   <= 9'h163;
217     LCD_LINE2+13: LUT_DATA   <= 9'h16B;
218     LCD_LINE2+14: LUT_DATA   <= 9'h165;
219     LCD_LINE2+15: LUT_DATA   <= 9'h164;
220     default: LUT_DATA    <= 9'h120;
221     endcase
222 end

```

Figure 15: LCD Test Module (6)

In figure 14 and 15, when lcd_state is 6, the LCD prints “Wrong Answer!” so that the user knows that a wrong password was entered.

Pin Assignment



The image shows a screenshot of a text editor window titled "Lab6.qsf". The address bar indicates the file path: "C: > Digital_Lab > DSLab6 > Lab6.qsf". The main text area contains a list of 30 lines of code, numbered 55 to 84. Each line is a "set_location_assignment" command, mapping a specific pin to a hardware component or signal. The assignments are as follows:

```
55  set_location_assignment PIN_H5 -to ps2_dat
56  set_location_assignment PIN_G6 -to ps2_clk
57  set_location_assignment PIN_AB28 -to rst
58  set_location_assignment PIN_Y2 -to clk
59  set_location_assignment PIN_F17 -to key1
60  set_location_assignment PIN_H15 -to key2
61  set_location_assignment PIN_E21 -to correct
62  set_location_assignment PIN_G19 -to wrong
63  set_location_assignment PIN_G18 -to Output_Number1[6]
64  set_location_assignment PIN_F22 -to Output_Number1[5]
65  set_location_assignment PIN_E17 -to Output_Number1[4]
66  set_location_assignment PIN_L26 -to Output_Number1[3]
67  set_location_assignment PIN_L25 -to Output_Number1[2]
68  set_location_assignment PIN_J22 -to Output_Number1[1]
69  set_location_assignment PIN_H22 -to Output_Number1[0]
70  set_location_assignment PIN_M24 -to Output_Number2[6]
71  set_location_assignment PIN_Y22 -to Output_Number2[5]
72  set_location_assignment PIN_W21 -to Output_Number2[4]
73  set_location_assignment PIN_W22 -to Output_Number2[3]
74  set_location_assignment PIN_W25 -to Output_Number2[2]
75  set_location_assignment PIN_U23 -to Output_Number2[1]
76  set_location_assignment PIN_U24 -to Output_Number2[0]
77  set_location_assignment PIN_AA25 -to Output_Number3[6]
78  set_location_assignment PIN_AA26 -to Output_Number3[5]
79  set_location_assignment PIN_Y25 -to Output_Number3[4]
80  set_location_assignment PIN_W26 -to Output_Number3[3]
81  set_location_assignment PIN_Y26 -to Output_Number3[2]
82  set_location_assignment PIN_W27 -to Output_Number3[1]
83  set_location_assignment PIN_W28 -to Output_Number3[0]
84  set_location_assignment PIN_V21 -to Output_Number4[6]
```

Figure 16: Pin Assignment (1)

```
Lab6.qsf x
C: > Digital_Lab > DSLab6 > Lab6.qsf
85  set_location_assignment PIN_U21 -to Output_Number4[5]
86  set_location_assignment PIN_AB20 -to Output_Number4[4]
87  set_location_assignment PIN_AA21 -to Output_Number4[3]
88  set_location_assignment PIN_AD24 -to Output_Number4[2]
89  set_location_assignment PIN_AF23 -to Output_Number4[1]
90  set_location_assignment PIN_Y19 -to Output_Number4[0]
91  set_location_assignment PIN_AD18 -to Output_Number6[6]
92  set_location_assignment PIN_AC18 -to Output_Number6[5]
93  set_location_assignment PIN_AB18 -to Output_Number6[4]
94  set_location_assignment PIN_AH19 -to Output_Number6[3]
95  set_location_assignment PIN_AG19 -to Output_Number6[2]
96  set_location_assignment PIN_AF18 -to Output_Number6[1]
97  set_location_assignment PIN_AH18 -to Output_Number6[0]
98  set_location_assignment PIN_E22 -to led_g
99  set_location_assignment PIN_F19 -to led_r
100 set_location_assignment PIN_M23 -to reset_LEDs
101 set_location_assignment PIN_M1 -to LCD_RW
102 set_location_assignment PIN_M2 -to LCD_RS
103 set_location_assignment PIN_L4 -to LCD_EN
104 set_location_assignment PIN_L3 -to DATA[0]
105 set_location_assignment PIN_L1 -to DATA[1]
106 set_location_assignment PIN_L2 -to DATA[2]
107 set_location_assignment PIN_K7 -to DATA[3]
108 set_location_assignment PIN_K1 -to DATA[4]
109 set_location_assignment PIN_K2 -to DATA[5]
110 set_location_assignment PIN_M3 -to DATA[6]
111 set_location_assignment PIN_M5 -to DATA[7]
```

Figure 17: Pin Assignment (2)

Other codes used will be included in the submission.

Conclusion

In conclusion, this laboratory session provided an in-depth exploration into understanding the intricacies of the PS/2 keyboard. By leveraging resources such as the DE2-115 Manual and online references, we engaged in activities such as thorough documentation reading, Verilog code development, and design establishment to guarantee optimal functionality. The use of Quartus as the main tool throughout the lab proved instrumental in constructing the necessary circuitry and conducting simulations, ensuring the logical coherence of the design.