Objective

The purpose of this lab is to further our knowledge of Verilog and circuit design by walking the student through the process of designing a combination lock. The lock is very similiar to the locks found on high school lockers. After building and designing a 3 digit lock, the student will then have to implement a 4-digit lock by modifying the existing design.

Design

Experiment 1

The first part of **Experiment 1** consisted of simulating the implemented design of the Finite State Machine (FSM) given below.

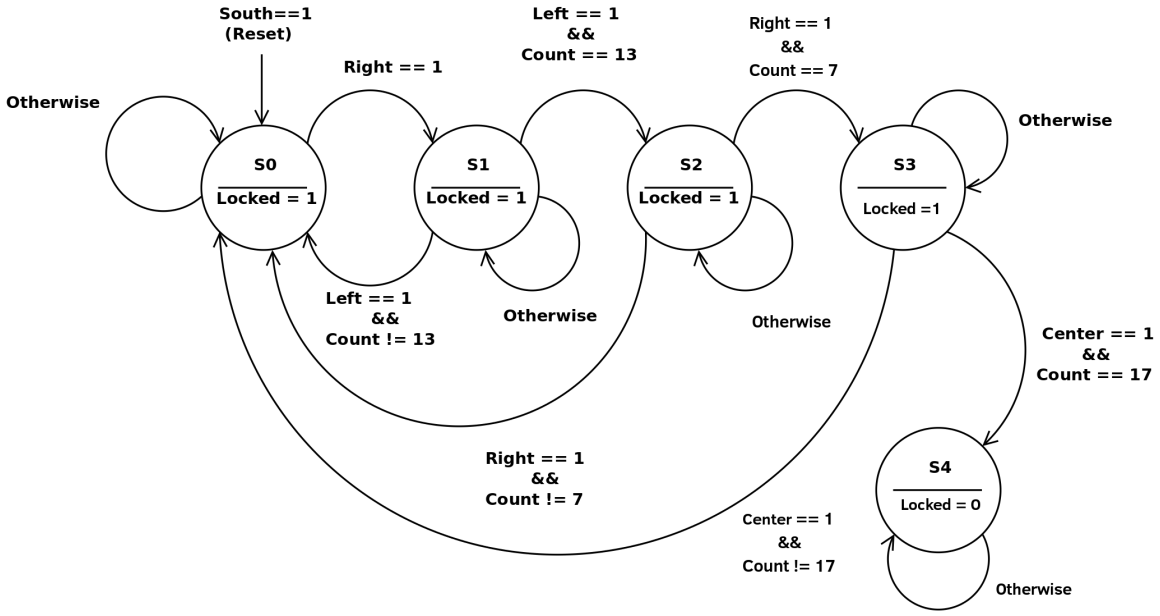

Figure 1: *Rotary Combination-Lock State Diagram*

Below is the Verilog code for the aforementioned Rotary Combination-Lock state diagram that was tested against the appropriate test bench.

Code Block 1: Combination Lock FSM

```
1 `timescale 1ns / 1ps
```

```
2
3  module combination_lock_fsm(output reg [2:0] state,
4    output wire Locked, // asserted when locked
5    input wire Right, Left, // indicate direction
6    input wire [4:0] Count, // indicate position
7    input wire Center, // the unlock button
8    input wire Clk, South // clock and reset
9    );
10   // Parameters for the 4 cases
11   parameter S0 = 3'b000,
12             S1 = 3'b001,
13             S2 = 3'b010,
14             S3 = 3'b011,
15             S4 = 3'b100,
16
17   reg [2:0] nextState;
18
19   always @ ( * ) begin
20     case (state)
21       S0: begin
22         if (Right)
23             nextState = S1;
24         else
25             nextState = S0;
26         end
27       S1: begin
28         if (Left)
29         // If digit is 13
30             if(Count == 5'b01101)
31                 nextState = S2;
32             else
```

```
33                  nextState = S0;
34              else
35                  nextState = S1;
36          end
37        S2: begin
38          if (Right)
39          // If digit is 7
40              if (Count == 5'b00111)
41                  nextState = S3;
42              else
43                  nextState = S0;
44          else
45              nextState = S2;
46          end
47        S3: begin
48          if (Left)
49          // If digit is 17
50              if (Count == 5'b10001)
51                  nextState = S4;
52              else
53                  nextState = S0;
54          else
55              nextState = S3;
56          end
57        S4: begin
58          nextState = S4;
59          end
60
61  default: begin
62  nextState = S0;
63  end
```

```
64      endcase
65    end
66      // If state S$ then unlock
67      assign Locked = (state == S4) ? 0:1;
68
69      always@ (posedge Clk)
70          if (South)
71              state <= S0;
72          else
73              state <= nextState;
74  endmodule // combination_lock_fsm
```

For the 2nd part of **Experiment 1**, the top level module was designed. The top level module is a 0-to-19 Up/Down Counter. This module keeps track of the position of the rotary knob located on the FBGA board. The module below was designed with behavioral Verilog and then tested against the appropriate test bench.

Code Block 2: Up/Down Counter

```
1  `timescale 1ns / 1ps
2
3  module up_down_counter(
4    output reg [4:0] Count,
5    input wire Up, Down,
6    input wire Clk, South
7    );
8
9    always @ (posedge Clk) begin
10 // If south button press, reset count
11      if (South)
12        Count <= 0;
13      else if (Up)
14        begin
```

```
15      //If rotating up add 1, if 19, reset
16        if (Count == 19)
17          Count <= 0;
18        else
19          Count <= Count +1;
20        end
21      else if (Down)
22      //If rotating down subtract 1, if 0, go to 19
23        if (Count == 0)
24          Count <= 19;
25        else
26          Count <= Count - 1;
27  end
28
29  endmodule // up_down_counter
```

Experiment 2

In the next part of the lab, the previously simulated modules were integrated with a given rotary encoder and LCD driver modules into a top-level module. The rotary combination lock module was set as the top level module. The other modules are as follows: "rotary_combination_lock.ucf" (the UCF for the top-level module), "synchronizer.v" (the synchronizer module for the asynchronous inputs), "lcd_driver.v" (the driver module for the character LCD screen), and "rotary_encoder_module.v" (the quadrature decoding module). These modules are below.

Code Block 3: Rotary Combination Lock Top Level Module

```
1  /*This is the top-level module for our digital *
2   *rotary combination-lock based on the diagram *
3   *provide in the lab manual                    */
4
5  module rotary_combination_lock(
6      /*LCD interface wires make up our output!*/
```

```
7       output wire LCD_E, LCD_RW, LCD_RS,
8       output wire [3:0] SF_D,
9       /*Let's output state for debugging!*/
10      output wire [2:0] J1,
11      input Clk,
12      /*the buttons and rotary encoder outputs*
13       *provide input to our top-level circuit*/
14      input Center,
15      input South,
16      input wire rotA, rotB
17  );
18
19      /*intermediate nets*/
20      wire CenterSync, SouthSync;
21      wire Right, Left;
22      wire Locked;
23      wire [4:0] Count;
24
25      /*synchronize button inputs*/
26      synchronizer syncA(CenterSync, Center, Clk);
27      synchronizer syncB(SouthSync, South, Clk);
28
29      /*wire up rotary encoder module*/
30      rotary_encoder_module U0(
31              .Left(Left),
32              .Right(Right),
33              .Clk(Clk),
34              .rotA(rotA),
35              .rotB(rotB)
36      );
37
```

```
38      /*wire up combination lock FSM*/
39      combination_lock_fsm U1(
40            .Locked(Locked),
41            .Right(Right),
42            .state(J1),
43            .Left(Left),
44            .Center(CenterSync),
45            .Clk(Clk),
46            .South(SouthSync),
47            .Count(Count)
48      );
49
50      /*instantiate up down counter*/
51      up_down_counter U2(
52            .Count(Count),
53            .Up (Left),
54            .Down(Right),
55            .Clk(Clk),
56            .South(South)
57      );
58
59      /*hook up LCD driver*/
60      lcd_driver U3(Clk, South, Count, Locked, SF_D, LCD_E,
61          LCD_RS, LCD_RW);
62
63  endmodule
```

Code Block 4: Rotary Combination Lock UCF File

```
1  #push buttons
2  NET "South"  LOC = "K17"  | IOSTANDARD = LVTTL | PULLDOWN ;
3  NET "Center" LOC = "V16"  | IOSTANDARD = LVTTL | PULLDOWN ;
```

```
4
5  #rotary encoder
6  NET "rotA" LOC = "K18" | IOSTANDARD = LVTTL | PULLUP ;
7  NET "rotB" LOC = "G18" | IOSTANDARD = LVTTL | PULLUP ;
8
9  #J1 connector for debugging!
10 NET "J1<0>" LOC = "B4" | IOSTANDARD = LVTTL | SLEW = SLOW
11       | DRIVE = 6 ;
12 NET "J1<1>" LOC = "A4" | IOSTANDARD = LVTTL | SLEW = SLOW
13       | DRIVE = 6 ;
14 NET "J1<2>" LOC = "D5" | IOSTANDARD = LVTTL | SLEW = SLOW
15       | DRIVE = 6 ;
16
17 #Clock
18 NET "Clk"  LOC = "C9"  ;
19 NET "Clk" PERIOD = 20.0ns HIGH 40%;
20
21 #LCD interface signals
22 NET "LCD_E" LOC = "M18" | IOSTANDARD = LVCMOS33 | DRIVE = 4
23       | SLEW = SLOW ;
24 NET "LCD_RS" LOC = "L18" | IOSTANDARD = LVCMOS33 | DRIVE = 4
25       | SLEW = SLOW ;
26 NET "LCD_RW" LOC = "L17" | IOSTANDARD = LVCMOS33 | DRIVE = 4
27       | SLEW = SLOW ;
28 #The LCD four-bit data interface is shared with the StrataFlash
29 NET "SF_D[0]" LOC = "R15" | IOSTANDARD = LVCMOS33 | DRIVE = 4
30       | SLEW = SLOW ;
31 NET "SF_D[1]" LOC = "R16" | IOSTANDARD = LVCMOS33 | DRIVE = 4
32       | SLEW = SLOW ;
33 NET "SF_D[2]" LOC = "P17" | IOSTANDARD = LVCMOS33 | DRIVE = 4
34       | SLEW = SLOW ;
```

```
35 NET "SF_D[3]" LOC = "M15" | IOSTANDARD = LVCMOS33 | DRIVE = 4
36     | SLEW = SLOW ;
```

Code Block 5: Synchronizer Module

```
1  /*This module provides the synchronization   *
2   *necessary to prevent metastability when    *
3   *transitioning from an asynchronous to a    *
4   *synchronous domain. In other words, when   *
5   *we bring an input signal in from the FPGA  *
6   *board into a clocked domain, we must do    *
7   *this buffering!                            */
8
9  module synchronizer(
10     output wire OutSignal,
11     input wire InSignal,
12     input wire Clk
13 );
14
15     /*intermediate nets*/
16     reg buff0, buff1, buff2;
17
18     always@(posedge Clk)
19             begin
20                 buff0 <= InSignal;
21                 buff1 <= buff0;
22                 buff2  <= buff1;
23             end
24
25     assign OutSignal = buff2;
26
27 endmodule
```

Code Block 6: LCD Driver Module

```
1  `timescale 1ns / 1ps
2  /*this module generates the interface signal necessary to*
3   *communicate with the character LCD on the Spartan 3e    *
4   *This code was modified from earlier semesters of 248    */
5
6  module lcd_driver(clk, reset, Position, Locked, SF_D, LCD_E,
7          LCD_RS, LCD_RW);
8  input [4:0] Position; //position of rotary knob
9  input Locked; //output of rotary combination FSM
10 input clk, reset;
11
12 /*needed at end of code to construct display output*/
13 reg [63:0] status_ascii; //8 characters for LOCKED or UNLOCKED
14 reg [15:0] count_ascii; //2 characters for decimal value of
15 // count
16
17 //This is ****** code but I don't have time to fix it right now
18 output [3:0] SF_D;
19 reg [3:0] SF_D=4'd0;
20 output LCD_E, LCD_RS, LCD_RW;
21 reg LCD_E=1'b0;
22 reg LCD_RS=1'b0;
23 reg LCD_RW=1'b0;
24 reg [20:0] count=21'd0;
25 reg [7:0] d=8'd0;
26 wire [255:0] display_data;
27
28 //Sequential Logic to generate Timing
```

```
29 always @ (posedge clk)
30 begin
31 case (reset)
32 1'b1:
33 begin
34         count <= count + 21'd1;
35         if ((count == 21'd750000) && (d==8'd0))
36         begin
37                 d <= d + 8'd1;
38                 count <= 21'd0;
39                 SF_D <= 4'b0011;
40                 LCD_E <= 1'b1;
41         end
42         if ((count == 21'd20) && (d==8'd1))
43         begin
44                 d <= d + 8'd1;
45                 count <= 21'd0;
46                 SF_D <= 4'b0000;
47                 LCD_E <= 1'b0;
48         end
49         if ((count == 21'd205000) && (d==8'd2))
50         begin
51                 d <= d + 8'd1;
52                 count <= 21'd0;
53                 SF_D <= 4'b0011;
54                 LCD_E <= 1'b1;
55         end
56         if ((count == 21'd20) && (d==8'd3))
57         begin
58                 d <= d + 8'd1;
59                 count <= 21'd0;
```

```
60                  SF_D <= 4'b0000;
61                  LCD_E <= 1'b0;
62          end
63          if ((count == 21'd5000) &&(d==8'd4))
64          begin
65                  d <= d + 8'd1;
66                  count <= 21'd0;
67                  SF_D <= 4'b0011;
68                  LCD_E <= 1'b1;
69          end
70          if ((count == 21'd20) && (d==8'd5))
71          begin
72                  d <= d + 8'd1;
73                  count <= 21'd0;
74                  SF_D <= 4'b0000;
75                  LCD_E <= 1'b0;
76          end
77          if ((count == 21'd2200) && (d==8'd6))
78          begin
79                  d <= d + 8'd1;
80                  count <= 21'd0;
81                  SF_D <= 4'b0010;
82                  LCD_E <= 1'b1;
83          end
84          if ((count == 21'd20) && (d==8'd7))
85          begin
86                  d <= d + 8'd1;
87                  count <= 21'd0;
88                  SF_D <= 4'b0000;
89                  LCD_E <= 1'b0;
90          end
```

```
91          //DISPLAY CONFIG
92          //Function Set Command
93          if ((count == 21'd2200) && (d==8'd8))
94          begin
95                  d <= d + 8'd1;
96                  count <= 21'd0;
97                  SF_D <= 4'b0010;
98                  LCD_E <= 1'b1;
99          end
100         if ((count == 21'd20) && (d==8'd9))
101         begin
102                 d <= d + 8'd1;
103                 count <= 21'd0;
104                 SF_D <= 4'b0000;
105                 LCD_E <= 1'b0;
106         end
107         if ((count == 21'd60) && (d==8'd10))
108         begin
109                 d <= d + 8'd1;
110                 count <= 21'd0;
111                 SF_D <= 4'b1000;
112                 LCD_E <= 1'b1;
113         end
114         if ((count == 21'd20) && (d==8'd11))
115         begin
116                 d <= d + 8'd1;
117                 count <= 21'd0;
118                 SF_D <= 4'b0000;
119                 LCD_E <= 1'b0;
120         end
121         //Entry Mode Set Command
```

```
122         if ((count == 21'd2200) && (d==8'd12))
123         begin
124                 d <= d + 8'd1;
125                 count <= 21'd0;
126                 SF_D <= 4'b0000;
127                 LCD_E <= 1'b1;
128         end
129         if ((count == 21'd20) && (d==8'd13))
130         begin
131                 d <= d + 8'd1;
132                 count <= 21'd0;
133                 SF_D <= 4'b0000;
134                 LCD_E <= 1'b0;
135         end
136         if ((count == 21'd60) && (d==8'd14))
137         begin
138                 d <= d + 8'd1;
139                 count <= 21'd0;
140                 SF_D <= 4'b0110;
141                 LCD_E <= 1'b1;
142         end
143         if ((count == 21'd20) && (d==8'd15))
144         begin
145                 d <= d + 8'd1;
146                 count <= 21'd0;
147                 SF_D <= 4'b0000;
148                 LCD_E <= 1'b0;
149         end
150         //Display ON/OFF command
151         if ((count == 21'd2200) && (d==8'd16))
152         begin
```

```
153                 d <= d + 8'd1;
154                 count <= 21'd0;
155                 SF_D <= 4'b0000;
156                 LCD_E <= 1'b1;
157         end
158         if ((count == 21'd20) && (d==8'd17))
159         begin
160                 d <= d + 8'd1;
161                 count <= 21'd0;
162                 SF_D <= 4'b0000;
163                 LCD_E <= 1'b0;
164         end
165         if ((count == 21'd60) && (d==8'd18))
166         begin
167                 d <= d + 8'd1;
168                 count <= 21'd0;
169                 SF_D <= 4'b1111;
170                 LCD_E <= 1'b1;
171         end
172         if ((count == 21'd20) && (d==8'd19))
173         begin
174                 d <= d + 8'd1;
175                 count <= 21'd0;
176                 SF_D <= 4'b0000;
177                 LCD_E <= 1'b0;
178         end
179         //Clear Display
180         if ((count == 21'd2200) && (d==8'd20))
181         begin
182                 d <= d + 8'd1;
183                 count <= 21'd0;
```

```
184                 SF_D <= 4'b0000;
185                 LCD_E <= 1'b1;
186         end
187         if ((count == 21'd20) && (d==8'd21))
188         begin
189                 d <= d + 8'd1;
190                 count <= 21'd0;
191                 SF_D <= 4'b0000;
192                 LCD_E <= 1'b0;
193         end
194         if ((count == 21'd60) && (d==8'd22))
195         begin
196                 d <= d + 8'd1;
197                 count <= 21'd0;
198                 SF_D <= 4'b0001;
199                 LCD_E <= 1'b1;
200         end
201         if ((count == 21'd20) && (d==8'd23))
202         begin
203                 d <= d + 8'd1;
204                 count <= 21'd0;
205                 SF_D <= 4'b0000;
206                 LCD_E <= 1'b0;
207         end
208         if ((count == 21'd80000) && (d==8'd24))
209         begin
210                 d <= d + 8'd1;
211                 count <= 21'd0;
212                 SF_D <= 4'b0000;
213                 LCD_E <= 1'b0;
214         end
```

```
215 end
216
217 1'b0:
218 begin
219         count <= count + 21'd1;
220         //Write Address and Data
221         //Write Initial Address
222         if ((count == 21'd2000) && (d==8'd25))
223         begin
224                 d <= d + 8'd1;
225                 count <= 21'd0;
226                 SF_D <= 4'b1000;
227                 LCD_E <= 1'b1;
228                 LCD_RS <= 1'b0;
229                 LCD_RW <=1'b0;
230         end
231         if ((count == 21'd20) && (d==8'd26))
232         begin
233                 d <= d + 8'd1;
234                 count <= 21'd0;
235                 SF_D <= 4'b0000;
236                 LCD_E <= 1'b0;
237                 LCD_RS <= 1'b0;
238                 LCD_RW <=1'b0;
239         end
240         if ((count == 21'd60) && (d==8'd27))
241         begin
242                 d <= d + 8'd1;
243                 count <= 21'd0;
244                 SF_D <= 4'b0000;
245                 LCD_E <= 1'b1;
```

```
246                 LCD_RS <= 1'b0;
247                 LCD_RW <=1'b0;
248             end
249             if ((count == 21'd20) && (d==8'd28))
250             begin
251                 d <= d + 8'd1;
252                 count <= 21'd0;
253                 SF_D <= 4'b0000;
254                 LCD_E <= 1'b0;
255             end
256             //Write Data=1 on each first space
257             if ((count == 21'd2200) && (d==8'd29))
258             begin
259                 d <= d + 8'd1;
260                 count <= 21'd0;
261                 SF_D <= display_data[255:252];
262                 LCD_E <= 1'b1;
263                 LCD_RS <= 1'b1;
264                 LCD_RW <=1'b0;
265
266             end
267             if ((count == 21'd20) && (d==8'd30))
268             begin
269                 d <= d + 8'd1;
270                 count <= 21'd0;
271                 SF_D <= 4'b0000;
272                 LCD_E <= 1'b0;
273             end
274             if ((count == 21'd60) && (d==8'd31))
275             begin
276                 d <= d + 8'd1;
```

```
277                 count <= 21'd0;
278                 SF_D <= display_data[251:248];
279                 LCD_E <= 1'b1;
280                 LCD_RS <= 1'b1;
281                 LCD_RW <=1'b0;
282         end
283         if ((count == 21'd20) && (d==8'd32))
284         begin
285                 d <= d + 8'd1;
286                 count <= 21'd0;
287                 SF_D <= 4'b0000;
288                 LCD_E <= 1'b0;
289         end
290         //Data=2
291         if ((count == 21'd2200) && (d==8'd33))
292         begin
293                 d <= d + 8'd1;
294                 count <= 21'd0;
295                 SF_D <= display_data[247:244];
296                 LCD_E <= 1'b1;
297                 LCD_RS <= 1'b1;
298                 LCD_RW <=1'b0;
299
300         end
301         if ((count == 21'd20) && (d==8'd34))
302         begin
303                 d <= d + 8'd1;
304                 count <= 21'd0;
305                 SF_D <= 4'b0000;
306                 LCD_E <= 1'b0;
307         end
```

```
308         if ((count == 21'd60) && (d==8'd35))
309         begin
310                 d <= d + 8'd1;
311                 count <= 21'd0;
312                 SF_D <= display_data[243:240];
313                 LCD_E <= 1'b1;
314                 LCD_RS <= 1'b1;
315                 LCD_RW <=1'b0;
316         end
317         if ((count == 21'd20) && (d==8'd36))
318         begin
319                 d <= d + 8'd1;
320                 count <= 21'd0;
321                 SF_D <= 4'b0000;
322                 LCD_E <= 1'b0;
323         end
324         //Data=3
325         if ((count == 21'd2200) && (d==8'd37))
326         begin
327                 d <= d + 8'd1;
328                 count <= 21'd0;
329                 SF_D <= display_data[239:236];
330                 LCD_E <= 1'b1;
331                 LCD_RS <= 1'b1;
332                 LCD_RW <=1'b0;
333
334         end
335         if ((count == 21'd20) && (d==8'd38))
336         begin
337                 d <= d + 8'd1;
338                 count <= 21'd0;
```

```
339                 SF_D <= 4'b0000;
340                 LCD_E <= 1'b0;
341         end
342         if ((count == 21'd60) && (d==8'd39))
343         begin
344                 d <= d + 8'd1;
345                 count <= 21'd0;
346                 SF_D <= display_data[235:232];
347                 LCD_E <= 1'b1;
348                 LCD_RS <= 1'b1;
349                 LCD_RW <=1'b0;
350         end
351         if ((count == 21'd20) && (d==8'd40))
352         begin
353                 d <= d + 8'd1;
354                 count <= 21'd0;
355                 SF_D <= 4'b0000;
356                 LCD_E <= 1'b0;
357         end
358         //Data=4
359         if ((count == 21'd2200) && (d==8'd41))
360         begin
361                 d <= d + 8'd1;
362                 count <= 21'd0;
363                 SF_D <= display_data[231:228];
364                 LCD_E <= 1'b1;
365                 LCD_RS <= 1'b1;
366                 LCD_RW <=1'b0;
367
368         end
369         if ((count == 21'd20) && (d==8'd42))
```

```
370             begin
371                     d <= d + 8'd1;
372                     count <= 21'd0;
373                     SF_D <= 4'b0000;
374                     LCD_E <= 1'b0;
375             end
376             if ((count == 21'd60) && (d==8'd43))
377             begin
378                     d <= d + 8'd1;
379                     count <= 21'd0;
380                     SF_D <= display_data[227:224];
381                     LCD_E <= 1'b1;
382                     LCD_RS <= 1'b1;
383                     LCD_RW <=1'b0;
384             end
385             if ((count == 21'd20) && (d==8'd44))
386             begin
387                     d <= d + 8'd1;
388                     count <= 21'd0;
389                     SF_D <= 4'b0000;
390                     LCD_E <= 1'b0;
391             end
392             //Data=5
393             if ((count == 21'd2200) && (d==8'd45))
394             begin
395                     d <= d + 8'd1;
396                     count <= 21'd0;
397                     SF_D <= display_data[223:220];
398                     LCD_E <= 1'b1;
399                     LCD_RS <= 1'b1;
400                     LCD_RW <=1'b0;
```

```
401
402             end
403             if ((count == 21'd20) && (d==8'd46))
404             begin
405                     d <= d + 8'd1;
406                     count <= 21'd0;
407                     SF_D <= 4'b0000;
408                     LCD_E <= 1'b0;
409             end
410             if ((count == 21'd60) && (d==8'd47))
411             begin
412                     d <= d + 8'd1;
413                     count <= 21'd0;
414                     SF_D <= display_data[219:216];
415                     LCD_E <= 1'b1;
416                     LCD_RS <= 1'b1;
417                     LCD_RW <=1'b0;
418             end
419             if ((count == 21'd20) && (d==8'd48))
420             begin
421                     d <= d + 8'd1;
422                     count <= 21'd0;
423                     SF_D <= 4'b0000;
424                     LCD_E <= 1'b0;
425             end
426             //Data=6
427             if ((count == 21'd2200) && (d==8'd49))
428             begin
429                     d <= d + 8'd1;
430                     count <= 21'd0;
431                     SF_D <= display_data[215:212];
```

```
432                 LCD_E <= 1'b1;
433                 LCD_RS <= 1'b1;
434                 LCD_RW <=1'b0;
435
436         end
437         if ((count == 21'd20) && (d==8'd50))
438         begin
439                 d <= d + 8'd1;
440                 count <= 21'd0;
441                 SF_D <= 4'b0000;
442                 LCD_E <= 1'b0;
443         end
444         if ((count == 21'd60) && (d==8'd51))
445         begin
446                 d <= d + 8'd1;
447                 count <= 21'd0;
448                 SF_D <= display_data[211:208];
449                 LCD_E <= 1'b1;
450                 LCD_RS <= 1'b1;
451                 LCD_RW <=1'b0;
452         end
453         if ((count == 21'd20) && (d==8'd52))
454         begin
455                 d <= d + 8'd1;
456                 count <= 21'd0;
457                 SF_D <= 4'b0000;
458                 LCD_E <= 1'b0;
459         end
460         //Data=7
461         if ((count == 21'd2200) && (d==8'd53))
462         begin
```

```
463                 d <= d + 8'd1;
464                 count <= 21'd0;
465                 SF_D <= display_data[207:204];
466                 LCD_E <= 1'b1;
467                 LCD_RS <= 1'b1;
468                 LCD_RW <=1'b0;
469
470         end
471         if ((count == 21'd20) && (d==8'd54))
472         begin
473                 d <= d + 8'd1;
474                 count <= 21'd0;
475                 SF_D <= 4'b0000;
476                 LCD_E <= 1'b0;
477         end
478         if ((count == 21'd60) && (d==8'd55))
479         begin
480                 d <= d + 8'd1;
481                 count <= 21'd0;
482                 SF_D <= display_data[203:200];
483                 LCD_E <= 1'b1;
484                 LCD_RS <= 1'b1;
485                 LCD_RW <=1'b0;
486         end
487         if ((count == 21'd20) && (d==8'd56))
488         begin
489                 d <= d + 8'd1;
490                 count <= 21'd0;
491                 SF_D <= 4'b0000;
492                 LCD_E <= 1'b0;
493         end
```

```
494         //Data=8
495         if ((count == 21'd2200) && (d==8'd57))
496         begin
497                 d <= d + 8'd1;
498                 count <= 21'd0;
499                 SF_D <= display_data[199:196];
500                 LCD_E <= 1'b1;
501                 LCD_RS <= 1'b1;
502                 LCD_RW <=1'b0;
503
504         end
505         if ((count == 21'd20) && (d==8'd58))
506         begin
507                 d <= d + 8'd1;
508                 count <= 21'd0;
509                 SF_D <= 4'b0000;
510                 LCD_E <= 1'b0;
511         end
512         if ((count == 21'd60) && (d==8'd59))
513         begin
514                 d <= d + 8'd1;
515                 count <= 21'd0;
516                 SF_D <= display_data[195:192];
517                 LCD_E <= 1'b1;
518                 LCD_RS <= 1'b1;
519                 LCD_RW <=1'b0;
520         end
521         if ((count == 21'd20) && (d==8'd60))
522         begin
523                 d <= d + 8'd1;
524                 count <= 21'd0;
```

```
525                 SF_D <= 4'b0000;
526                 LCD_E <= 1'b0;
527         end
528         //Data=9
529         if ((count == 21'd2200) && (d==8'd61))
530         begin
531                 d <= d + 8'd1;
532                 count <= 21'd0;
533                 SF_D <= display_data[191:188];
534                 LCD_E <= 1'b1;
535                 LCD_RS <= 1'b1;
536                 LCD_RW <=1'b0;
537
538         end
539         if ((count == 21'd20) && (d==8'd62))
540         begin
541                 d <= d + 8'd1;
542                 count <= 21'd0;
543                 SF_D <= 4'b0000;
544                 LCD_E <= 1'b0;
545         end
546         if ((count == 21'd60) && (d==8'd63))
547         begin
548                 d <= d + 8'd1;
549                 count <= 21'd0;
550                 SF_D <= display_data[187:184];
551                 LCD_E <= 1'b1;
552                 LCD_RS <= 1'b1;
553                 LCD_RW <=1'b0;
554         end
555         if ((count == 21'd20) && (d==8'd64))
```

```
556         begin
557                 d <= d + 8'd1;
558                 count <= 21'd0;
559                 SF_D <= 4'b0000;
560                 LCD_E <= 1'b0;
561         end
562         //Data=10
563         if ((count == 21'd2200) && (d==8'd65))
564         begin
565                 d <= d + 8'd1;
566                 count <= 21'd0;
567                 SF_D <= display_data[183:180];
568                 LCD_E <= 1'b1;
569                 LCD_RS <= 1'b1;
570                 LCD_RW <=1'b0;
571
572         end
573         if ((count == 21'd20) && (d==8'd66))
574         begin
575                 d <= d + 8'd1;
576                 count <= 21'd0;
577                 SF_D <= 4'b0000;
578                 LCD_E <= 1'b0;
579         end
580         if ((count == 21'd60) && (d==8'd67))
581         begin
582                 d <= d + 8'd1;
583                 count <= 21'd0;
584                 SF_D <= display_data[179:176];
585                 LCD_E <= 1'b1;
586                 LCD_RS <= 1'b1;
```

```
587                 LCD_RW <=1'b0;
588             end
589             if ((count == 21'd20) && (d==8'd68))
590             begin
591                 d <= d + 8'd1;
592                 count <= 21'd0;
593                 SF_D <= 4'b0000;
594                 LCD_E <= 1'b0;
595             end
596             //Data=11
597             if ((count == 21'd2200) && (d==8'd69))
598             begin
599                 d <= d + 8'd1;
600                 count <= 21'd0;
601                 SF_D <= display_data[175:172];
602                 LCD_E <= 1'b1;
603                 LCD_RS <= 1'b1;
604                 LCD_RW <=1'b0;
605
606             end
607             if ((count == 21'd20) && (d==8'd70))
608             begin
609                 d <= d + 8'd1;
610                 count <= 21'd0;
611                 SF_D <= 4'b0000;
612                 LCD_E <= 1'b0;
613             end
614             if ((count == 21'd60) && (d==8'd71))
615             begin
616                 d <= d + 8'd1;
617                 count <= 21'd0;
```

```
618                 SF_D <= display_data[171:168];
619                 LCD_E <= 1'b1;
620                 LCD_RS <= 1'b1;
621                 LCD_RW <=1'b0;
622             end
623             if ((count == 21'd20) && (d==8'd72))
624             begin
625                 d <= d + 8'd1;
626                 count <= 21'd0;
627                 SF_D <= 4'b0000;
628                 LCD_E <= 1'b0;
629             end
630             //Data=12
631             if ((count == 21'd2200) && (d==8'd73))
632             begin
633                 d <= d + 8'd1;
634                 count <= 21'd0;
635                 SF_D <= display_data[167:164];
636                 LCD_E <= 1'b1;
637                 LCD_RS <= 1'b1;
638                 LCD_RW <=1'b0;
639
640             end
641             if ((count == 21'd20) && (d==8'd74))
642             begin
643                 d <= d + 8'd1;
644                 count <= 21'd0;
645                 SF_D <= 4'b0000;
646                 LCD_E <= 1'b0;
647             end
648             if ((count == 21'd60) && (d==8'd75))
```

```
649         begin
650                 d <= d + 8'd1;
651                 count <= 21'd0;
652                 SF_D <= display_data[163:160];
653                 LCD_E <= 1'b1;
654                 LCD_RS <= 1'b1;
655                 LCD_RW <=1'b0;
656         end
657         if ((count == 21'd20) && (d==8'd76))
658         begin
659                 d <= d + 8'd1;
660                 count <= 21'd0;
661                 SF_D <= 4'b0000;
662                 LCD_E <= 1'b0;
663         end
664         //Data=13
665         if ((count == 21'd2200) && (d==8'd77))
666         begin
667                 d <= d + 8'd1;
668                 count <= 21'd0;
669                 SF_D <= display_data[159:156];
670                 LCD_E <= 1'b1;
671                 LCD_RS <= 1'b1;
672                 LCD_RW <=1'b0;
673
674         end
675         if ((count == 21'd20) && (d==8'd78))
676         begin
677                 d <= d + 8'd1;
678                 count <= 21'd0;
679                 SF_D <= 4'b0000;
```

```
680                 LCD_E <= 1'b0;
681             end
682             if ((count == 21'd60) && (d==8'd79))
683             begin
684                 d <= d + 8'd1;
685                 count <= 21'd0;
686                 SF_D <= display_data[155:152];
687                 LCD_E <= 1'b1;
688                 LCD_RS <= 1'b1;
689                 LCD_RW <=1'b0;
690             end
691             if ((count == 21'd20) && (d==8'd80))
692             begin
693                 d <= d + 8'd1;
694                 count <= 21'd0;
695                 SF_D <= 4'b0000;
696                 LCD_E <= 1'b0;
697             end
698             //Data=14
699             if ((count == 21'd2200) && (d==8'd81))
700             begin
701                 d <= d + 8'd1;
702                 count <= 21'd0;
703                 SF_D <= display_data[151:148];
704                 LCD_E <= 1'b1;
705                 LCD_RS <= 1'b1;
706                 LCD_RW <=1'b0;
707
708             end
709             if ((count == 21'd20) && (d==8'd82))
710             begin
```

```
711                 d <= d + 8'd1;
712                 count <= 21'd0;
713                 SF_D <= 4'b0000;
714                 LCD_E <= 1'b0;
715         end
716         if ((count == 21'd60) && (d==8'd83))
717         begin
718                 d <= d + 8'd1;
719                 count <= 21'd0;
720                 SF_D <= display_data[147:144];
721                 LCD_E <= 1'b1;
722                 LCD_RS <= 1'b1;
723                 LCD_RW <=1'b0;
724         end
725         if ((count == 21'd20) && (d==8'd84))
726         begin
727                 d <= d + 8'd1;
728                 count <= 21'd0;
729                 SF_D <= 4'b0000;
730                 LCD_E <= 1'b0;
731         end
732         //Data=15
733         if ((count == 21'd2200) && (d==8'd85))
734         begin
735                 d <= d + 8'd1;
736                 count <= 21'd0;
737                 SF_D <= display_data[143:140];
738                 LCD_E <= 1'b1;
739                 LCD_RS <= 1'b1;
740                 LCD_RW <=1'b0;
741
```

```
742             end
743             if ((count == 21'd20) && (d==8'd86))
744             begin
745                     d <= d + 8'd1;
746                     count <= 21'd0;
747                     SF_D <= 4'b0000;
748                     LCD_E <= 1'b0;
749             end
750             if ((count == 21'd60) && (d==8'd87))
751             begin
752                     d <= d + 8'd1;
753                     count <= 21'd0;
754                     SF_D <= display_data[139:136];
755                     LCD_E <= 1'b1;
756                     LCD_RS <= 1'b1;
757                     LCD_RW <=1'b0;
758             end
759             if ((count == 21'd20) && (d==8'd88))
760             begin
761                     d <= d + 8'd1;
762                     count <= 21'd0;
763                     SF_D <= 4'b0000;
764                     LCD_E <= 1'b0;
765             end
766             //Data=16
767             if ((count == 21'd2200) && (d==8'd89))
768             begin
769                     d <= d + 8'd1;
770                     count <= 21'd0;
771                     SF_D <= display_data[135:132];
772                     LCD_E <= 1'b1;
```

```
773                 LCD_RS <= 1'b1;
774                 LCD_RW <=1'b0;
775
776         end
777         if ((count == 21'd20) && (d==8'd90))
778         begin
779                 d <= d + 8'd1;
780                 count <= 21'd0;
781                 SF_D <= 4'b0000;
782                 LCD_E <= 1'b0;
783         end
784         if ((count == 21'd60) && (d==8'd91))
785         begin
786                 d <= d + 8'd1;
787                 count <= 21'd0;
788                 SF_D <= display_data[131:128];
789                 LCD_E <= 1'b1;
790                 LCD_RS <= 1'b1;
791                 LCD_RW <=1'b0;
792         end
793         if ((count == 21'd20) && (d==8'd92))
794         begin
795                 d <= d + 8'd1;
796                 count <= 21'd0;
797                 SF_D <= 4'b0000;
798                 LCD_E <= 1'b0;
799         end
800         //Write 2nd Line Initial Address
801         if ((count == 21'd2000) && (d==8'd93))
802         begin
803                 d <= d + 8'd1;
```

```
804                 count <= 21'd0;
805                 SF_D <= 4'b1100;
806                 LCD_E <= 1'b1;
807                 LCD_RS <= 1'b0;
808                 LCD_RW <=1'b0;
809         end
810         if ((count == 21'd20) && (d==8'd94))
811         begin
812                 d <= d + 8'd1;
813                 count <= 21'd0;
814                 SF_D <= 4'b0000;
815                 LCD_E <= 1'b0;
816         end
817         if ((count == 21'd60) && (d==8'd95))
818         begin
819                 d <= d + 8'd1;
820                 count <= 21'd0;
821                 SF_D <= 4'b0000;
822                 LCD_E <= 1'b1;
823                 LCD_RS <= 1'b0;
824                 LCD_RW <=1'b0;
825         end
826         if ((count == 21'd20) && (d==8'd96))
827         begin
828                 d <= d + 8'd1;
829                 count <= 21'd0;
830                 SF_D <= 4'b0000;
831                 LCD_E <= 1'b0;
832         end
833         ///Data=17
834         if ((count == 21'd2200) && (d==8'd97))
```

```
835         begin
836                 d <= d + 8'd1;
837                 count <= 21'd0;
838                 SF_D <= display_data[127:124];
839                 LCD_E <= 1'b1;
840                 LCD_RS <= 1'b1;
841                 LCD_RW <=1'b0;
842
843         end
844         if ((count == 21'd20) && (d==8'd98))
845         begin
846                 d <= d + 8'd1;
847                 count <= 21'd0;
848                 SF_D <= 4'b0000;
849                 LCD_E <= 1'b0;
850         end
851         if ((count == 21'd60) && (d==8'd99))
852         begin
853                 d <= d + 8'd1;
854                 count <= 21'd0;
855                 SF_D <= display_data[123:120];
856                 LCD_E <= 1'b1;
857                 LCD_RS <= 1'b1;
858                 LCD_RW <=1'b0;
859         end
860         if ((count == 21'd20) && (d==8'd100))
861         begin
862                 d <= d + 8'd1;
863                 count <= 21'd0;
864                 SF_D <= 4'b0000;
865                 LCD_E <= 1'b0;
```

```
866         end
867         //Data=18
868         if ((count == 21'd2200) && (d==8'd101))
869         begin
870                 d <= d + 8'd1;
871                 count <= 21'd0;
872                 SF_D <= display_data[119:116];
873                 LCD_E <= 1'b1;
874                 LCD_RS <= 1'b1;
875                 LCD_RW <=1'b0;
876
877         end
878         if ((count == 21'd20) && (d==8'd102))
879         begin
880                 d <= d + 8'd1;
881                 count <= 21'd0;
882                 SF_D <= 4'b0000;
883                 LCD_E <= 1'b0;
884         end
885         if ((count == 21'd60) && (d==8'd103))
886         begin
887                 d <= d + 8'd1;
888                 count <= 21'd0;
889                 SF_D <= display_data[115:112];
890                 LCD_E <= 1'b1;
891                 LCD_RS <= 1'b1;
892                 LCD_RW <=1'b0;
893         end
894         if ((count == 21'd20) && (d==8'd104))
895         begin
896                 d <= d + 8'd1;
```

```
897                 count <= 21'd0;
898                 SF_D <= 4'b0000;
899                 LCD_E <= 1'b0;
900         end
901         //Data=19
902         if ((count == 21'd2200) && (d==8'd105))
903         begin
904                 d <= d + 8'd1;
905                 count <= 21'd0;
906                 SF_D <= display_data[111:108];
907                 LCD_E <= 1'b1;
908                 LCD_RS <= 1'b1;
909                 LCD_RW <=1'b0;
910
911         end
912         if ((count == 21'd20) && (d==8'd106))
913         begin
914                 d <= d + 8'd1;
915                 count <= 21'd0;
916                 SF_D <= 4'b0000;
917                 LCD_E <= 1'b0;
918         end
919         if ((count == 21'd60) && (d==8'd107))
920         begin
921                 d <= d + 8'd1;
922                 count <= 21'd0;
923                 SF_D <= display_data[107:104];
924                 LCD_E <= 1'b1;
925                 LCD_RS <= 1'b1;
926                 LCD_RW <=1'b0;
927         end
```

```
928         if ((count == 21'd20) && (d==8'd108))
929         begin
930                 d <= d + 8'd1;
931                 count <= 21'd0;
932                 SF_D <= 4'b0000;
933                 LCD_E <= 1'b0;
934         end
935         //Data=20
936         if ((count == 21'd2200) && (d==8'd109))
937         begin
938                 d <= d + 8'd1;
939                 count <= 21'd0;
940                 SF_D <= display_data[103:100];
941                 LCD_E <= 1'b1;
942                 LCD_RS <= 1'b1;
943                 LCD_RW <=1'b0;
944
945         end
946         if ((count == 21'd20) && (d==8'd110))
947         begin
948                 d <= d + 8'd1;
949                 count <= 21'd0;
950                 SF_D <= 4'b0000;
951                 LCD_E <= 1'b0;
952         end
953         if ((count == 21'd60) && (d==8'd111))
954         begin
955                 d <= d + 8'd1;
956                 count <= 21'd0;
957                 SF_D <= display_data[99:96];
958                 LCD_E <= 1'b1;
```

```
959                 LCD_RS <= 1'b1;
960                 LCD_RW <=1'b0;
961         end
962         if ((count == 21'd20) && (d==8'd112))
963         begin
964                 d <= d + 8'd1;
965                 count <= 21'd0;
966                 SF_D <= 4'b0000;
967                 LCD_E <= 1'b0;
968         end
969         //Data=21
970         if ((count == 21'd2200) && (d==8'd113))
971         begin
972                 d <= d + 8'd1;
973                 count <= 21'd0;
974                 SF_D <= display_data[95:92];
975                 LCD_E <= 1'b1;
976                 LCD_RS <= 1'b1;
977                 LCD_RW <=1'b0;
978
979         end
980         if ((count == 21'd20) && (d==8'd114))
981         begin
982                 d <= d + 8'd1;
983                 count <= 21'd0;
984                 SF_D <= 4'b0000;
985                 LCD_E <= 1'b0;
986         end
987         if ((count == 21'd60) && (d==8'd115))
988         begin
989                 d <= d + 8'd1;
```

```
990                 count <= 21'd0;
991                 SF_D <= display_data[91:88];
992                 LCD_E <= 1'b1;
993                 LCD_RS <= 1'b1;
994                 LCD_RW <=1'b0;
995         end
996         if ((count == 21'd20) && (d==8'd116))
997         begin
998                 d <= d + 8'd1;
999                 count <= 21'd0;
1000                SF_D <= 4'b0000;
1001                LCD_E <= 1'b0;
1002        end
1003        //Data=22
1004        if ((count == 21'd2200) && (d==8'd117))
1005        begin
1006                d <= d + 8'd1;
1007                count <= 21'd0;
1008                SF_D <= display_data[87:84];
1009                LCD_E <= 1'b1;
1010                LCD_RS <= 1'b1;
1011                LCD_RW <=1'b0;
1012
1013        end
1014        if ((count == 21'd20) && (d==8'd118))
1015        begin
1016                d <= d + 8'd1;
1017                count <= 21'd0;
1018                SF_D <= 4'b0000;
1019                LCD_E <= 1'b0;
1020        end
```

```
1021        if ((count == 21'd60) && (d==8'd119))
1022        begin
1023                d <= d + 8'd1;
1024                count <= 21'd0;
1025                SF_D <= display_data[83:80];
1026                LCD_E <= 1'b1;
1027                LCD_RS <= 1'b1;
1028                LCD_RW <=1'b0;
1029        end
1030        if ((count == 21'd20) && (d==8'd120))
1031        begin
1032                d <= d + 8'd1;
1033                count <= 21'd0;
1034                SF_D <= 4'b0000;
1035                LCD_E <= 1'b0;
1036        end
1037        //Data=23
1038        if ((count == 21'd2200) && (d==8'd121))
1039        begin
1040                d <= d + 8'd1;
1041                count <= 21'd0;
1042                SF_D <= display_data[79:76];
1043                LCD_E <= 1'b1;
1044                LCD_RS <= 1'b1;
1045                LCD_RW <=1'b0;
1046
1047        end
1048        if ((count == 21'd20) && (d==8'd122))
1049        begin
1050                d <= d + 8'd1;
1051                count <= 21'd0;
```

```
1052                SF_D <= 4'b0000;
1053                LCD_E <= 1'b0;
1054            end
1055            if ((count == 21'd60) && (d==8'd123))
1056            begin
1057                d <= d + 8'd1;
1058                count <= 21'd0;
1059                SF_D <= display_data[75:72];
1060                LCD_E <= 1'b1;
1061                LCD_RS <= 1'b1;
1062                LCD_RW <=1'b0;
1063            end
1064            if ((count == 21'd20) && (d==8'd124))
1065            begin
1066                d <= d + 8'd1;
1067                count <= 21'd0;
1068                SF_D <= 4'b0000;
1069                LCD_E <= 1'b0;
1070            end
1071            //Data=24
1072            if ((count == 21'd2200) && (d==8'd125))
1073            begin
1074                d <= d + 8'd1;
1075                count <= 21'd0;
1076                SF_D <= display_data[71:68];
1077                LCD_E <= 1'b1;
1078                LCD_RS <= 1'b1;
1079                LCD_RW <=1'b0;
1080
1081            end
1082            if ((count == 21'd20) && (d==8'd126))
```

```
1083            begin
1084                    d <= d + 8'd1;
1085                    count <= 21'd0;
1086                    SF_D <= 4'b0000;
1087                    LCD_E <= 1'b0;
1088            end
1089            if ((count == 21'd60) && (d==8'd127))
1090            begin
1091                    d <= d + 8'd1;
1092                    count <= 21'd0;
1093                    SF_D <= display_data[67:64];
1094                    LCD_E <= 1'b1;
1095                    LCD_RS <= 1'b1;
1096                    LCD_RW <=1'b0;
1097            end
1098            if ((count == 21'd20) && (d==8'd128))
1099            begin
1100                    d <= d + 8'd1;
1101                    count <= 21'd0;
1102                    SF_D <= 4'b0000;
1103                    LCD_E <= 1'b0;
1104            end
1105            //Data=25
1106            if ((count == 21'd2200) && (d==8'd129))
1107            begin
1108                    d <= d + 8'd1;
1109                    count <= 21'd0;
1110                    SF_D <= display_data[63:60];
1111                    LCD_E <= 1'b1;
1112                    LCD_RS <= 1'b1;
1113                    LCD_RW <=1'b0;
```

```
1114
1115            end
1116            if ((count == 21'd20) && (d==8'd130))
1117            begin
1118                    d <= d + 8'd1;
1119                    count <= 21'd0;
1120                    SF_D <= 4'b0000;
1121                    LCD_E <= 1'b0;
1122            end
1123            if ((count == 21'd60) && (d==8'd131))
1124            begin
1125                    d <= d + 8'd1;
1126                    count <= 21'd0;
1127                    SF_D <= display_data[59:56];
1128                    LCD_E <= 1'b1;
1129                    LCD_RS <= 1'b1;
1130                    LCD_RW <=1'b0;
1131            end
1132            if ((count == 21'd20) && (d==8'd132))
1133            begin
1134                    d <= d + 8'd1;
1135                    count <= 21'd0;
1136                    SF_D <= 4'b0000;
1137                    LCD_E <= 1'b0;
1138            end
1139            //Data=26
1140            if ((count == 21'd2200) && (d==8'd133))
1141            begin
1142                    d <= d + 8'd1;
1143                    count <= 21'd0;
1144                    SF_D <= display_data[55:52];
```

```
1145                LCD_E <= 1'b1;
1146                LCD_RS <= 1'b1;
1147                LCD_RW <=1'b0;
1148
1149        end
1150        if ((count == 21'd20) && (d==8'd134))
1151        begin
1152                d <= d + 8'd1;
1153                count <= 21'd0;
1154                SF_D <= 4'b0000;
1155                LCD_E <= 1'b0;
1156        end
1157        if ((count == 21'd60) && (d==8'd135))
1158        begin
1159                d <= d + 8'd1;
1160                count <= 21'd0;
1161                SF_D <= display_data[51:48];
1162                LCD_E <= 1'b1;
1163                LCD_RS <= 1'b1;
1164                LCD_RW <=1'b0;
1165        end
1166        if ((count == 21'd20) && (d==8'd136))
1167        begin
1168                d <= d + 8'd1;
1169                count <= 21'd0;
1170                SF_D <= 4'b0000;
1171                LCD_E <= 1'b0;
1172        end
1173        //Data=27
1174        if ((count == 21'd2200) && (d==8'd137))
1175        begin
```

```
1176                    d <= d + 8'd1;
1177                    count <= 21'd0;
1178                    SF_D <= display_data[47:44];
1179                    LCD_E <= 1'b1;
1180                    LCD_RS <= 1'b1;
1181                    LCD_RW <=1'b0;
1182
1183            end
1184            if ((count == 21'd20) && (d==8'd138))
1185            begin
1186                    d <= d + 8'd1;
1187                    count <= 21'd0;
1188                    SF_D <= 4'b0000;
1189                    LCD_E <= 1'b0;
1190            end
1191            if ((count == 21'd60) && (d==8'd139))
1192            begin
1193                    d <= d + 8'd1;
1194                    count <= 21'd0;
1195                    SF_D <= display_data[43:40];
1196                    LCD_E <= 1'b1;
1197                    LCD_RS <= 1'b1;
1198                    LCD_RW <=1'b0;
1199            end
1200            if ((count == 21'd20) && (d==8'd140))
1201            begin
1202                    d <= d + 8'd1;
1203                    count <= 21'd0;
1204                    SF_D <= 4'b0000;
1205                    LCD_E <= 1'b0;
1206            end
```

```
1207        //Data=28
1208        if ((count == 21'd2200) && (d==8'd141))
1209        begin
1210                d <= d + 8'd1;
1211                count <= 21'd0;
1212                SF_D <= display_data[39:36];
1213                LCD_E <= 1'b1;
1214                LCD_RS <= 1'b1;
1215                LCD_RW <=1'b0;
1216
1217        end
1218        if ((count == 21'd20) && (d==8'd142))
1219        begin
1220                d <= d + 8'd1;
1221                count <= 21'd0;
1222                SF_D <= 4'b0000;
1223                LCD_E <= 1'b0;
1224        end
1225        if ((count == 21'd60) && (d==8'd143))
1226        begin
1227                d <= d + 8'd1;
1228                count <= 21'd0;
1229                SF_D <= display_data[35:32];
1230                LCD_E <= 1'b1;
1231                LCD_RS <= 1'b1;
1232                LCD_RW <=1'b0;
1233        end
1234        if ((count == 21'd20) && (d==8'd144))
1235        begin
1236                d <= d + 8'd1;
1237                count <= 21'd0;
```

```
1238                SF_D <= 4'b0000;
1239                LCD_E <= 1'b0;
1240            end
1241            //Data=29
1242            if ((count == 21'd2200) && (d==8'd145))
1243            begin
1244                d <= d + 8'd1;
1245                count <= 21'd0;
1246                SF_D <= display_data[31:28];
1247                LCD_E <= 1'b1;
1248                LCD_RS <= 1'b1;
1249                LCD_RW <=1'b0;
1250
1251            end
1252            if ((count == 21'd20) && (d==8'd146))
1253            begin
1254                d <= d + 8'd1;
1255                count <= 21'd0;
1256                SF_D <= 4'b0000;
1257                LCD_E <= 1'b0;
1258            end
1259            if ((count == 21'd60) && (d==8'd147))
1260            begin
1261                d <= d + 8'd1;
1262                count <= 21'd0;
1263                SF_D <= display_data[27:24];
1264                LCD_E <= 1'b1;
1265                LCD_RS <= 1'b1;
1266                LCD_RW <=1'b0;
1267            end
1268            if ((count == 21'd20) && (d==8'd148))
```

```
1269            begin
1270                    d <= d + 8'd1;
1271                    count <= 21'd0;
1272                    SF_D <= 4'b0000;
1273                    LCD_E <= 1'b0;
1274            end
1275            //Data=30
1276            if ((count == 21'd2200) && (d==8'd149))
1277            begin
1278                    d <= d + 8'd1;
1279                    count <= 21'd0;
1280                    SF_D <= display_data[23:20];
1281                    LCD_E <= 1'b1;
1282                    LCD_RS <= 1'b1;
1283                    LCD_RW <=1'b0;
1284
1285            end
1286            if ((count == 21'd20) && (d==8'd150))
1287            begin
1288                    d <= d + 8'd1;
1289                    count <= 21'd0;
1290                    SF_D <= 4'b0000;
1291                    LCD_E <= 1'b0;
1292            end
1293            if ((count == 21'd60) && (d==8'd151))
1294            begin
1295                    d <= d + 8'd1;
1296                    count <= 21'd0;
1297                    SF_D <= display_data[19:16];
1298                    LCD_E <= 1'b1;
1299                    LCD_RS <= 1'b1;
```

```
1300                LCD_RW <=1'b0;
1301            end
1302            if ((count == 21'd20) && (d==8'd152))
1303            begin
1304                d <= d + 8'd1;
1305                count <= 21'd0;
1306                SF_D <= 4'b0000;
1307                LCD_E <= 1'b0;
1308            end
1309            //Data=31
1310            if ((count == 21'd2200) && (d==8'd153))
1311            begin
1312                d <= d + 8'd1;
1313                count <= 21'd0;
1314                SF_D <= display_data[15:12];
1315                LCD_E <= 1'b1;
1316                LCD_RS <= 1'b1;
1317                LCD_RW <=1'b0;
1318
1319            end
1320            if ((count == 21'd20) && (d==8'd154))
1321            begin
1322                d <= d + 8'd1;
1323                count <= 21'd0;
1324                SF_D <= 4'b0000;
1325                LCD_E <= 1'b0;
1326            end
1327            if ((count == 21'd60) && (d==8'd155))
1328            begin
1329                d <= d + 8'd1;
1330                count <= 21'd0;
```

```
1331                SF_D <= display_data[11:8];
1332                LCD_E <= 1'b1;
1333                LCD_RS <= 1'b1;
1334                LCD_RW <=1'b0;
1335            end
1336            if ((count == 21'd20) && (d==8'd156))
1337            begin
1338                d <= d + 8'd1;
1339                count <= 21'd0;
1340                SF_D <= 4'b0000;
1341                LCD_E <= 1'b0;
1342            end
1343            //Data=32
1344            if ((count == 21'd2200) && (d==8'd157))
1345            begin
1346                d <= d + 8'd1;
1347                count <= 21'd0;
1348                SF_D <= display_data[7:4];
1349                LCD_E <= 1'b1;
1350                LCD_RS <= 1'b1;
1351                LCD_RW <=1'b0;
1352
1353            end
1354            if ((count == 21'd20) && (d==8'd158))
1355            begin
1356                d <= d + 8'd1;
1357                count <= 21'd0;
1358                SF_D <= 4'b0000;
1359                LCD_E <= 1'b0;
1360            end
1361            if ((count == 21'd60) && (d==8'd159))
```

```
1362            begin
1363                    d <= d + 8'd1;
1364                    count <= 21'd0;
1365                    SF_D <= display_data[3:0];
1366                    LCD_E <= 1'b1;
1367                    LCD_RS <= 1'b1;
1368                    LCD_RW <=1'b0;
1369            end
1370            if ((count == 21'd20) && (d==8'd160))
1371            begin
1372                    d <= 8'd25;
1373                    count <= 21'd0;
1374                    SF_D <= 4'b0000;
1375                    LCD_E <= 1'b0;
1376            end
1377            //Wait for Next Data & Go back to Address Cycle
1378            //Wait for more than half a second here
1379
1380
1381    end
1382    endcase
1383    end
1384
1385    /*wow this has to be the worst Verilog code I have ever seen...*/
1386    /*okay my modifications start here... let's see how it goes...*/
1387
1388    /*Let's just keep is simple okay  so the following is an example
1389     *of the LCD output when locked with 7 dialed in
1390      >07  LOCKED
1391
1392     *Now here is what it should look like with 17 dialed in after
```

```
1393  *being unlocked
1394
1395  >17  UNLOCKED
1396
1397  *ASCII cheat sheet:
1398  * character  Hex Code
1399  * >          3e
1400  * 0          30
1401  * 1          31
1402  * 2          32
1403  * 3          33
1404  * 4          34
1405  * 5          35
1406  * 6          36
1407  * 7          37
1408  * 8          38
1409  * 9          39
1410  * U          55
1411  * N          4e
1412  * L          4c
1413  * O          4f
1414  * C          43
1415  * K          4b
1416  * E          45
1417  * D          44
1418  *SPACE       20
1419 */
1420
1421 /*status_ascii is the ascii representation of LOCKED or UNLOCKED
      */
1422 always@(*)
```

```
1423        if(Locked)//print LOCKED
1424            status_ascii = {8'h20, 8'h20, 8'h4c, 8'h4f, 8'h43,
1425                    8'h4b, 8'h45, 8'h44};
1426        else //print UNLOCKED
1427            status_ascii = {8'h55, 8'h4e, 8'h4c, 8'h4f, 8'h43,
1428                    8'h4b, 8'h45, 8'h44};
1429
1430    /*count_ascii is the ascii representation of the
1431            2 decimal digits of count*/
1432    always@(*)//giant encoder!
1433        case(Position)
1434            5'd0:  count_ascii = 16'h3030;
1435            5'd1:  count_ascii = 16'h3031;
1436            5'd2:  count_ascii = 16'h3032;
1437            5'd3:  count_ascii = 16'h3033;
1438            5'd4:  count_ascii = 16'h3034;
1439            5'd5:  count_ascii = 16'h3035;
1440            5'd6:  count_ascii = 16'h3036;
1441            5'd7:  count_ascii = 16'h3037;
1442            5'd8:  count_ascii = 16'h3038;
1443            5'd9:  count_ascii = 16'h3039;
1444            5'd10: count_ascii = 16'h3130;
1445            5'd11: count_ascii = 16'h3131;
1446            5'd12: count_ascii = 16'h3132;
1447            5'd13: count_ascii = 16'h3133;
1448            5'd14: count_ascii = 16'h3134;
1449            5'd15: count_ascii = 16'h3135;
1450            5'd16: count_ascii = 16'h3136;
1451            5'd17: count_ascii = 16'h3137;
1452            5'd18: count_ascii = 16'h3138;
1453            5'd19: count_ascii = 16'h3139;
```

```
1454            5'd20: count_ascii = 16'h3230;
1455            5'd21: count_ascii = 16'h3231;
1456            5'd22: count_ascii = 16'h3232;
1457            5'd23: count_ascii = 16'h3233;
1458            5'd24: count_ascii = 16'h3234;
1459            5'd25: count_ascii = 16'h3235;
1460            5'd26: count_ascii = 16'h3236;
1461            5'd27: count_ascii = 16'h3237;
1462            5'd28: count_ascii = 16'h3238;
1463            5'd29: count_ascii = 16'h3239;
1464            5'd30: count_ascii = 16'h3330;
1465            5'd31: count_ascii = 16'h3331;
1466        endcase
1467
1468    /*Display data is a 256 bit vector which hold 32 8-bit ascii *
1469     *characters which are to be displayed on the LCD screen     */
1470    assign display_data = {8'h3e, count_ascii, 8'h20,
1471            8'h20, status_ascii, {19{8'h20}}};
1472
1473    endmodule
```

Code Block 7: Rotary Encoder Module

```
1   `timescale 1 ns / 1 ps
2   `default_nettype none
3
4   /*This module takes as input the quadrature outputs *
5    *A and B from the rotary encoder on the Spartan 3e *
6    *board, filters out electrical chatter, and outputs*
7    *a Right and a Left signal. Left pulses every 18   *
8    *as the rotary shaft rotates to the left, while    *
9    *Right pulses every 18 degrees as the rotary shaft *
```

```
10  *rotates to the right.                                    *
11  *The technique describe here is provided by an            *
12  *an Xilinx application engineer in a document             *
13  *entitled "Rotary Encoder Interface for                   *
14  *Spartan-3E Starter Kit"                                  */
15  module rotary_encoder_module(
16      output wire Left, Right,
17      input Clk,
18      input wire rotA, rotB
19  );
20
21      /*internal nets*/
22      wire buffA, buffB; //input buffers
23      reg rotary_q1, rotary_q2;
24      reg rotary_event, rotary_left;
25      reg rotary_q1_old;
26
27
28      /*buffer inputs*/
29      synchronizer syncA(buffA, rotA, Clk);
30      synchronizer syncB(buffB, rotB, Clk);
31
32
33      /*elimentate bounce! The result is q1 and q2*/
34      /*XNOR*/
35      always@(posedge Clk)
36          if(buffA & buffB)
37              rotary_q1 <= 1'b1;
38          else if(~buffA & ~ buffB)
39              rotary_q1 <= 1'b0;
40      /*XOR*/
```

```
41        always@(posedge Clk)
42              if(~buffA & buffB)
43                    rotary_q2 <= 1'b1;
44              else if(buffA & ~buffB)
45                    rotary_q2 <= 1'b0;
46
47        /*detect rising edge on q1 to signal an event*/
48        always@(posedge Clk)
49              rotary_q1_old <= rotary_q1;
50
51        always@(posedge Clk)
52              rotary_event <= ~rotary_q1_old & rotary_q1;
53
54        /*determine the direction of rotation based on q1 and q2*/
55        always@(posedge Clk)
56              if(~rotary_q1_old & rotary_q1)
57                    rotary_left <= rotary_q2;
58
59        /*generate Left and Right signals*/
60        assign Left  = rotary_event & rotary_left;
61        assign Right = rotary_event & ~rotary_left;
62
63  endmodule
```

The modules were then synthesized onto the FBGA board and tested.

In order to add a 4^{th} digit (17), the following changes were made to the combination lock Verilog code.

Code Block 8: Combination Lock FSM

```
1  `timescale 1ns / 1ps
2
3  module combination_lock_fsm(output reg [2:0] state,
```

```
4   output wire Locked, // asserted when locked
5   input wire Right, Left, // indicate direction
6   input wire [4:0] Count, // indicate position
7   input wire Center, // the unlock button
8   input wire Clk, South // clock and reset
9   );
10  // Parameters for 5 cases
11  parameter S0 = 3'b000,
12            S1 = 3'b001,
13            S2 = 3'b010,
14            S3 = 3'b011,
15            S4 = 3'b100,
16            S5 = 3'b101;
17
18  reg [2:0] nextState;
19
20  always @ ( * ) begin
21    case (state)
22      S0: begin
23        if (Right)
24            nextState = S1;
25        else
26            nextState = S0;
27        end
28      S1: begin
29        if (Left)
30        // If digit is 13
31            if(Count == 5'b01101)
32                nextState = S2;
33            else
34                nextState = S0;
```

```
35          else
36              nextState = S1;
37          end
38        S2: begin
39          if (Right)
40          // If digit 7
41              if (Count == 5'b00111)
42                  nextState = S3;
43              else
44                  nextState = S0;
45          else
46              nextState = S2;
47          end
48        S3: begin
49          if (Left)
50          // If digit is 17
51              if (Count == 5'b10001)
52                  nextState = S4;
53              else
54                  nextState = S0;
55          else
56              nextState = S3;
57          end
58        S4: begin
59          if (Center)
60          // If digit is 17
61              if (Count == 5'b10001)
62                  nextState = S5;
63              else
64                  nextState = S0;
65              else if (Right)
```

```
66                      nextState = S0;
67              else
68                  nextState = S4;
69              end
70          S5: begin
71              nextState = S5;
72              end
73
74          default: begin
75              nextState = S0;
76          end
77          endcase
78      end
79
80          assign Locked = (state == S5) ? 0:1;
81
82          always@ (posedge Clk)
83              if (South)
84                  state <= S0;
85              else
86                  state <= nextState;
87  endmodule // combination_lock_fsm
```

This module was then synthesized, along with all the rest, onto the FBGA board.

Results

Experiment 1

Below are the results of the the tests as well as the waveform for the combination lock FSM Verilog design that was designed in the prelab and modified in lab.

Next are the test results and the waveform for the Up/Down Counter.

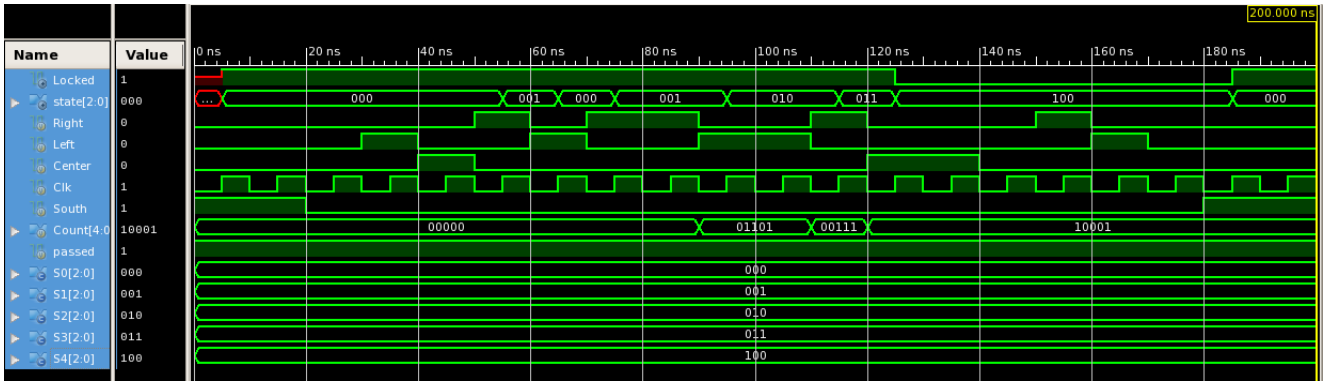

Figure 2: *Combination Lock FMS Waveform*

```
INFO:Security:50 - The XILINXD_LICENSE_FILE environment variable is set to '2100@165.91.159.26'.
INFO:Security:53 - The LM_LICENSE_FILE environment variable is not set.
WARNING:Security:43 - No license file was found in the standard Xilinx license directory.
WARNING:Security:44 - Since no license file was found,
    please run the Xilinx License Configuration Manager
    (xlcm or "Manage Xilinx Licenses")
    to assist in obtaining a license.
WARNING:Security:42 - Your software subscription period has lapsed. Your current version of Xilinx tools will continue to function, but you no longer qualify for Xilinx s

--------------------------------------------------------------------------------
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
All Tests Passed!!!
Stopped at time : 200 ns :  in File "/home/ugrads/j/josephmart/ecen248/lab11/code/combination_lock_fsm_tb.v" Line 218
```

Figure 3: *Combination Lock FMS Test Results*

Experiment 2

For the next part of the lab, the three number combination lock was tested. It passed all of the TA's tests with the code being 13-7-17 with the first digit being selected from a clockwise rotation.

The last part of the lab involved testing the four number combination lock. This design also passed all of the TA's tests with the code being 13-7-17-17.

Conclusion

In conclusion, this lab allowed me to learn more about Verilog and circuit design. This lab showed me how through the iteration of modules and through synthesizing, an

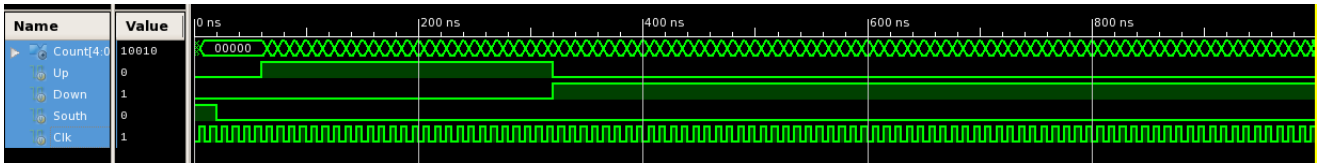

Figure 4: *Up/Down Counter Waveform*

```
INFO:Security:50 - The XILINXD_LICENSE_FILE environment variable is set to '2100@165.91.159.26'.
INFO:Security:53 - The LM_LICENSE_FILE environment variable is not set.
WARNING:Security:43 - No license file was found in the standard Xilinx license directory.
WARNING:Security:44 - Since no license file was found,
    please run the Xilinx License Configuration Manager
    (xlcm or "Manage Xilinx Licenses")
    to assist in obtaining a license.
WARNING:Security:42 - Your software subscription period has lapsed. Your current version of Xilinx tools will continue to function, but you no longer qualify for Xilinx software updates or new releases.

-----------------------------------------------------------------------
This is a Full version of ISim.
Time resolution is 1 ps
Simulator is doing circuit initialization process.
Finished circuit initialization process.
All tests passed!!!
```

Figure 5: *Up/Down Counter Test Results*

everyday use object (combination lock) can be created, with a little help, by a simple sophomore engineer.

Questions

1. **Include the source code with comments for all modules you simulated and/or implemented in lab. You do not have to include test bench code that was provided! Code without comments will not be accepted!**

   In the report

2. **Include screenshots of all waveforms captured during simulation in addition to the test bench console output for each test bench simulation.**

   In the report

3. **Answer all questions throughout the lab manual.**

   (a) **Likewise, take a look at the simulation waveform and take note of the tests that the test bench performs. Is this an exhaustive test?**

Why or why not?

The simulation waveform above does not display an exhaustive test because there are too many inputs and all of them are not taken into account. There are about $8,000$ different combinations. Also, it does not seem to take into account for the last digit, the case when the user rotates pasts the number and rotates all the way back around to the number.

4. **A possible attack on your combination-lock is a brute-force attack in which every possible input combination is tried. Given the original design with a combination of three numbers between 0 and 19, how many possible input combinations exist? How about for the modified design with a combination of four numbers?**

   For the combination lock with three numbers between 0 and 19, there are $20 \cdot 20 \cdot 20 = 20^3 = 8,000$ combinations. For the modified combination lock with four numbers ranging between 0 and 19, there are $20^4 = 160,000$ different combinations.

Student Feedback

1. **What did you like most about the lab assignment and why? What did you like least about it and why?**

   I really liked that we had to design a combination lock for this lab. There was nothing I disliked in this lab.

2. **Were there any section of the lab manual that were unclear? If so, what was unclear? Do you have any suggestions for improving the clarity?**

   No, the lab was very straightforward.

3. **What suggestions do you have to improve the overall lab assignment?**

   I really enjoyed this lab.