

Hotel Safe Box Lock

Eric Wu, 75286914

University of British Columbia

## Description

Inspired by my quarantine hotel safe box (Figure 1), I thought that implementing a hotel safe box lock would be a pretty interesting use for FSMs. However, to narrow the scope of the project, the HEX display was not implemented.



Figure 1.

The design is separated into two modules. The number button capturer and the locking FSM.

The number button capturer assumes number buttons are logic 0 when unpressed and logic 1 when pressed. It not only detects the edge of the button press, but also keeps the value of the number.

The locking FSM can be separated into three parts:

- Setting up a security code (for hotel owners)
- Setting up a customer code (for customer use)
- Unlocking/Resetting

The user interface would look something like this (Figure 2):

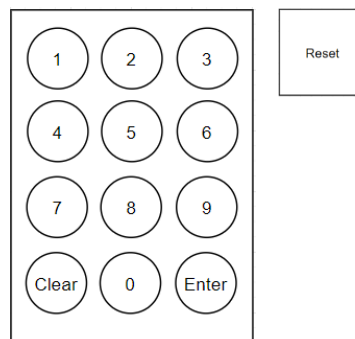


Figure 2.

## Block Diagrams

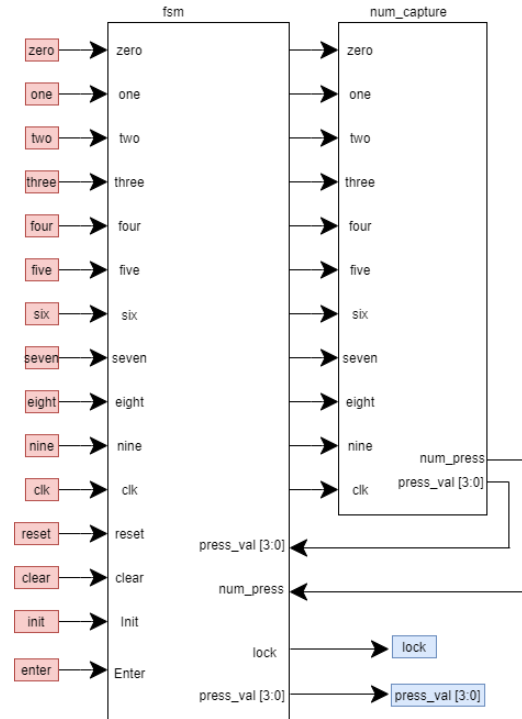


Figure 3. Block diagram of FSM modules (inputs are red, outputs are blue)

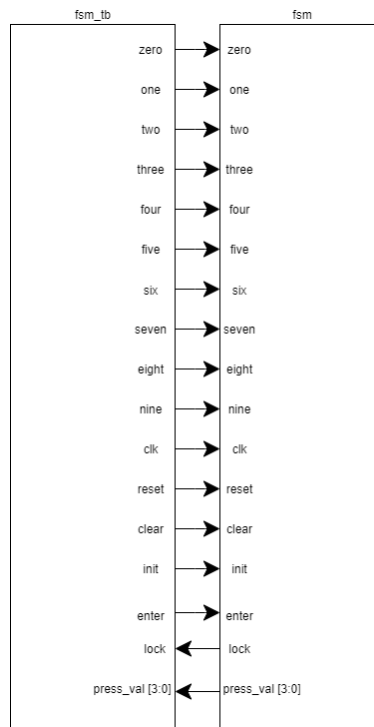


Figure 4. Block diagram of TB (fsm\_tb) and DUT (fsm)

### State diagram of FSM flow

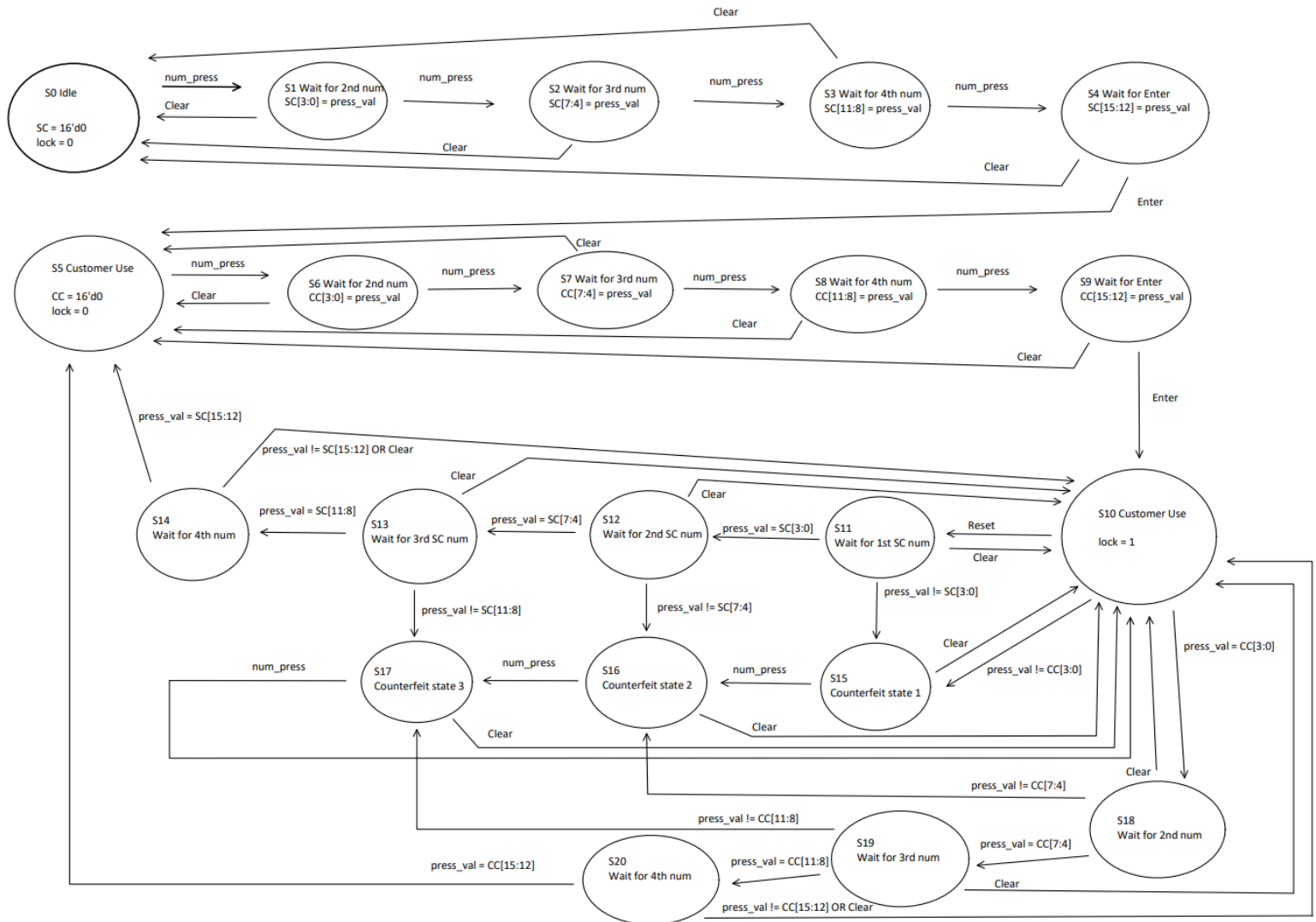


Figure 5. State diagram of FSM

S0-S4 sets up the security code, S5 is the state where it is ready to be used by the customer.

S5-S9 sets up the customer code, S10 is the state where it is locked.

S11-S14 is the reset process. When the customer forgets the customer code, it can be reset by entering the security code.

S15-S17 are counterfeit states. These states prevent logic loopholes when a HEX display is used. If pressing a wrong number goes directly back to S10, the HEX display will tell the person the number is wrong for that digit, meaning that the lock can be easily unlocked with at most 40 tries. With counterfeit states, it needs at most 10000 tries to be unlocked.

S18-S19 is the normal unlock process.

# SystemVerilog Modules

num\_capture:

```
1  /*
2  Number button capturer
3  (c) Copyright Eric Wu
4  All rights reserved.
5
6  Author: Eric Wu
7  Email: ew820120@gmail.com
8  Student ID: 75286914
9
10 */
11
12 /*
13 Description:
14 A number button capturer that detects button press and saves pressed value.
15
16 Inputs:
17 zero          - button signal for zero (pressed = 1, unpressed = 0)
18 one           - button signal for one (pressed = 1, unpressed = 0)
19 two           - button signal for two (pressed = 1, unpressed = 0)
20 three         - button signal for three (pressed = 1, unpressed = 0)
21 four          - button signal for four (pressed = 1, unpressed = 0)
22 five          - button signal for five (pressed = 1, unpressed = 0)
23 six           - button signal for six (pressed = 1, unpressed = 0)
24 seven         - button signal for seven (pressed = 1, unpressed = 0)
25 eight         - button signal for eight (pressed = 1, unpressed = 0)
26 nine         - button signal for nine (pressed = 1, unpressed = 0)
27 clk           - clock
28
29 Outputs:
30 press_val     - value of number pressed
31 num_press     - edge captured signal of button press
32 */
33
34
35
36 module num_capture(input logic zero, one, two, three, four, five, six, seven, eight, nine,
37                   input logic clk,
38                   output logic [3:0] press_val,
39                   output logic num_press);
40
41 /*
42 edge detector: detects key press and provides output edge captured signal num_press.
43 two inverters were used as delay for the output num_press, amount of delay will be configured after synthesis.
44 */
45 logic key_press;
46 logic delay_reg;
47 logic num_press_d0, num_press_d1;
48
49 assign key_press = zero | one | two | three | four | five | six | seven | eight | nine;
50
51 always_ff @(posedge clk)
52 delay_reg <= key_press;
53
54 assign num_press_d0 = ~delay_reg & key_press;
55 assign num_press_d1 = ~num_press_d0;
56 assign num_press = ~num_press_d1;
57
58
59 //gets number press value each key press
60 always_ff @(posedge key_press)
61 case({zero, one, two, three, four, five, six, seven, eight, nine})
62 10'b000000001: press_val = 4'd9;
63 10'b000000010: press_val = 4'd8;
64 10'b000000100: press_val = 4'd7;
65 10'b000001000: press_val = 4'd6;
66 10'b000010000: press_val = 4'd5;
67 10'b000100000: press_val = 4'd4;
68 10'b001000000: press_val = 4'd3;
69 10'b010000000: press_val = 4'd2;
70 10'b100000000: press_val = 4'd1;
71 10'b100000000: press_val = 4'd0;
72 default: press_val = 4'b1111; //idle
73 endcase
74
75 endmodule
```

fsm:

```
1  /*
2  Hotel safe box lock FSM
3  (c) Copyright Eric Wu
4  All rights reserved.
5
6  Author: Eric Wu
7  Email: ew820120@gmail.com
8  Student ID: 75286914
9  */
10
11 /*
12 Description:
13 A FSM that goes through three main stages.
14 - Setting up security code S0~S4
15 - Setting up customer code S5~S9
16 - Unlocking and Resetting S10~S20
17
18 Security code saved in sc[15:0]
19 Customer code saved in cc[15:0]
20
21 Inputs:
22   clk          - clock
23   zero         - button signal for zero (pressed = 1, unpressed = 0)
24   one          - button signal for one (pressed = 1, unpressed = 0)
25   two         - button signal for two (pressed = 1, unpressed = 0)
26   three        - button signal for three (pressed = 1, unpressed = 0)
27   four         - button signal for four (pressed = 1, unpressed = 0)
28   five         - button signal for five (pressed = 1, unpressed = 0)
29   six          - button signal for six (pressed = 1, unpressed = 0)
30   seven        - button signal for seven (pressed = 1, unpressed = 0)
31   eight        - button signal for eight (pressed = 1, unpressed = 0)
32   nine         - button signal for nine (pressed = 1, unpressed = 0)
33   enter        - button signal for enter (pressed = 1, unpressed = 0)
34   init         - signal for force initialization (state goes to S0)
35   reset        - button signal for resetting cc[15:0] (pressed = 1, unpressed = 0)
36   clear        - button signal for clearing previous entered numbers (pressed = 1, unpressed = 0)
37
38 Outputs:
39   press_val    - value of number pressed (for HEX display purposes, which is not included in this project)
40   lock         - lock signal, locked = 1, unlocked = 0 (controls the actual lock)
41
42 */
43
44 module fsm(input logic clk,
45            input logic zero, one, two, three, four, five, six, seven, eight, nine, enter,
46            input logic init, reset, clear,
47            output logic [3:0] press_val,
48            output logic lock);
49
50
51
52 //instantiation of num_capture
53 logic num_press;
54 num_capture asdf(.zero(zero), .one(one), .two(two), .three(three), .four(four), .five(five), .six(six), .seven(seven), .eight(eight), .nine(nine),
55                 .clk(clk), .press_val(press_val), .num_press(num_press));
56
57 //fsm state declaration and variables
58 typedef enum logic [4:0] {S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17,
59                          S18, S19, S20, S21, S22, S23, S24, S25, S26, S27, S28, S29, S30, S31} statetype;
60
61 statetype state, next_state;
62 logic [15:0] sc; //security code
63 logic [15:0] cc; //customer code
64
65 //state register
66 always_ff @(posedge clk)
67 if (init) state <= S0;
68 else state <= next_state;
69
70 //next state logic
71
72 always_comb
73 case(state)
74
75 //Security code setup
76 S0: begin
77   lock = 1'b0;
78   sc = 16'd0;
79   if (num_press) next_state = S1;
80   else next_state = S0;
81 end
```

```

83  □ /*
84  sc[3:0] = press_val is in if/else statement because press_val gets its new value the moment the number button is pressed while state
85  needs to wait for next posedge clk to go to next stage, the press_val meant for sc[7:4] would replace original sc[3:0] if it was outside if statement
86  */
87
88  □ S1: begin
89  if (clear) next_state = S0;
90  else if (num_press) next_state = S2;
91  □ else begin
92  next_state = S1;
93  sc[3:0] = press_val;
94  end
95  end
96
97  □ S2: begin
98  if (clear) next_state = S0;
99  else if (num_press) next_state = S3;
100 □ else begin
101 next_state = S2;
102 sc[7:4] = press_val;
103 end
104 end
105
106 □ S3: begin
107 if (clear) next_state = S0;
108 else if (num_press) next_state = S4;
109 □ else begin
110 next_state = S3;
111 sc[11:8] = press_val;
112 end
113 end
114
115 □ S4: begin
116 if (clear) next_state = S0;
117 else if (enter) next_state = S5;
118 □ else begin
119 next_state = S4;
120 sc[15:12] = press_val;
121 end
122 end

```

```

124 //Customer code setup
125 □ S5: begin
126 lock = 1'b0;
127 cc = 16'b0;
128 if (num_press) next_state = S6;
129 else next_state = S5;
130 end
131
132 □ S6: begin
133 if (clear) next_state = S5;
134 else if (num_press) next_state = S7;
135 □ else begin
136 next_state = S6;
137 cc[3:0] = press_val;
138 end
139 end
140
141 □ S7: begin
142 if (clear) next_state = S5;
143 else if (num_press) next_state = S8;
144 □ else begin
145 next_state = S7;
146 cc[7:4] = press_val;
147 end
148 end
149
150 □ S8: begin
151 if (clear) next_state = S5;
152 else if (num_press) next_state = S9;
153 □ else begin
154 next_state = S8;
155 cc[11:8] = press_val;
156 end
157 end
158
159 □ S9: begin
160 if (clear) next_state = S5;
161 else if (enter) next_state = S10;
162 □ else begin
163 next_state = S9;
164 cc[15:12] = press_val;
165 end
166 end

```

```

168 //Unlock and Reset states
169 S10: begin
170     lock = 1'b1;
171     if (reset) next_state = S11;
172 else if (num_press) begin
173     if (press_val == cc[3:0])
174         next_state = S18;
175     else
176         next_state = S15;
177     end
178 else next_state = S10;
179 end
180
181 S11: begin
182     if (clear) next_state = S10;
183 else if (num_press) begin
184     if (press_val == sc[3:0])
185         next_state = S12;
186     else next_state = S15;
187     end
188 else next_state = S11;
189 end
190
191 S12: begin
192     if (clear) next_state = S10;
193 else if (num_press) begin
194     if (press_val == sc[7:4])
195         next_state = S13;
196     else next_state = S16;
197     end
198 else next_state = S12;
199 end
200
201 S13: begin
202     if (clear) next_state = S10;
203 else if (num_press) begin
204     if (press_val == sc[11:8])
205         next_state = S14;
206     else next_state = S17;
207     end
208 else next_state = S13;
209 end

```

```

211 S14: begin
212     if (clear) next_state = S10;
213 else if (num_press) begin
214     if (press_val == sc[15:12])
215         next_state = S5;
216     else next_state = S10;
217     end
218 else next_state = S14;
219 end
220
221 S15:
222     if (clear)
223         next_state = S10;
224     else if (num_press)
225         next_state = S16;
226     else
227         next_state = S15;
228
229 S16:
230     if (clear)
231         next_state = S10;
232     else if (num_press)
233         next_state = S17;
234     else
235         next_state = S16;
236
237 S17:
238     if (clear)
239         next_state = S10;
240     else if (num_press)
241         next_state = S10;
242     else
243         next_state = S17;
244
245 S18: begin
246     if (clear) next_state = S10;
247 else if (num_press) begin
248     if (press_val == cc[7:4])
249         next_state = S19;
250     else next_state = S16;
251     end
252 else next_state = S18;
253 end
254

```



```
255 S19: begin
256   if (clear) next_state = S10;
257   else if (num_press) begin
258     if (press_val == cc[11:8])
259       next_state = S20;
260     else next_state = S17;
261   end
262   else next_state = S19;
263 end
264
265 S20: begin
266   if (clear) next_state = S10;
267   else if (num_press) begin
268     if (press_val == cc[15:12])
269       next_state = S5;
270     else next_state = S10;
271   end
272   else next_state = S20;
273 end
274
275 default: next_state = S0;
276
277 endcase
278
279 endmodule
```

fsm\_tb:

My testbench goes through every possible situation.

DUT was made sure working by checking “lock” signal, states and sc/cc values.

```
1 module fsm_tb();
2
3 logic clk;
4 logic zero, one, two, three, four, five, six, seven, eight, nine, enter;
5 logic init, reset, clear;
6 logic [3:0] press_val;
7 logic lock;
8
9 //instantiation of DUT
10 fsm qwer(.clk(clk), .zero(zero), .one(one), .two(two), .three(three), .four(four), .five(five), .six(six), .seven(seven), .eight(eight), .nine(nine),
11         .enter(enter), .init(init), .reset(reset), .clear(clear), .press_val(press_val), .lock(lock));
12
13 //generate clock
14 always
15 begin
16     clk = 1; #5; clk = 0; #5;
17 end
18
19 //test
20 initial begin
21     //these are physical buttons, unpressed = 0, pressed = 1.
22     zero = 0;
23     one = 0;
24     two = 0;
25     three = 0;
26     four = 0;
27     five = 0;
28     six = 0;
29     seven = 0;
30     eight = 0;
31     nine = 0;
32     enter = 0;
33     clear = 0;
34     reset = 0;
35
36     //initialize (not really required since our default state is S0, just to make sure)
37     init = 1; #30; init = 0; #18; //18 delay here is a random number since button press can happen at any time
38
39
40
41 /*
42  Testing Security code states (S0-S4)
43  */
44
45 //Clear tests
46 //S0 -> S1 -> S0
47 two = 1; #30; two = 0; #30;
48 clear = 1; #30; clear = 0; #30;
49 //S0 -> S1 -> S2 -> S0
50 two = 1; #30; two = 0; #30;
51 one = 1; #30; one = 0; #30;
52 clear = 1; #30; clear = 0; #30;
53 //S0 -> S1 -> S2 -> S3 -> S0
54 three = 1; #30; three = 0; #30;
55 two = 1; #30; two = 0; #30;
56 four = 1; #30; four = 0; #30;
57 clear = 1; #30; clear = 0; #30;
58 //S0 -> S1 -> S2 -> S3 -> S4 -> S0
59 one = 1; #30; one = 0; #30;
60 three = 1; #30; three = 0; #30;
61 two = 1; #30; two = 0; #30;
62 four = 1; #30; four = 0; #30;
63 clear = 1; #30; clear = 0; #30;
64 //Enter next stage (sc = 16'h1672)
65 //S0 -> S1 -> S2 -> S3 -> S4 -> S5
66 two = 1; #30; two = 0; #30;
67 seven = 1; #30; seven = 0; #30;
68 six = 1; #30; six = 0; #30;
69 one = 1; #30; one = 0; #30;
70 enter = 1; #30; enter = 0;
71
```

```

73  /*
74  Testing Customer code states (S5-S9)
75  */
76
77  //Clear tests
78  //S5 -> S6 -> S5
79  five = 1; #30; five = 0; #30;
80  clear = 1; #30; clear = 0; #30;
81  //S5 -> S6 -> S7 -> S5
82  eight = 1; #30; eight = 0; #30;
83  one = 1; #30; one = 0; #30;
84  clear = 1; #30; clear = 0; #30;
85  //S5 -> S6 -> S7 -> S8 -> S5
86  nine = 1; #30; nine = 0; #30;
87  two = 1; #30; two = 0; #30;
88  four = 1; #30; four = 0; #30;
89  clear = 1; #30; clear = 0; #30;
90  //S5 -> S6 -> S7 -> S8 -> S9 -> S5
91  one = 1; #30; one = 0; #30;
92  three = 1; #30; three = 0; #30;
93  two = 1; #30; two = 0; #30;
94  four = 1; #30; four = 0; #30;
95  clear = 1; #30; clear = 0; #30;
96  //Enter next stage (cc = 16'h6420)
97  //S5 -> S6 -> S7 -> S8 -> S9 -> S10
98  zero = 1; #30; zero = 0; #30;
99  two = 1; #30; two = 0; #30;
100 four = 1; #30; four = 0; #30;
101 six = 1; #30; six = 0; #30;
102 enter = 1; #30; enter = 0;
103
104
105  /*
106  Testing Resetting/Counterfeit states (S10-S17)
107  */
108  //Clear tests
109  //S10 -> S11 -> S10
110  reset = 1; #30; reset = 0; #30;
111  clear = 1; #30; clear = 0; #30;
112  //S10 -> S11 -> S12 -> S10
113  reset = 1; #30; reset = 0; #30;
114  two = 1; #30; two = 0; #30; //sc[3:0]
115  clear = 1; #30; clear = 0; #30;
116  //S10 -> S11 -> S12 -> S13 -> S10
117  reset = 1; #30; reset = 0; #30;
118  two = 1; #30; two = 0; #30; //sc[3:0]
119  seven = 1; #30; seven = 0; #30; //sc[7:4]
120  clear = 1; #30; clear = 0; #30;
121  //S10 -> S11 -> S12 -> S13 -> S14 -> S10
122  reset = 1; #30; reset = 0; #30;
123  two = 1; #30; two = 0; #30; //sc[3:0]
124  seven = 1; #30; seven = 0; #30; //sc[7:4]
125  six = 1; #30; six = 0; #30; //sc[11:8]
126  clear = 1; #30; clear = 0; #30;
127  //S10 -> S15 -> S10
128  one = 1; #30; one = 0; #30;
129  clear = 1; #30; clear = 0; #30;
130  //S10 -> S15 -> S16 -> S10
131  one = 1; #30; one = 0; #30;
132  one = 1; #30; one = 0; #30;
133  clear = 1; #30; clear = 0; #30;
134  //S10 -> S15 -> S16 -> S17 -> S10
135  one = 1; #30; one = 0; #30;
136  one = 1; #30; one = 0; #30;
137  one = 1; #30; one = 0; #30;
138  clear = 1; #30; clear = 0; #30;
139

```

```

140 //Security code fail tests
141 //S10 -> S11 -> S15 -> S16 -> S17 -> S10
142 reset = 1; #30; reset = 0; #30;
143 one = 1; #30; one = 0; #30;
144 one = 1; #30; one = 0; #30;
145 one = 1; #30; one = 0; #30;
146 zero = 1; #30; zero = 0; #30;
147 //S10 -> S11 -> S12 -> S16 -> S17 -> S10
148 reset = 1; #30; reset = 0; #30;
149 two = 1; #30; two = 0; #30; //sc[3:0]
150 three = 1; #30; three = 0; #30;
151 one = 1; #30; one = 0; #30;
152 zero = 1; #30; zero = 0; #30;
153 //S10 -> S11 -> S12 -> S13 -> S17 -> S10
154 reset = 1; #30; reset = 0; #30;
155 two = 1; #30; two = 0; #30; //sc[3:0]
156 seven = 1; #30; seven = 0; #30; //sc[7:4]
157 three = 1; #30; three = 0; #30;
158 zero = 1; #30; zero = 0; #30;
159 //S10 -> S11 -> S12 -> S13 -> S14 -> S10
160 reset = 1; #30; reset = 0; #30;
161 two = 1; #30; two = 0; #30; //sc[3:0]
162 seven = 1; #30; seven = 0; #30; //sc[7:4]
163 six = 1; #30; six = 0; #30; //sc[11:8]
164 zero = 1; #30; zero = 0; #30;
165
166 //Fully Counterfeit test
167 //S10 -> S15 -> S16 -> S17 -> S10
168 three = 1; #30; three = 0; #30;
169 three = 1; #30; three = 0; #30;
170 three = 1; #30; three = 0; #30;
171 three = 1; #30; three = 0; #30;
172
173
174 //Reset test
175 //S10 -> S11 -> S12 -> S13 -> S14 -> S5
176 reset = 1; #30; reset = 0; #30;
177 two = 1; #30; two = 0; #30; //sc[3:0]
178 seven = 1; #30; seven = 0; #30; //sc[7:4]
179 six = 1; #30; six = 0; #30; //sc[11:8]
180 one = 1; #30; one = 0; #30; //sc[15:12]
181
182 //Back to S10 with same cc (cc = 16'h6420)
183 //S5 -> S6 -> S7 -> S8 -> S9 -> S10
184 zero = 1; #30; zero = 0; #30;
185 two = 1; #30; two = 0; #30;
186 four = 1; #30; four = 0; #30;
187 six = 1; #30; six = 0; #30;
188 enter = 1; #30; enter = 0;
189

```

```

190  /*
191  Testing Unlocking states (S18-S20)
192  */
193
194  //Clear tests
195      //S10 -> S18 -> S10
196      zero = 1; #30; zero = 0; #30;    //cc[3:0]
197      clear = 1; #30; clear = 0; #30;
198      //S10 -> S18 -> S19 -> S10
199      zero = 1; #30; zero = 0; #30;    //cc[3:0]
200      two = 1; #30; two = 0; #30;      //cc[7:4]
201      clear = 1; #30; clear = 0; #30;
202      //S10 -> S18 -> S19 -> S20 -> S10
203      zero = 1; #30; zero = 0; #30;    //cc[3:0]
204      two = 1; #30; two = 0; #30;      //cc[7:4]
205      four = 1; #30; four = 0; #30;    //cc[11:8]
206      clear = 1; #30; clear = 0; #30;
207
208  //Customer code fail tests
209      //S10 -> S18 -> S16 -> S17 -> S10
210      zero = 1; #30; zero = 0; #30;    //cc[3:0]
211      three = 1; #30; three = 0; #30;
212      three = 1; #30; three = 0; #30;
213      three = 1; #30; three = 0; #30;
214      //S10 -> S18 -> S19 -> S17 -> S10
215      zero = 1; #30; zero = 0; #30;    //cc[3:0]
216      two = 1; #30; two = 0; #30;      //cc[7:4]
217      three = 1; #30; three = 0; #30;
218      three = 1; #30; three = 0; #30;
219      //S10 -> S18 -> S19 -> S20 -> S10
220      zero = 1; #30; zero = 0; #30;    //cc[3:0]
221      two = 1; #30; two = 0; #30;      //cc[7:4]
222      four = 1; #30; four = 0; #30;    //cc[11:8]
223      three = 1; #30; three = 0; #30;
224
225  //Unlock test
226      //S10 -> S18 -> S16 -> S17 -> S5
227      zero = 1; #30; zero = 0; #30;    //cc[3:0]
228      two = 1; #30; two = 0; #30;      //cc[7:4]
229      four = 1; #30; four = 0; #30;    //cc[11:8]
230      six = 1; #30; six = 0; #30;      //cc[15:12]
231
232  end
233  endmodule

```

## Simulation

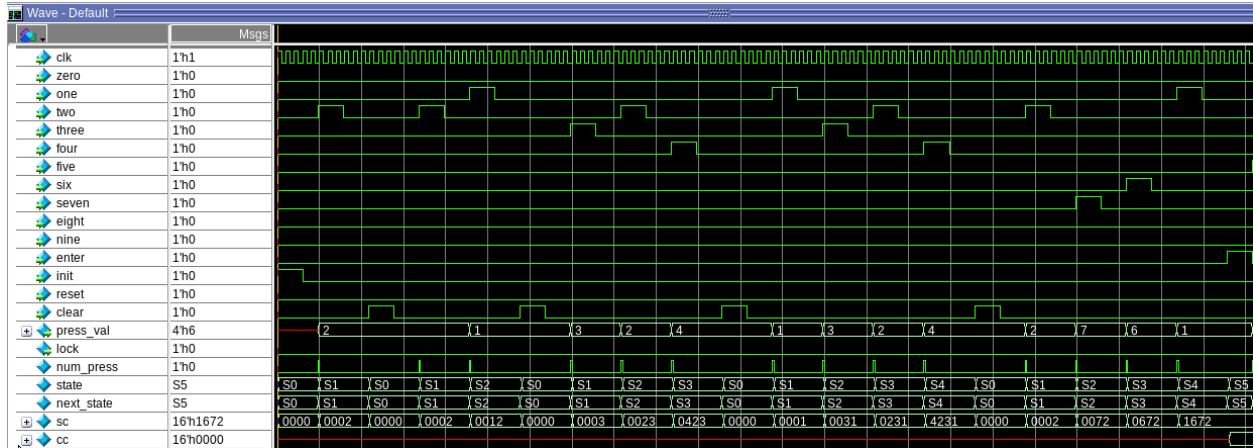


Figure 6. Waveform of “Testing security code states (S0-S4)” in TB

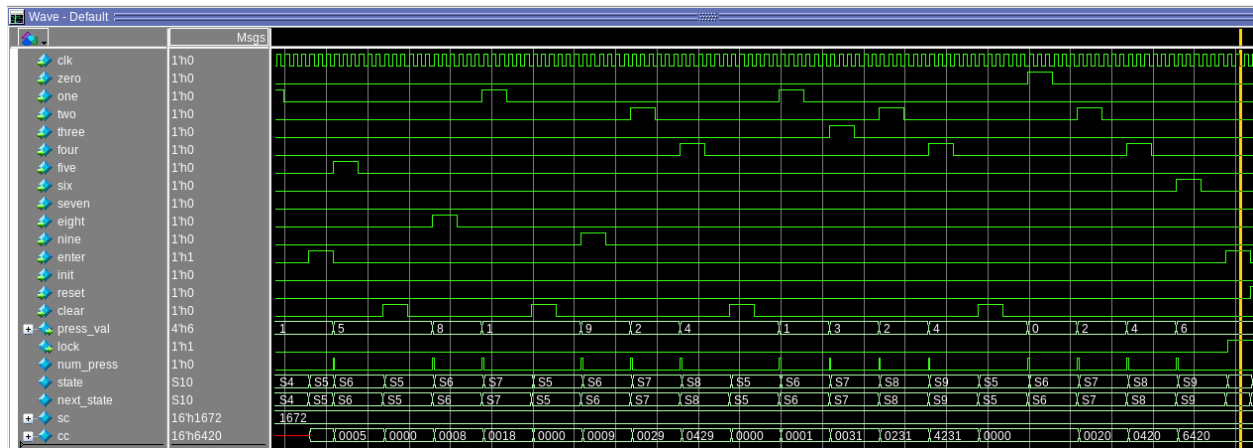


Figure 7. Waveform of “Testing customer code states (S5-S9)” in TB

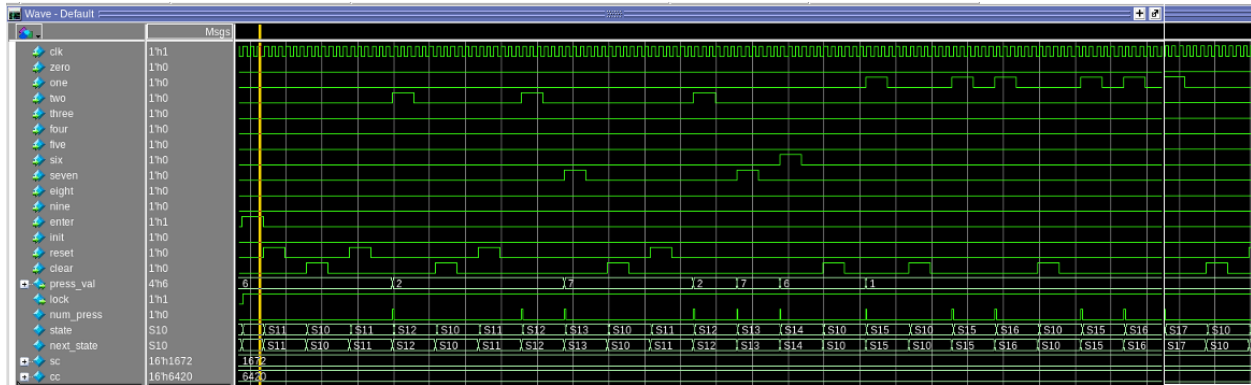


Figure 8. Waveform of “Testing resetting/counterfeit states (S10-S17)”, “Clear tests” in TB

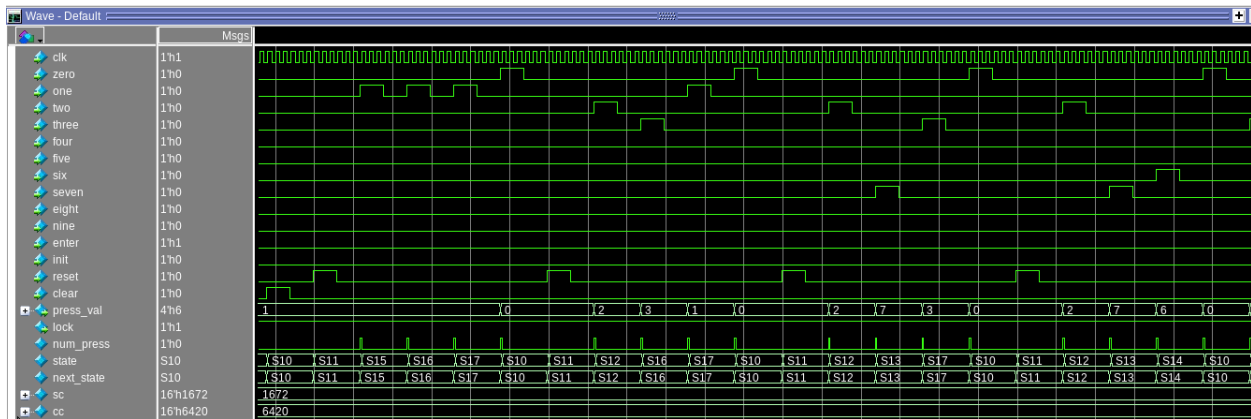


Figure 9. Waveform of “Testing resetting/counterfeit states (S10-S17)”, “Security code fail tests” in TB

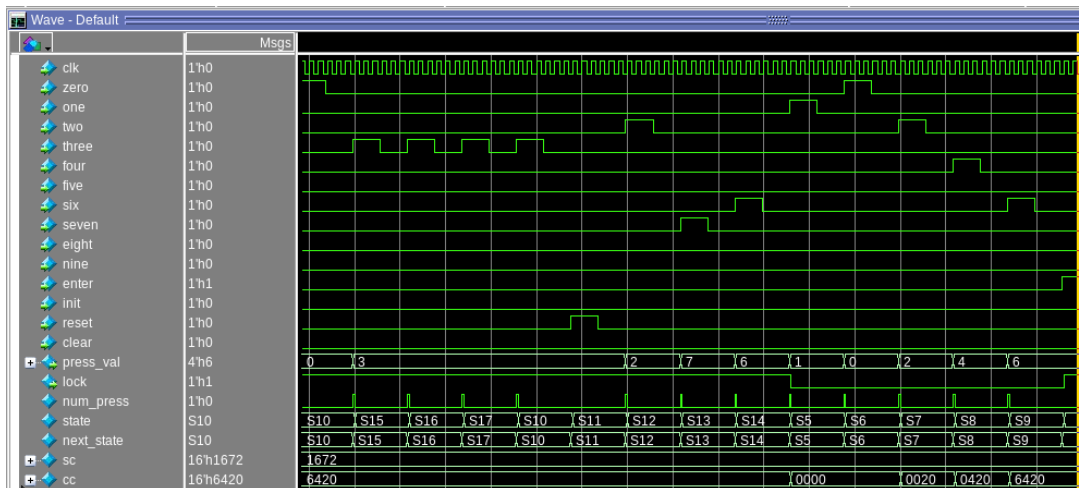


Figure 10. Waveform of “Testing resetting/counterfeit states (S10-S17)”, “Fully counterfeit test” + “Reset test” + “Back to S10” with same cc” in TB

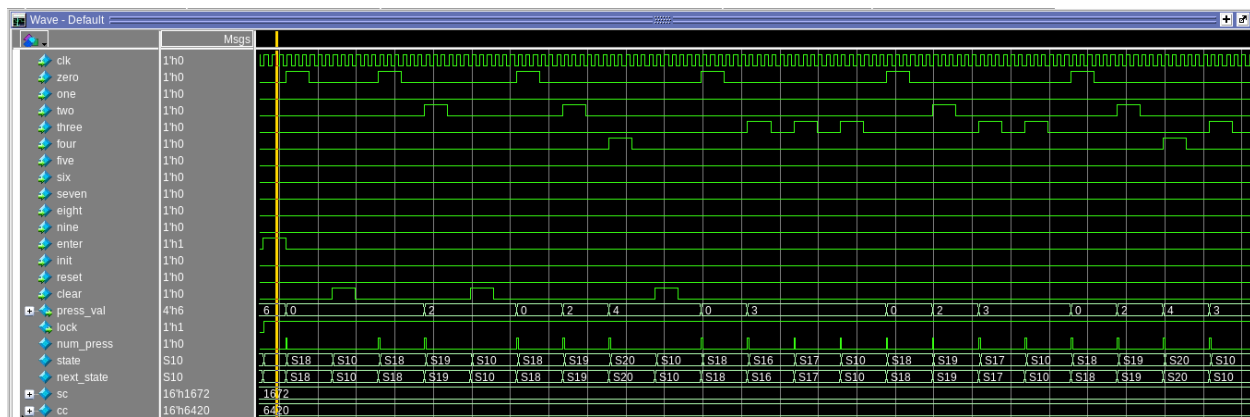


Figure 11. Waveform of “Testing unlocking states”, “Clear tests” + “Customer code fail tests” in TB

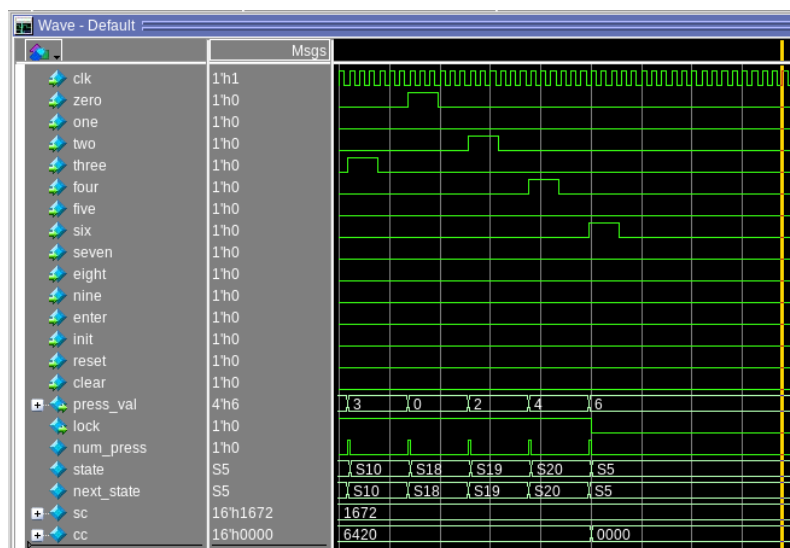


Figure 12. Waveform of “Testing unlocking states”, “Unlock test” in TB