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):

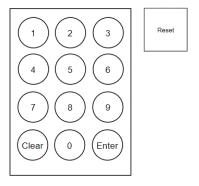


Figure 2.

# **Block Diagrams**

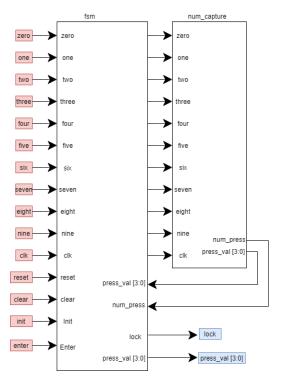


Figure 3. Block diagram of FSM modules (inputs are red, outputs are blue)

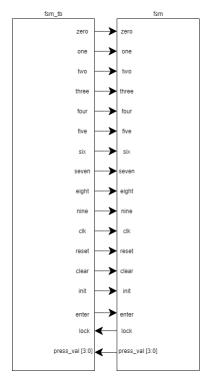


Figure 4. Block diagram of TB (fsm\_tb) and DUT (fsm)

### State diagram of FSM flow

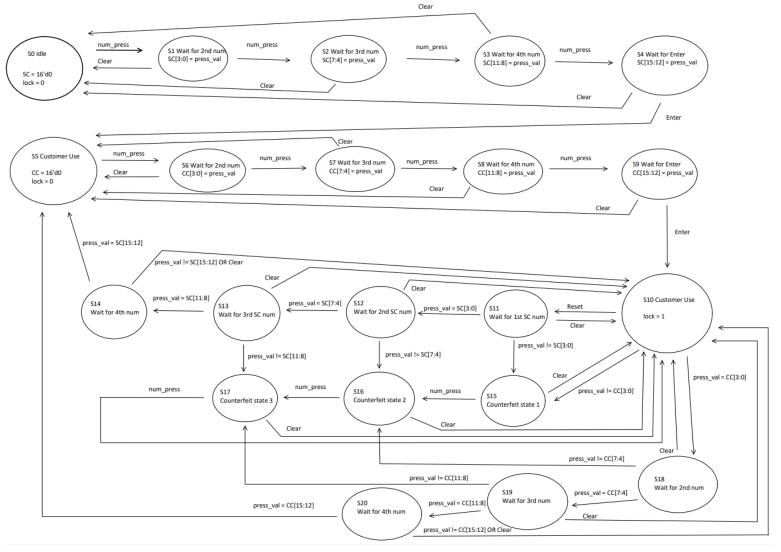


Figure 5. State diagram of FSM

- 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:

```
Number button capturer
          (c) Copyright Eric Wu
All rights reserved.
          Author: Eric Wu
          Email: ew820120@gmail.com
          Student ID: 75286914
10
11
12
13
14
15
16
17
18
          Description:
          A number button capturer that detects button press and saves pressed value.

    button signal for zero (pressed = 1, unpressed = 0)
    button signal for one (pressed = 1, unpressed = 0)
    button signal for two (pressed = 1, unpressed = 0)

               one
                                  - button signal for three (pressed = 1, unpressed = 0)
- button signal for four (pressed = 1, unpressed = 0)
- button signal for five (pressed = 1, unpressed = 0)
20
21
               three
               four
22
23
24
25
26
27
28
               five
                                 - button signal for five (pressed = 1, unpressed = 0)
- button signal for six (pressed = 1, unpressed = 0)
- button signal for seven (pressed = 1, unpressed = 0)
- button signal for eight (pressed = 1, unpressed = 0)
- button signal for nine (pressed = 1, unpressed = 0)
               six
               seven
               eight
               nine
               clk
                                           value of number pressededge captured signal of button press
30
31
               press_val
               num_press
36
37
38
40
41
42
44
45
46
47
48
49
50
51
55
55
56
67
68
69
70
71
72
73
74
        module num_capture(input logic zero, one, two, three, four, five, six, seven, eight, nine,
                                        input logic clk,
output logic [3:0] press_val,
                                        output logic num_press);
           edge detector: detects key press and provides output edge captured signal num_press.
           two inverters were used as delay for the output num_press, amount of delay will be configured after synthesis.
          logic key_press;
logic delay_reg;
          logic num_press_d0, num_press_d1;
           assign key_press = zero | one | two | three | four | five | six | seven | eight | nine;
           always_ff @(posedge clk)
          delay_reg <= key_press;
          assign num_press_d0 = ~delay_reg & key_press;
assign num_press_d1 = ~num_press_d0;
           assign num_press = ~num_press_d1;
           //gets number press value each key press
          always_ff @(posedge key_press)

case((zero, one, two, three, four, five, six ,seven ,eight, nine))
10'b0000000001: press_val = 4'd9;
10'b0000000010: press_val = 4'd8;
          10'b0000000100: press_val
10'b0000001000: press_val
                                                   = 4'd7;
= 4'd6;
           10'b0000010000: press_val
                                                   = 4'd5;
= 4'd4;
           10'b0000100000: press_val
           10'b00010000000: press_val
           10'b00100000000: press_val = 4'd2;
           10'b0100000000: press_val
          10'b1000000000: press_val = 4'd0;
default: press_val = 4'b1111; //idle
           endcase
          endmodule
```

#### fsm:

```
Hotel safe box lock FSM
2
       (c) Copyright Eric Wu
       All rights reserved.
5
6
       Author: Eric Wu
7
       Email: ew820120@gmail.com
       Student ID: 75286914
9
10
11
12
       Description:
       A FSM that goes through three main stages.
       - Setting up security code S0~S4
- Setting up customer code S5~S9
14
15
       - Unlocking and Resetting S10~S20
18
       Security code saved in sc[15:0]
19
       Customer code saved in cc[15:0]
20
21
       Inputs:
           clk
22
                          - clock
23
           zero
                         - button signal for zero (pressed = 1, unpressed = 0)
24
                         - button signal for one (pressed = 1, unpressed = 0)
- button signal for two (pressed = 1, unpressed = 0)
           one
25
           two
26
           three
                          - button signal for three (pressed = 1, unpressed = 0)
27
                          - button signal for four (pressed = 1, unpressed = 0)
           four
28
           five
                         - button signal for five (pressed = 1, unpressed = 0)
29
           six
                         - button signal for six (pressed = 1, unpressed = 0)
                         - button signal for seven (pressed = 1, unpressed = 0)

- button signal for eight (pressed = 1, unpressed = 0)
30
           seven
31
           eight
32
                          - button signal for nine (pressed = 1, unpressed = 0)
           nine
33
                         - button signal for enter (pressed = 1, unpressed = 0)
                          - signal for force initialization (state goes to S0)
34
35
           init
                         - button signal for resetting cc[15:0] (pressed = 1, unpressed = 0)
           reset
                         - button signal for clearing previous entered numbers (pressed = 1, unpressed = 0)
36
           clear
37
38
           press_val
39
                                 - value of number pressed (for HEX display purposes, which is not included in this project)
                                 - lock signal, locked = 1, unlocked = 0 (controls the actual lock)
40
           lock
41
42
45
     □ module fsm(input logic clk,
                     input logic zero, one, two, three, four, five, six, seven, eight, nine, enter, input logic init, reset, clear, output logic [3:0] press_val,
46
48
49
                     output logic lock);
50
51
52
53
        //instantiation of num_capture
        logic num press:
      num_capture asdf(.zero(zero), .one(one), .two(two), .three(three), .four(four), .five(five), .six(six), .seven(seven), .eight(eight), .nine(nine), .clk(clk), .press_val(press_val), .num_press(num_press));
54
55
56
57
58
      //fsm state declaration and variables
typedef enum logic [4:0] (S0, S1, S2, S3, S4, S5, S6, S7, S8, S9, S10, S11, S12, S13, S14, S15, S16, S17,
59
                                      $18, $19, $20, $21, $22, $23, $24, $25, $26, $27, $28, $29, $30, $31} statetype;
        statetype state, next_state;
logic [15:0] sc; //security code
logic [15:0] cc; //customer code
60
61
62
63
64
        //state register
        always_ff @(posedge clk)
if (init) state <= S0;
65
67 68
        else state <= next_state;</pre>
69
        //next state logic
70
71
72
73
        always_comb
      case(state)
75
76
        //Security code setup
        S0: begin
lock = 1'b0;
        sc = 16'd0;
if (num_press) next_state = S1:
79
        else next_state = S0;
80
```

```
resc[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 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
 85
86
 87
88
      S1: begin
 89
90
91
        if (clear) next_state = S0;
      else if (num_press) next_state = S2; else begin
        next_state = S1;
 92
93
94
95
96
97
98
        sc[3:0] = press_val;
         end
      S2: begin
if (clear) next_state = S0;
99
100
        else if (num_press) next_state = S3;
      else begin
next_state = S2;
101
102
103
        sc[7:4] = press_val;
104
105
        end
      S3: begin

if (clear) next_state = S0;

else if (num_press) next_state = S4;
106
107
109
110
      else begin
next_state = S3;
111
112
113
         sc[11:8] = press_val;
        end
114
115
      116
117
        if (clear) next_state = S0;
else if (enter) next_state = S5;
        else begin
        next_state = S4;
119
        sc[15:12] = press_val;
121
122
        end
        end
       //Customer code setup
124
 125
         lock = 1'b0;
 126
          cc = 16'b0;
if (num_press) next_state = S6;
 127
 128
 129
          else next_state = S5;
 130
 131
        S6: begin if (clear) next_state = S5;
 132
 133
 134
         else if (num_press) next_state = S7;
        else begin
 135
          next_state = S6;
cc[3:0] = press_val;
 136
137
 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
          next_state = S7;
 145
 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;
         else begin
 154
          next_state = S8;
           cc[11:8] = press_val;
 155
 156
          end
 157
           end
 158
        □ S9: begin
 159
          if (clear) next_state = S5;
 160
          else if (enter) next_state = S10;
 161
 162
         else begin
 163
           next_state = S9;
 164
           cc[15:12] = press_val;
 165
 166
          - end
```

```
//Unlock and Reset states

□ S10: begin
         lock = 1'b1;
  171
           if (reset) next_state = S11;
         else if (num_press) begin
  172
  173
           if (press_val == cc[3:0])
  174
            next_state = S18;
 175
176
            else
            next_state = S15;
  177
            end
  178
          else next_state = S10;
  179
           end
  180
        S11: begin if (clear)
  181
  182
          if (clear) next_state = S10;
        else if (clear) next_state = Si
else if (num_press) begin
if (press_val == sc[3:0])
next_state = S12;
else next_state = S15;
 184
185
  186
            end
  188
          else next_state = S11;
  189
           end
  190
        S12: begin
if (clear) next_state = S10;
else if (num_press) begin
if (press_val == sc[7:4])
  191
  192
  193
  194
  195
            next_state = S13;
            else next_state = S16;
  197
            end
          else next_state = S12;
  198
  199
          end
  200
        S13: begin
if (clear) next_state = S10;
else if (num_press) begin
if (press_val == sc[11:8])
  201
 202
203
  204
  205
            next_state = S14;
 206 207
            else next_state = S17;
            end
  208
           else next_state = S13;
       □ S14: begin
212
        if (clear) next_state = S10;
       else if (num_press) begin
214
215
216
          if (press_val == sc[15:12])
next_state = S5;
          else next_state = S10;
217
          end
218 219
         else next_state = S14;
         end
220
         S15:
if (clear)
next_state = S10;
221
222
223
224
          else if (num_press)
225
226
227
         next_state = S16;
         else
         next_state = S15;
228
229
         S16:
230
         if (clear)
231
         next_state = S10;
          else if (num_press)
233
         next_state = S17;
234
         else
235
         next_state = S16;
236
237
238
         S17:
if (clear)
239
         next_state = S10;
240
          else if (num_press)
241 242
         next_state = S10;
         else
243
         next_state = S17;
244
       □ S18: begin
245
       if (clear) next_state = $10;
else if (num_press) begin
if (press_val == cc[7:4])
246
248
249
250
          next_state = S19;
else next_state = S16;
251
          end
         else next_state = S18;
253
         end
```

```
255 🛱 S19: begin
     if (clear) next_state = S10;
else if (num_press) begin
256
257
       if (press_val == cc[11:8])
next_state = S20;
258
259
260
       else next_state = S17;
       end
261
262
       else next_state = S19;
263
      - end
264
265 卓 S20: begin
266  if (clear) next_state = S10;
267 pelse if (num_press) begin
       if (press_val == cc[15:12])
268
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
      endmodule
279
```

#### fsm tb:

My testbench goes through every possible situation.

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

```
□ module fsm_tb();
     logic clk;
     logic zero, one, two, three, four, five, six, seven, eight, nine, enter;
     logic init, reset, clear;
     logic [3:0] press_val;
     logic lock:
     //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
12
             .enter(enter), .init(init), .reset(reset), .clear(clear), .press_val(press_val), .lock(lock));
     //generate clock
14
15
16
     always
    begin
     clk = 1; #5; clk = 0; #5;
17
18
19
     end
20
21
     //test
   initial begin
22
23
24
     //these are physical buttons, unpressed = 0, pressed = 1.
     zero = 0;
     one = 0;
two = 0;
25
26
27
28
29
30
31
32
     three = 0;
     four = 0;
five = 0;
     seven = 0:
     eight = 0;
33
34
35
     enter = 0:
     clear = 0;
36
37
     reset = 0;
     //initialize (not really required since our default state is S0, just to make sure)
    init = 1; #30; init = 0; #18; //18 delay here is a random number since button press can happen at any time
41
      占 /*
 42
        Testing Security code states (S0-S4)
 43
        */
 44
 45
            //Clear tests
 46
               //S0 -> S1 -> S0
 47
               two = 1; #30; two = 0; #30;
               clear = 1; #30; clear = 0; #30;
 48
 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;
               three = 1; #30; three = 0; #30;
 60
                two = 1; #30; two = 0; #30;
 61
 62
               four = 1; #30; four = 0; #30;
 63
               clear = 1; #30; clear = 0; #30;
 64
            //Enter next stage (sc = 16'h1672)
               //S0 -> S1 -> S2 -> S3 -> S4 -> S5
 65
 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
          //Clear tests
 77
 78
             //S5 -> S6 -> S5
             five = 1; #30; five = 0; #30;
 79
 80
             clear = 1; #30; clear = 0; #30;
             //S5 -> S6 -> S7 -> S5
 81
             eight = 1; #30; eight = 0; #30;
 82
             one = 1; #30; one = 0; #30;
 83
 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;
             four = 1; #30; four = 0; #30;
 88
 89
             clear = 1; #30; clear = 0; #30;
 90
             //S5 -> S6 -> S7 -> S8 -> S9 -> S5
 91
             one = 1; #30; one = 0; #30;
             three = 1; #30; three = 0; #30;
 92
             two = 1; #30; two = 0; #30;
 93
 94
             four = 1; #30; four = 0; #30;
 95
             clear = 1; #30; clear = 0; #30;
          //Enter next stage (cc = 16'h6420)
 96
 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
      Testing Resetting/Counterfeit states (S10-S17)
106
107
       */
          //Clear tests
108
 109
             //S10 -> S11 -> S10
             reset = 1; #30; reset = 0; #30;
 110
 111
             clear = 1; #30; clear = 0; #30;
             //S10 -> S11 -> S12 -> S10
112
             reset = 1; #30; reset = 0; #30;
 113
                                              //sc[3:0]
114
             two = 1; #30; two = 0; #30;
             clear = 1; #30; clear = 0; #30;
115
             //S10 -> S11 -> S12 -> S13 -> S10
116
             reset = 1; #30; reset = 0; #30;
 117
             two = 1; #30; two = 0; #30;
118
                                              //sc[3:0]
             seven = 1; #30; seven = 0; #30; //sc[7:4]
 119
             clear = 1; #30; clear = 0; #30;
 120
             //S10 -> S11 -> S12 -> S13 -> S14 -> S10
 121
             reset = 1; #30; reset = 0; #30;
122
123
             two = 1; \#30; two = 0; \#30;
                                              //sc[3:0]
             seven = 1; #30; seven = 0; #30;
 124
                                              //sc[7:4]
                                              //sc[11:8]
 125
             six = 1; #30; six = 0; #30;
 126
             clear = 1; #30; clear = 0; #30;
             //S10 -> S15 -> S10
 127
 128
             one = 1; #30; one = 0; #30;
             clear = 1; #30; clear = 0; #30;
 129
             //S10 -> S15 -> S16 -> S10
130
131
             one = 1; #30; one = 0; #30;
132
             one = 1; #30; one = 0; #30;
             clear = 1; #30; clear = 0; #30;
 133
             //S10 -> S15 -> S16 -> S17 -> S10
134
             one = 1; #30; one = 0; #30;
 135
             one = 1; #30; one = 0; #30;
 136
 137
             one = 1; #30; one = 0; #30;
138
             clear = 1; #30; clear = 0; #30;
139
```

```
140 白
          //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;
             //S10 -> S11 -> S12 -> S16 -> S17 -> S10
147
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
```

```
白 /*
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;
              //S10 -> S18 -> S19 -> S20 -> S10
202
              zero = 1; #30; zero = 0; #30;
203
                                                //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
     白
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;
              //S10 -> S18 -> S19 -> S17 -> S10
214
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]
              two = 1; #30; two = 0; #30;
221
                                                //cc[7:4]
222
              four = 1; #30; four = 0; #30;
                                                //cc[11:8]
223
              three = 1; #30; three = 0; #30;
224
225
     中
          //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

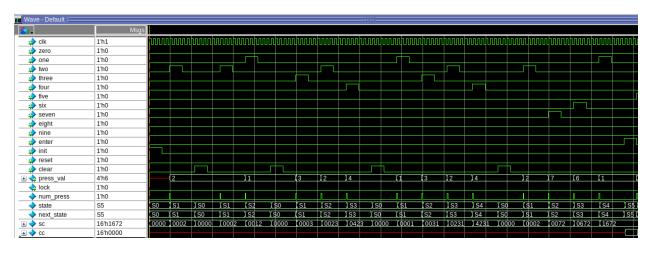


Figure 6. Waveform of "Testing security code states (S0-S4)" in TB

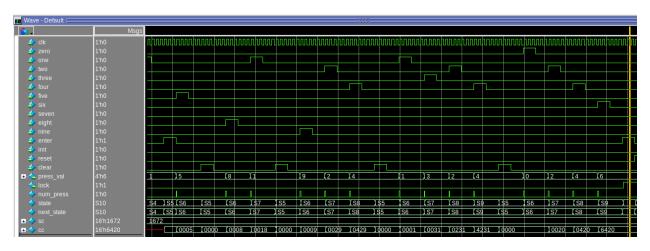


Figure 7. Waveform of "Testing customer code states (S5-S9)" in TB

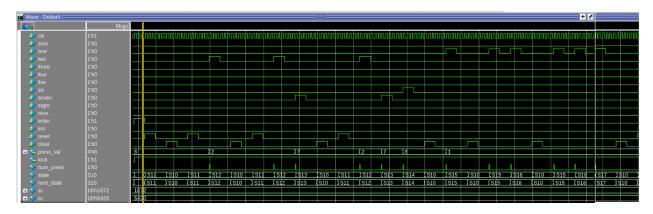


Figure 8. Waveform of "Testing resetting/counterfeit states (S10-S17)", "Clear tests" in TB

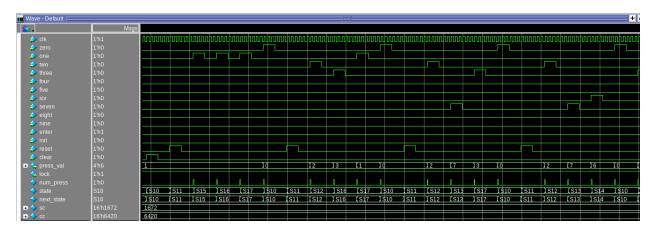


Figure 9. Waveform of "Testing resetting/counterfeit states (S10-S17)", "Security code fail tests" in TB

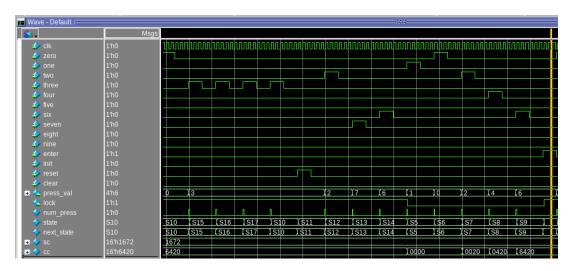


Figure 10. Waveform of "Testing resetting/counterfeit states (S10-S17)", "Fully counterfeit test" + "Reset test" + "Back to S10" with same cc" in TB

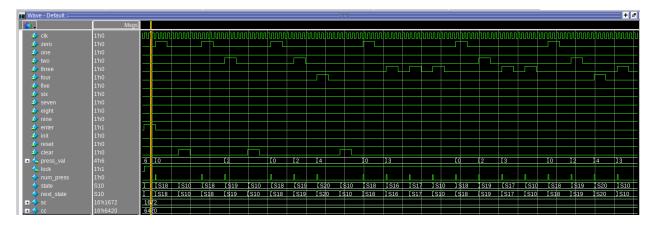


Figure 11. Waveform of "Testing unlocking states", "Clear tests" + "Customer code fail tests" in TB

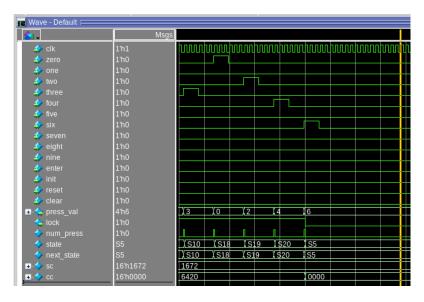


Figure 12. Waveform of "Testing unlocking states", "Unlock test" in TB