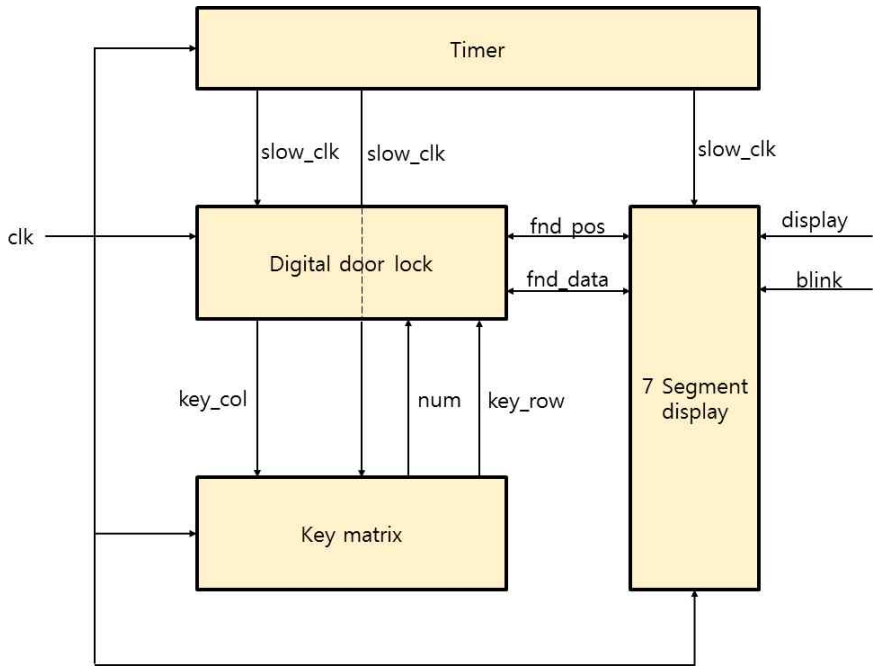
디논실 기말 프로젝트
2013147513 조영재

Assignment #2 - Digital Door Lock design

1. 개요

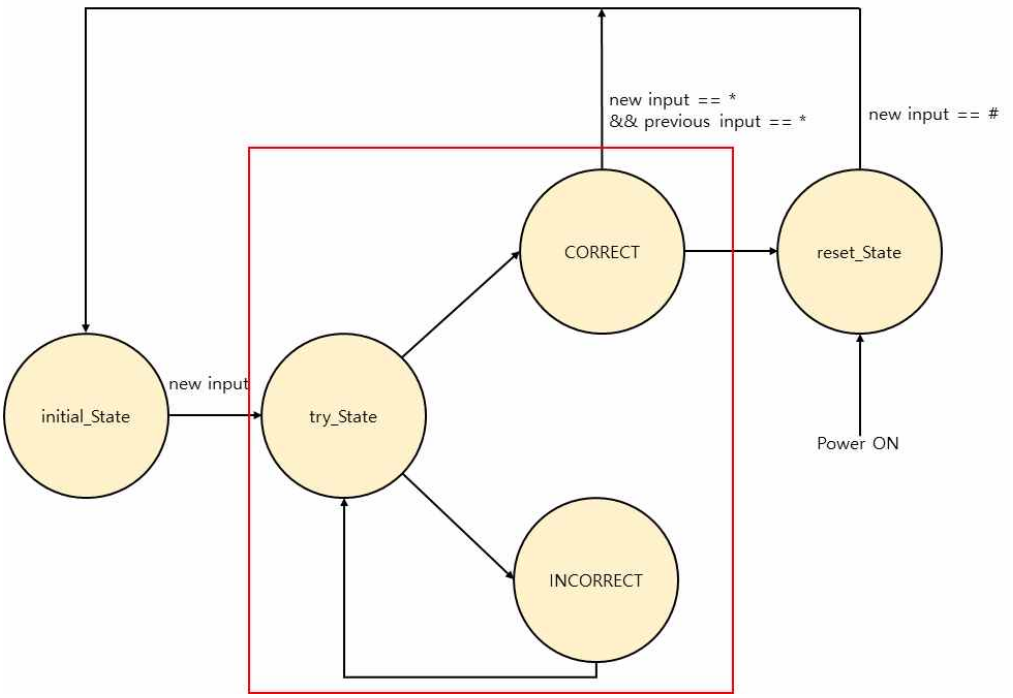
이번 프로젝트에서는 Quartus II를 활용하여 Combinational digital door lock을 설계하고, FPGA상에서 시험해본다.

2. 블록 다이어그램

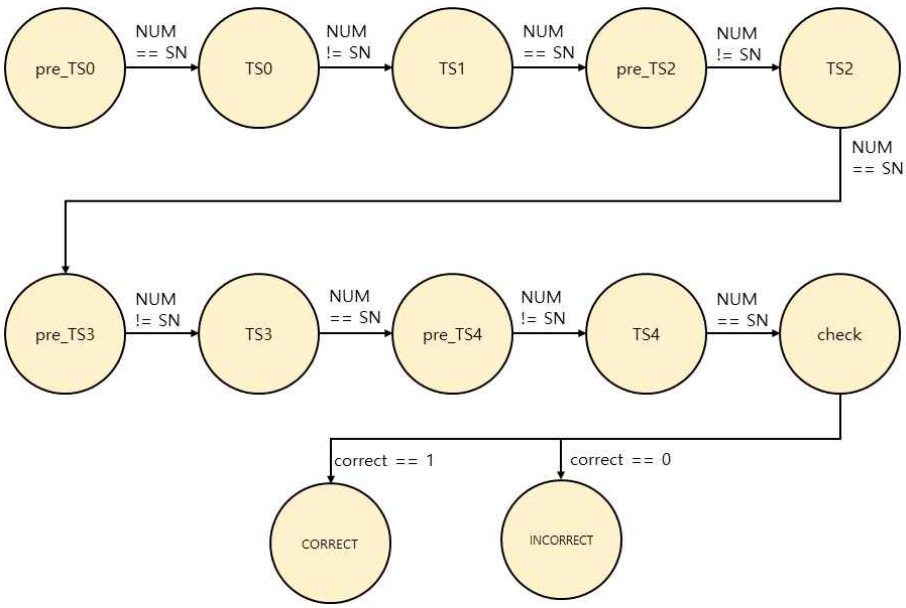


디논실 기말 프로젝트
2013147513 조영재

3. FSM 상태전이도



전체적인 상태전이도는 위와 같고 빨간색 사각형의 자세한 상태전이도는 다음과 같다.

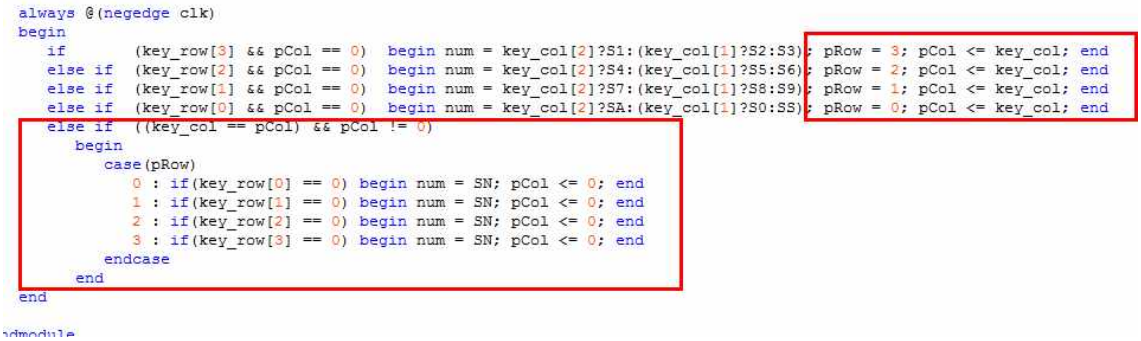


디논실 기말 프로젝트
2013147513 조영재

4. 모듈 별 동작 설명

- key_matrix

기본적으로 주어져 있던 코드 중 가장 먼저 필요하였던 것이 key matrix에 아무 입력도 없을 때 num에 SN의 값이 연결되도록 하는 것 이었다. 따라서 key matrix코드를 다음과 같이 약간 수정하였다.



```
always @(negedge clk)
begin
   if       (key_row[3] && pCol == 0)  begin num = key_col[2]?S1:(key_col[1]?S2:S3); pRow = 3; pCol <= key_col; end
   else if  (key_row[2] && pCol == 0)  begin num = key_col[2]?S4:(key_col[1]?S5:S6); pRow = 2; pCol <= key_col; end
   else if  (key_row[1] && pCol == 0)  begin num = key_col[2]?S7:(key_col[1]?S8:S9); pRow = 1; pCol <= key_col; end
   else if  (key_row[0] && pCol == 0)  begin num = key_col[2]?SA:(key_col[1]?S0:SS); pRow = 0; pCol <= key_col; end
   else if  ((key_col == pCol) && pCol != 0)
      begin
         case(pRow)
            0 : if(key_row[0] == 0) begin num = SN; pCol <= 0; end
            1 : if(key_row[1] == 0) begin num = SN; pCol <= 0; end
            2 : if(key_row[2] == 0) begin num = SN; pCol <= 0; end
            3 : if(key_row[3] == 0) begin num = SN; pCol <= 0; end
         endcase
      end
end
```

위의 부분의 코드를 추가하여서 어떠한 key matrix도 눌리지 않는다면, num에는 SN이 저장될 수 있도록 하였다.

- digital_door_lock

위에서 수정한 key_matrix를 가지고 digital door lock을 작성하였다. 3번에서 그린 FSM상태전이도와 같은 수행을 하도록 설계하였다.

① reset_State

reset_State는 처음 전원이 들어왔을 때의 State이므로, initial문 내에 초기화를 시켜준다. 또한, 그 FSM상태전이도는 다음과 같이 설계할 수 있다.

```
//초기화
initial begin
   Current_State = reset_State;
   R_STATE = RS0;
end
```

디논실 기말 프로젝트
2013147513 조영재

```
always @ (posedge slow_clk) begin
   //리셋상태일 경우 비밀번호 입력
   case(Current_State)
      pre_reset_State: if(num == SN) Current_State = reset_State;
      reset_State:
            case(R_STATE)
               pre_RS0: if(num == SN) R_STATE = RS0;
               RS0:begin
                     display[23:20] = SN;
                     display[19:16] = SN;
                     display[15:12] = SN;
                     display[11:8]  = SN;
                     display[7:4]   = SN;
                     display[3:0]   = SS;
                     if(num != SN) R_STATE <= RS1;
                  end
               RS1:begin
                     if(num != SN) C3 = num;
                     else if(num == SN) begin
                        display[23:20] = SN;
                        display[19:16] = SN;
                        display[15:12] = SN;
                        display[11:8]  = SN;
                        display[7:4]   = SS;
                        display[3:0]   = C3;
                        R_STATE <= pre_RS2;
                     end
                  end
               pre_RS2:
                     if(num != SN) begin C2 = num; R_STATE <= RS2; end
               RS2:begin
                     if(num == SN) begin
                        display[23:20] = SN;
                        display[19:16] = SN;
                        display[15:12] = SN;
                        display[11:8]  = SS;
                        display[7:4]   = C3;
                        display[3:0]   = C2;
                        R_STATE <= pre_RS3;
                     end
                  end
               pre_RS3:
                     if(num != SN) begin C1 = num; R_STATE <= RS3; end
               RS3:begin
                     if(num == SN) begin
                        display[23:20] = SN;
                        display[19:16] = SN;
                        display[15:12] = SS;
                        display[11:8]  = C3;
                        display[7:4]   = C2;
                        display[3:0]   = C1;
                        R_STATE <= pre_RS4;
                     end
                  end
               pre_RS4:
                     if(num != SN) begin C0 = num; R_STATE <= RS4; end
               RS4:begin
                     if(num == SN) begin
                        display[23:20] = SN;
                        display[19:16] = SS;
                        display[15:12] = C3;
                        display[11:8]  = C2;
                        display[7:4]   = C1;
                        display[3:0]   = C0;
                        R_STATE <= R_Confirm;
                     end
                  end
               R_Confirm:begin
                     if(num == SS) begin
                        R_STATE <= pre_RS0;
                        Current_State <= pre_initial_State;
```

위에서 pre_RS1과 같이 pre state를 둔 것은, 입력이 들어왔을 때, key matrix에서 손을 떼는 순간에 상태변화를 시키기 위하여 삽입된 state이다. 처음에 이 pre state들 없이 설계를 하였을 때, key matrix를 한번 누르는것 만으로 4자리의 모든 숫자가 입력되었었다. 클락의 단위가 ps나 길어야 ns이기 때문에 살짝 눌렀다 떼는 그 사이에 4번의 입력이 들어가고, 4번의 state transition이 일어나게 되었던 것이다.

② initial_State

초기단계로써 display에 아무것도 출력을 하지 않으며 * 또는 #가 들어올 경우 try_State로 상태전이 하도록 설계하였다.

③ try_State

try_State는 reset_State에서 비밀번호를 결정하고, * 또는 #가 입력되었을 경우 비밀번호를 입력하고, 기존의 비밀번호와 일치하는가를 판별하는 단계이다. 비밀번호 입력모드와 비밀번호 변경모드 모두 공통으로 처음에는 비밀번호를 입력하여 맞는지 확인하는 단계가 있으므로 하나의 state로 통합하였다. 그리고 그 결과인 CORRECT와 INCORRECT에 따라서 INCORRECT인 경우는 무조건 처음 입력상태로 돌아가고(pre_TS0), CORRECT인 경우는 * 를 입력받고 들어왔을 경우 initial_State로, #를 입력받고 들어왔을 경우는 reset_State로 상태가 전이되도록 설계하였다. reset_State와 마찬가지로 입력값이 한번에 입력되는 것을 방지하기 위하여 pre state를 추가하였다.

④ blink_State

```
blink_State:begin
      cnt = cnt + 1;
      if(cnt >= 0 && cnt < 409600) begin
         display[23:20] = SN;
         display[19:16] = SN;
         display[15:12] = SN;
         display[11:8]  = SN;
         display[7:4]   = SN;
         display[3:0]   = SN;
      end
      else if(cnt >= 409600 && cnt < 819200) begin
         display[23:20] = SN;
         display[19:16] = sel;
         display[15:12] = digit3;
         display[11:8]  = digit2;
         display[7:4]   = digit1;
         display[3:0]   = digit0;
      end
      else if(cnt >= 819200 && cnt < 1228800) begin
         display[23:20] = SN;
         display[19:16] = SN;
         display[15:12] = SN;
         display[11:8]  = SN;
         display[7:4]   = SN;
         display[3:0]   = SN;
      end
      else begin
         cnt = 0;
         T_STATE <= pre_TS0;
         if(sel == SA) Current_State <= pre_initial_State;
         else if(sel == SS) Current_State <= reset_State;
      end
   end
```

왼쪽과 같이 try_State상태 중 blink_State라는 상태가 존재한다. 이 특수한 케이스는 try_State에서 입력한 4자리 암호가 옳은 암호일 경우 display되어있는 숫자들을 깜빡이도록 하는 state이다. 주어진 변수를 사용하여 blink=1로 두고 해보려고 하였지만, 어찌된 일인지 깜빡이지 않고 계속 켜져 있었다. 그래도 깜빡임을 구현해야 했으므로 일정한 시간동안 모든 display가 꺼졌다, 또 일정한 시간동안 켜졌다가 다시 꺼지도록 설계를 하였다.

5. FPGA동작 결과

내가 설계한 digital door lock은 기대했던 상태전이도 그대로 잘 작동을 하였다. 처음 전원을 켰을 경우 reset_State에 의해 네자리의 숫자를 입력받는다. 그리고 * 또는 #에 따라 비밀번호 입력/변경모드로 들어간다. 입력모드에서 잘못된 입력을 하였을 경우 새로운 입력을 시도받는 상태로 잘 넘어갔고, 옳은 입력을 받았을 경우 숫자가 깜빡인 후 초기상태로 되돌아감을 확인할 수 있었다. 또한 #를 입력하고 올바른 비밀번호를 입력하여 비밀번호 변경모드로 들어간 경우 새로운 비밀번호를 입력해 주고, 다시 그 새로운 비밀번호가 제대로 잘 작동하는지 확인 할 수 있었다.

6. 결과 분석 및 토의, 고찰할 점

결과적으로 내가 설계한 digital door lock은 정상적으로 작동을 하였다. 설계를 하면서 가장 어려웠던 점은 “딸깍” key matrix를 한번 누르는 순간이 FPGA의 클럭속도 입장에서는 수없이 많은 입력이 연속되는 것으로 인식된다는 것이었다. 이를 해결하기 위해서 가장 시급했던 것이 아무 버튼도 누르지 않는 상태를 인식시키는 것이었고, 주어진 key_matrix.v에는 그 상태가 구현되어 있지 않아 새로 추가를 하는 부분이 가장 많은 시간이 걸렸었다. 또 다른 방법으로는 wait 명령어를 사용해 보기도 하였는데, 동작결과가 그때그때 달라서 쓰지 않기로 하였다.

한가지 고민해볼 점은 왜 blink가 제대로 동작하지 않았는가 이다. 추측하건데 digital door lock의 clock과 seven_segment_display의 clock이 공통으로 slow_clk를 사용하고 있어서 그럴 수 있다고 생각한다. 또한 한번 버튼을 누를 때 여러개의 입력이 되지 않도록 pre state를 두었음에도 불구하고, 한번의 입력에 의해 그 똑같은 입력이 2번, 3번되는 경우가 자주 있었다. 이는 FPGA key matrix의 물리적 결함으로 인해 한번 누름이 여러번 끊어서 누른 것으로 인식되었기 때문이라 생각된다.

7. 시연동영상

별도첨부