Simple Door Lock Verilog-Based Project

Team no.16

Team Members:

- Muhammad Sayed Abdel-Salam.
- Muhammad Sami Ahmad.
- Hager Samir Ibrahim.
- Youssef Ahmad Shawky.

Objective & Abstract Description of the Project

- We are implementing a simple door lock system for security purposes. Such system includes the following functionalities:
    - The password is a 3-digit hexadecimal number, where each digit is in a range from 0 to F.

    - The digits are inputted in binary through switches SW[3:0], the confirmation signal for reading in a digit is generated by KEY[0]. The current value that the switches represent is shown on HEX4 as labeled in the attached block diagram.

    - The password is passed to the processing state machine when there is a confirmation signal from KEY[1]. The value of the inputted password is shown on HEX2 through HEX0, showing password can be enabled/disabled using switch SW[4].

    - The design can operate in 3 different modes: **SET**, **VERIFY** and **FREEZE**. The first 2 modes can be interchange using KEY[2]. The current operating mode will be shown on HEX7.

    - In **SET** mode, the user can set a password for the lock. After pressing KEY[1] to confirm the password, it will be saved, and the mode will change to **VERIFY**.

- In **VERIFY** mode, pressing KEY[1] will put the current inputted password into a check. If the inputted password is identical to the saved one set earlier, a LEDG will blink with **5Hz** frequency, otherwise a LEDR will blink with **1Hz** frequency. The user has **3** trials to check the password, the number of trials will be reset if the password is correct. When it reaches 0, the circuit will enter a **FREEZE** state. The number of trials will also be shown on HEX6.

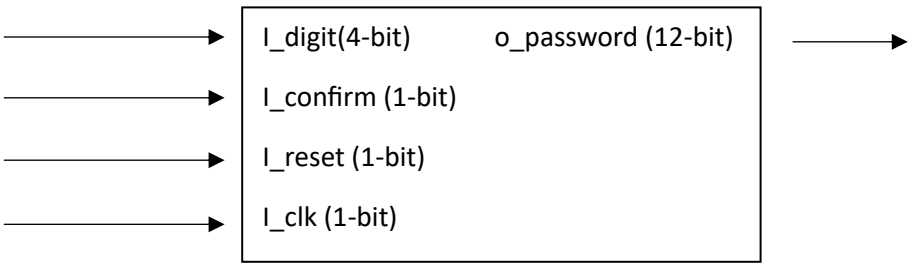
- In the **FREEZE** state, only the switches work, all keys mentioned above are disabled. Therefore, the state machine won't work.

- Hard reset signal can be generated by KEY[3]. When the circuit is in the FREEZE state, only KEY[3] will work and it will reset the whole circuit.

Circuit Working Demonstration

- We are going to consider each block of these blocks separately and after that we are going to consider the top view of the whole circuitry.

Password Getter



- It's a unit responsible for taking the user input password. Such function is attained by providing the password 3-digits digit by digit in the form of 4 binary bits through some sort of switches SW[3:0]. After providing each digit we confirm for the unit to start left shift and store them in the internal register which is directly connected to the output wire which in turn connected to FSM unit and password decoder unit.
- A code fragment to show how such function is attained:

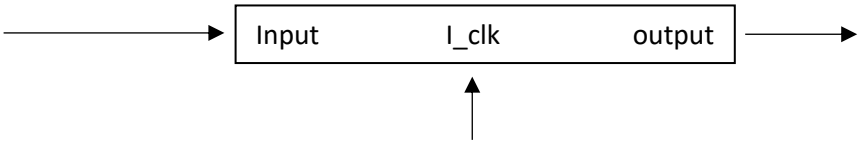
```
i_digit = 4'hB;
i_confirm_getter = 1;
i_confirm_getter = 0;
i_digit = 4'hA;
i_confirm_getter = 1;
i_confirm_getter = 0;
i_digit = 4'hD;
i_confirm_getter = 1;
i_confirm_getter = 0;
```

- Such a unit is synchronized and can also be reset through the hard reset input KEY[3] which is common between it and the FSM unit.

  The source code of such unit exists in **Password_Getter.v** file.

  The testbench code of such unit exists in **Password_Getter_tb.v** file.

Positive Edge Detector

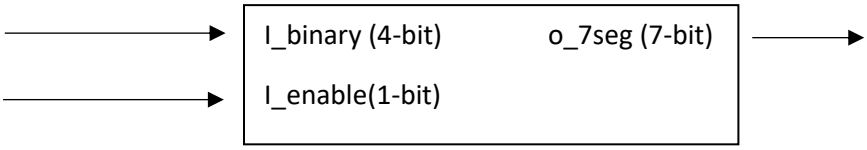


- It's aptly named, all its function is to detect the positive edge of the clock and based on that it buffers its inputs.

  All the inputs of the system are buffered by this unit except for the **i_digit.**

  The source code for such unit exists in **Posedge_Detector.v** file.

7-Segment Display



- Such a unit is used to display the input hexadecimal number. It consists of 7 segments, each one of them is active low, meaning it lights up when its logic is 0.
- Based on its input binary It start to deactivate the segement contributing to displaying such hexadecimal number by putting logic 0 on them.

- Code fragment demonstrating how such function is attained:

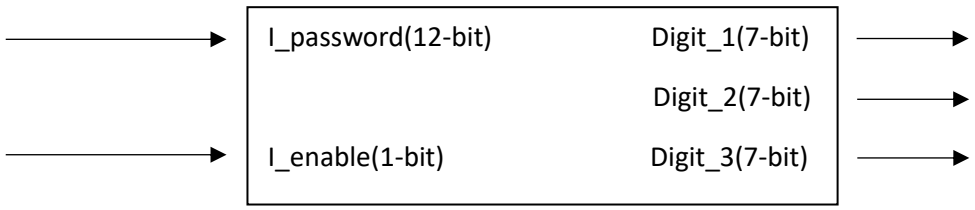
```
case (i_binary)
    // Cases for each possible value of i_binary
    4'h0: value_7seg = 7'b1000000;  // Assign value for binary 0
    4'h1: value_7seg = 7'b1111001;  // Assign value for binary 1
    4'h2: value_7seg = 7'b0100100;  // Assign value for binary 2
    4'h3: value_7seg = 7'b0110000;  // Assign value for binary 3
    4'h4: value_7seg = 7'b0011001;  // Assign value for binary 4
    4'h5: value_7seg = 7'b0010010;  // Assign value for binary 5
    4'h6: value_7seg = 7'b0000010;  // Assign value for binary 6
    4'h7: value_7seg = 7'b1111000;  // Assign value for binary 7
    4'h8: value_7seg = 7'b0000000;  // Assign value for binary 8
    4'h9: value_7seg = 7'b0010000;  // Assign value for binary 9
    4'hA: value_7seg = 7'b0001000;  // Assign value for binary A
    4'hB: value_7seg = 7'b0000011;  // Assign value for binary B
    4'hC: value_7seg = 7'b1000110;  // Assign value for binary C
    4'hD: value_7seg = 7'b0100001;  // Assign value for binary D
    4'hE: value_7seg = 7'b0000110;  // Assign value for binary E
    4'hF: value_7seg = 7'b0001110;  // Assign value for binary F
    default: value_7seg = 7'b1111111;  // Assign default value for other cases
endcase
```

- Such unit is used at the input of the Password Getter to display the input hexadecimal digit. In addition, it's being used in the output of the FSM unit to display the number of trials and the state of operation.
- It can be disenabled using the **I_enable** input.

  The source code of such unit exists in **led7_decoder.v** file.

  The testbench of such unit exists in **t_led7_decoder.v** file.

Password decoder



- Such unit acts as three 7-segment display units integrated with each other, which deals with the whole password instead of some digits of it. So, we are making use of the **led7_decoder** module to display each 4-bit slice of the password.

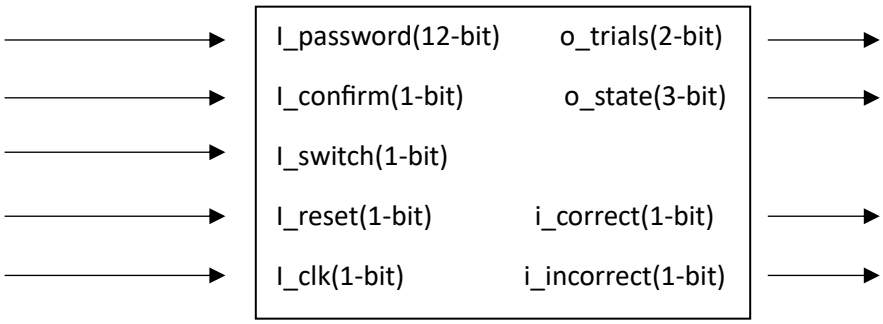
- Code fragment demonstrating how such function is attained:

```
led7_decoder D1(
    .i_en(i_en),
    .i_binary(i_password[3:0]),
    .o_7seg(o_7seg_0)
);
led7_decoder D2(
    .i_en(i_en),
    .i_binary(i_password[7:4]),
    .o_7seg(o_7seg_1)
);
led7_decoder D3(
    .i_en(i_en),
    .i_binary(i_password[11:8]),
    .o_7seg(o_7seg_2)
);
```

The source code of such unit exists in **pass_detector.v** file.

The testbench of such unit exists in **t_pass_detector.v** file.

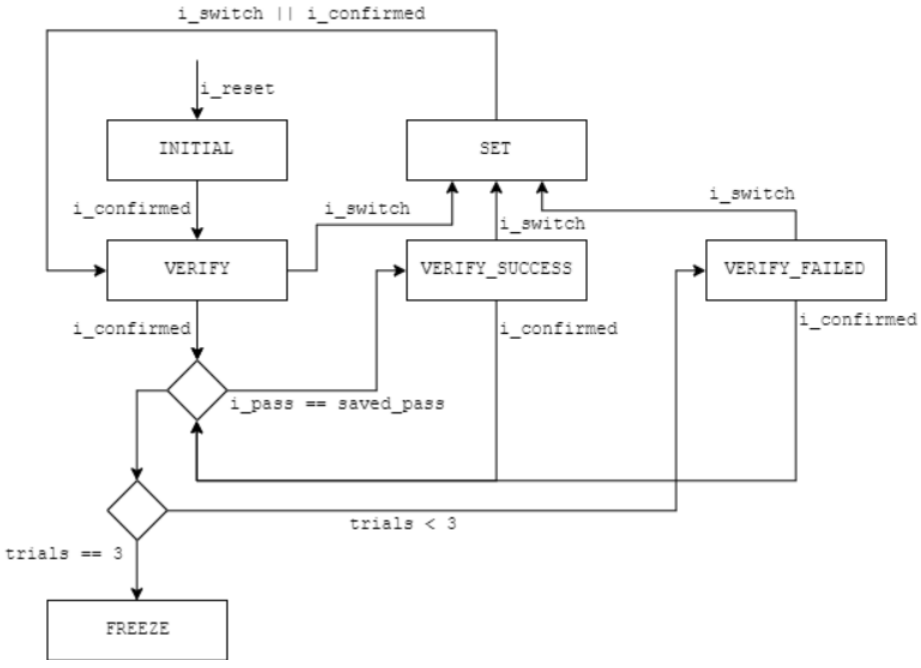
FSM



- Such unit is a synchronous mealy finite state machine, this is where all the logic of the design is processed.
- Such system has 6 states of operation, which are declared as follows:

```
localparam [2:0] INITIAL_SET = 3'b000;
localparam [2:0] VERIFY = 3'b001;
localparam [2:0] VERIFY_SUCCESS = 3'b010;
localparam [2:0] VERIFY_FAILED = 3'b011;
localparam [2:0] SET = 3'b100;
localparam [2:0] FREEZE = 3'b101;
```

- Based on the current state of operation, the flow of the program changes. The following diagram shows the flow of the program and how the state changes with the inputs:



- From the above state machine flow graph, we can deduce some logic:
  - On resetting, the system starts at the **INITIAL_SET** state, which requires from the user to input a security password, which is received by the **Password Getter** unit and then hand overed to the **FSM** unit which waits for a confirm using the **I_confrim** to save the input password as the security password.

  - After that, the system automatically switches to the **VERIFY** state, which requires from the user to reenter the security password, to give him access to the system. The user follows the same procedure of inputting the password of verification as that for **INITIAL_SET** password setting.

  - If the input password is identical to that of the security password, the state automatically switches to the **VERIFY_SECCESS** making the **i_correct** high which is interpreted by the **Led blinker** unit with **5 Hz** led fluctuation.

  - If the input password is incorrect and the number of trials below **3**, the state will automatically switch to **VERIFY_FAIL**, making **i_incorrect** high which is interpreted by the **Led blinker** unit with **1 Hz** led fluctuation.

  - If the state is at either **VERIFY**, **VERIFY_SECCESS** or **VERIFY_FAIL**, making **i_switch** high will result in switching the user to the **SET** state only, which does the same thing **INITIAL_SET** state does but without **i_reset**, as a way of changing the security password.

  - If the state is at **SET**, the **I_switch** will result in switching the user to only the **VERIFY** state.

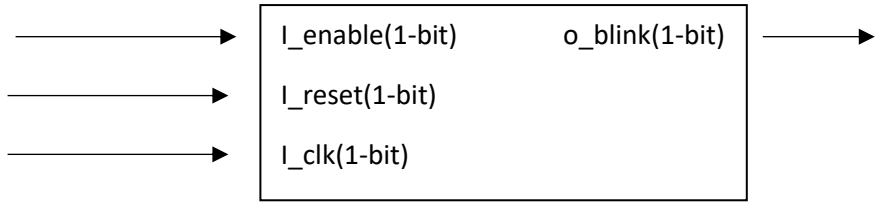
  - If the input password is incorrect for the third time in order, the state will switch automatically to **FREEZE** state, which makes the whole system unresponsive except for the **i_reset**, which brings us to the **INITIAL_SET** state.

- On each positive edge of the clock, the FSM refreshes to display the current state and current number of trials if exists using the **7-Segment Display** discussed previously.

  The source code for such a unit exists in **FSM_DL.v** file.

  The testbench for such unit exists in **FSM_DL_tb.v** file.

Led Blinker



- The function of such a unit is to indicate the success or failure of verification of the security password visually by Led blinking.
- If such unit has received logic 1 from **o_correct** port of the **FSM** unit, which indicated by the **I_enable** input on the **Led Blinker** unit, it will blink in 5 Hz frequency indicating success has been achieved.
- If such unit has received logic 1 from the **o_incorrect** port of the **FSM** unit, which indicated by the I_enable input on the **Led Blinker** unit, it will blink in 1 Hz frequency indicating failure has happened.
- The I_correct and I_incorrect ports of the FSM unit are connected to its specific programmed Led blinker, so that it attains the above function.

- Code fragment showing the method of instantiation of such unit for each port:

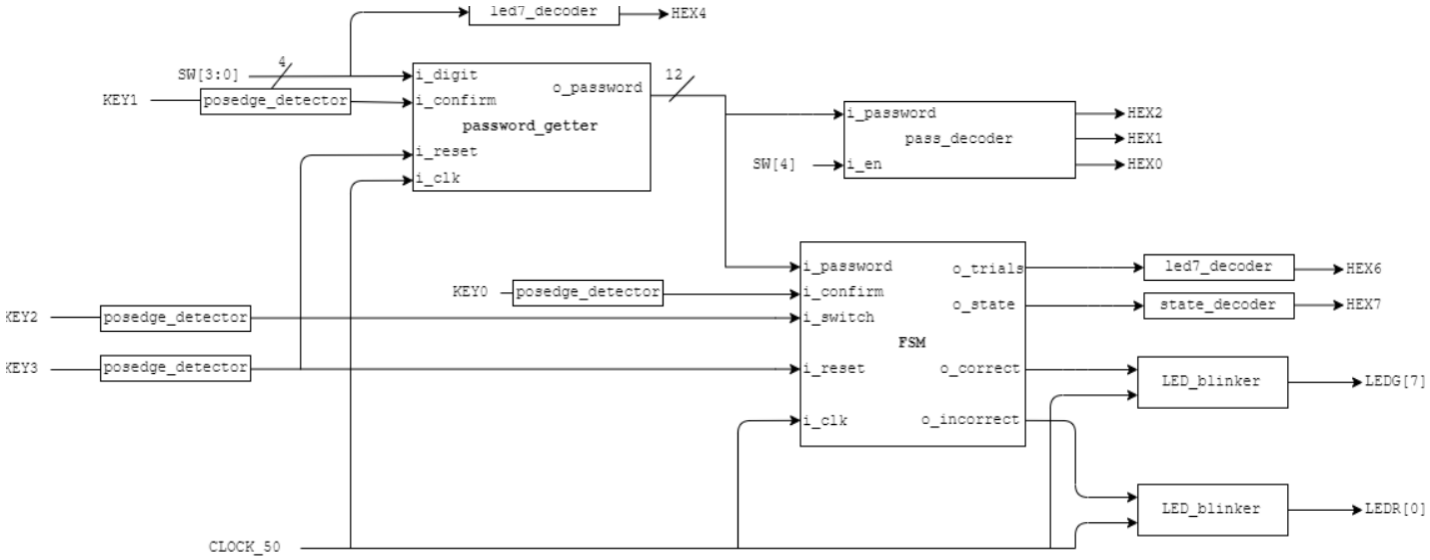
```
LED_blinker #(.CLK_IN(CLK_IN), .FREQ_OUT(1)) LED_R(
    .i_clk(i_clk),
    .i_en(incorrect),
    .i_reset(i_hard_reset),
    .o_blink(o_incorrect_led)
);

LED_blinker #(.CLK_IN(CLK_IN), .FREQ_OUT(5)) LED_G(
    .i_clk(i_clk),
    .i_en(correct),
    .i_reset(i_hard_reset),
    .o_blink(o_correct_led)
);
```

- The logic of such a unit depends on the method of frequency division.

  The source code for such a unit exists in **Led-Blinker.v** file.
  The testbench for such a unit exists in **Led-Blinker-tb.v** file.

Top View of The System

- The final touch seems to happen here, where we have interconnected all the above units to produce the password security system of interest.
- By achieving the compatibility of the outputs of each unit with the inputs of the successive ones. We get such output:



The source code for the top view exists in **DL_top.v** file.

The testbench for the top view exists in **DL_top_tb.v** file.

Challenges

- We have encountered a bunch of problems along the way which were a stem for a lot of benefits. Considering the most remarkable ones:

    - Dealing with Modelsim IDE may represent the most annoying obstacle any one of the team members has encountered.
        - Unfortunately, we have been forced to adapt to it and try to go over and discover how to perform each teeny tiny regarding the simulation of the waves adopting the learning approach try and error.
    - Designing the state machine diagram which represents all the logic of the project.
        - After a bit of time, we could have been able to imagine and implement such logic.
    - Bugs regarding the testbench of the **top_DL** module, where we have used incompatible parameters for the modules of each unit.
        - After an extensive time of debugging, we could find the cause of the error.