FPGA VHDL Digital Lock Implementation

Mohammadhossein Keyvanfar

Logical Circuits Lab Course - 14041 Semester

Instructor: Ms. M. Gholami

Department of Computer Engineering, Faculty of Engineering, University of Guilan

2025-12-13

Table of Contents

1. Project Assignment 2
2. Overview 3
3. Top-level entity and ports 4
4. Types, Constants, and Top-level Signals .. 6
5. Outputs Assignment 8
6. Finite State Machine Design 9
   - 6.1 state_update_proc Process 10
   - 6.2 seg_slots_next_set_proc Process 11
   - 6.3 set_transitions_proc Process 13
   - 6.4 State Diagram 14
7. Processes and their Responsibilities 15
   - 7.1 Display Refresh Process (refresh_prc) . 16
   - 7.2 Button Synchronizer (sync_proc) 18
   - 7.3 Register Current State (state_update_proc) 19
   - 7.4 Next State Logic (set_transitions_proc) 20
   - 7.5 Store Next Segment Indexes (seg_slots_reg_update_proc) 21
   - 7.6 One-second Counter (counter_state) . 22
   - 7.7 Compute Next Segments (seg_slots_next_set_proc) 24
   - 7.8 Password Comparator (pw_check_proc) 25
   - 7.9 Animated Light Show (light_show_proc) 26
   - 7.10 Blink on Wrong Password (wrong_pw_seg_blink_proc) 28
8. Animation Logic 29

1. Project Assignment

The digital lock is implemented on an FPGA using VHDL to meet the following requirements:

1. Upon powering up the board, the last two digits of Student Number A and the last two digits of Student Number B must be displayed on the four 7-segment displays.
2. After pressing a designated key, the system counts from 0 to 15 using a 1-second clock rhythm, displaying the two-digit value on the 7-segment displays.
3. A 4-bit password is entered using four input switches. When the password is confirmed by pressing a specific key, if it matches the stored key, the four 7-segment displays show the word “PASS.” If the password is incorrect, all board LEDs and displays blink.
4. While in “PASS” mode, holding down a designated push button animates the word “PASS” across the displays or moves it from one side to the other.

2. Overview

This design implements a small state-machine controlled display module for a 4-digit common-anode 7-segment display. This design has:

- Four 7-segment digits (`seg`, `ad`) that are scanned at `TOTAL_SCAN_HZ` (multiplexing).
- Four push buttons (`ts`) and a 4-bit password input (`pw_in`).
- Four States: show student number, count 0–15, enter password, light show.
- Shows “PASS” on correct password with animation triggered by a holding push button or blink all four 7-segments on wrong password input.

3. Top-level entity and ports

Entity: `LCL_Project1_14041`

Generics:

- `CLK_FREQ`: default set to 24_000_000 Hz (24MgHz) to match Posedge LogiX1 megawing board - Rev 1.2 on-board clock.
- `TOTAL_SCAN_HZ`: default set to 4000 Hz for multiplex refresh timing.

Ports:

- `clk : in std_logic`: board clock source.
- `ts : in std_logic_vector(3 downto 0)`: four push buttons used for mode selection and control (active-low).
- `pw_in : in std_logic_vector(3 downto 0)`: 4-bit parallel password switches.
- `seg : out std_logic_vector(6 downto 0)`: 7bit 7-segment outputs (a..g) encoded by `seg_display` constant.
- `ad : out std_logic_vector(3 downto 0)`: 4-bit digit-enable pattern (`ad_pattern`) for multiplexing, −1 index in `ad_pattern` is used to blank the display during blinking.

3. Top-level entity and ports (cont.)

```vhdl
entity LCL_Project1_14041 is
  generic (
    CLK_FREQ      : integer := 24_000_000;
    TOTAL_SCAN_HZ : integer := 4000
  );

  port (
    clk   : in  std_logic;
    ts    : in  std_logic_vector(3 downto 0);
    pw_in : in  std_logic_vector(3 downto 0);
    seg   : out std_logic_vector(6 downto 0);
    ad    : out std_logic_vector(3 downto 0)
  );
end entity;
```

4. Types, Constants, and Top-level Signals

Custom Types

- `type state is (student_number, counter, enter_pw, lightshow);` - main FSM states.
- `type seg_array is array (0 to 30) of std_logic_vector(6 downto 0);` - 7-seg patterns.
- Small arrays used to hold the multiple slot indexes (`array2d_4to5`, `array2d_5to4`).

Important Constants

- `seg_display` (0..30) - maps an integer index to a 7-segment pattern (0-9 plus several light-show patterns and letters for “PASS”).
- `DIV_TICKS := integer(CLK_FREQ / TOTAL_SCAN_HZ)` - number of clock cycles per scan step.
- `light_show_hz := integer(CLK_FREQ / 2)`
  `blinker_hz := integer(CLK_FREQ / 2)` - counters for blinking/light-show timing.

4. Types, Constants, and Top-level Signals (cont.)

Key Signals

- `cr_state`, `next_state` - current and next FSM states.
- `refresh_cnt`, `active_slot` - digit-scan refresh counter and active digit index.
- `seg_slots_reg`, `seg_slots_next` - the 4 slot values (each 5 bits) that index seg_display.
- `sec_count`, `counter_val` - for 1-second counter mode.
- `lightshow_count`, `lightshow_step` - for animated light show.
- `blinker_count`, `blinker_state` - for wrong-password blink behavior.
- `correct_pw` - indicates whether `pw_in` matches hardcoded pw.
- `ts_sync_0`, `ts_sync_1` - synchronizers for push buttons.

5. Outputs Assignment

```vhdl
seg <= seg_display(30) when active_slot = -1 else
       seg_display(to_integer(unsigned(seg_slots_reg(active_slot))));
ad  <= ad_pattern(active_slot);
```

- `seg` is driven by looking up the 5-bit value in `seg_slots_reg`(`active_slot`) (an index 0..30) and converting it into a 7-segment pattern using `seg_display`, if `active_slot = -1` then `seg` will be driven by 30th index of `seg_display` which is a simple line '`-`'.
- `ad` is a direct lookup into `ad_pattern`, using `active_slot` to drive which digit is active.

6. Finite State Machine Design

The system behavior is mapped directly to an FSM with the following states:

- `student_number`: the last two digits of two student numbers are shown across the four digits.
- `counter`: a two-digit counter counts 0..15 with a 1-second period and displays the current value.
- `enter_pw`: the 4-bit password is sampled. upon confirm, a match results in “PASS” being displayed, otherwise blink behavior is generated.
- `lightshow`: while PASS is active and a push button is held, the PASS pattern animates across the digits.

The Finite State Machine (FSM) is implemented using three processes:

6.1 state_update_proc Process

1. A synchronous process called **state_update_proc** that captures the **next_state** into **cr_state** on the rising edge of the clock (**clk**).

```vhdl
state_update_proc: process (clk) is
  begin
    if rising_edge(clk) then
      cr_state <= next_state;
    end if;
  end process;
```

6.2 seg_slots_next_set_proc Process

2. A combinational process named **seg_slots_next_set_proc** that sets the **seg_slots_next** values (each representing a 7-segment display value) based on the current state (**cr_state**).

```vhdl
seg_slots_next_set_proc: process (cr_state, counter_val, correct_pw, lightshow_step)
    variable counter_v   : integer;
    variable lightshow_s : integer;
  begin
    case cr_state is
      when student_number => -- display 1100
        seg_slots_next(3) <= "00000";
        seg_slots_next(2) <= "00000";
        seg_slots_next(1) <= "00001";
        seg_slots_next(0) <= "00001";
      when counter =>
        counter_v := counter_val; -- integer range 0 to 15
        seg_slots_next(3) <= std_logic_vector(to_unsigned(counter_v mod 10, 5));
        seg_slots_next(2) <= std_logic_vector(to_unsigned((counter_v / 10) mod 10, 5));
```

6.2 seg_slots_next_set_proc Process (cont.)

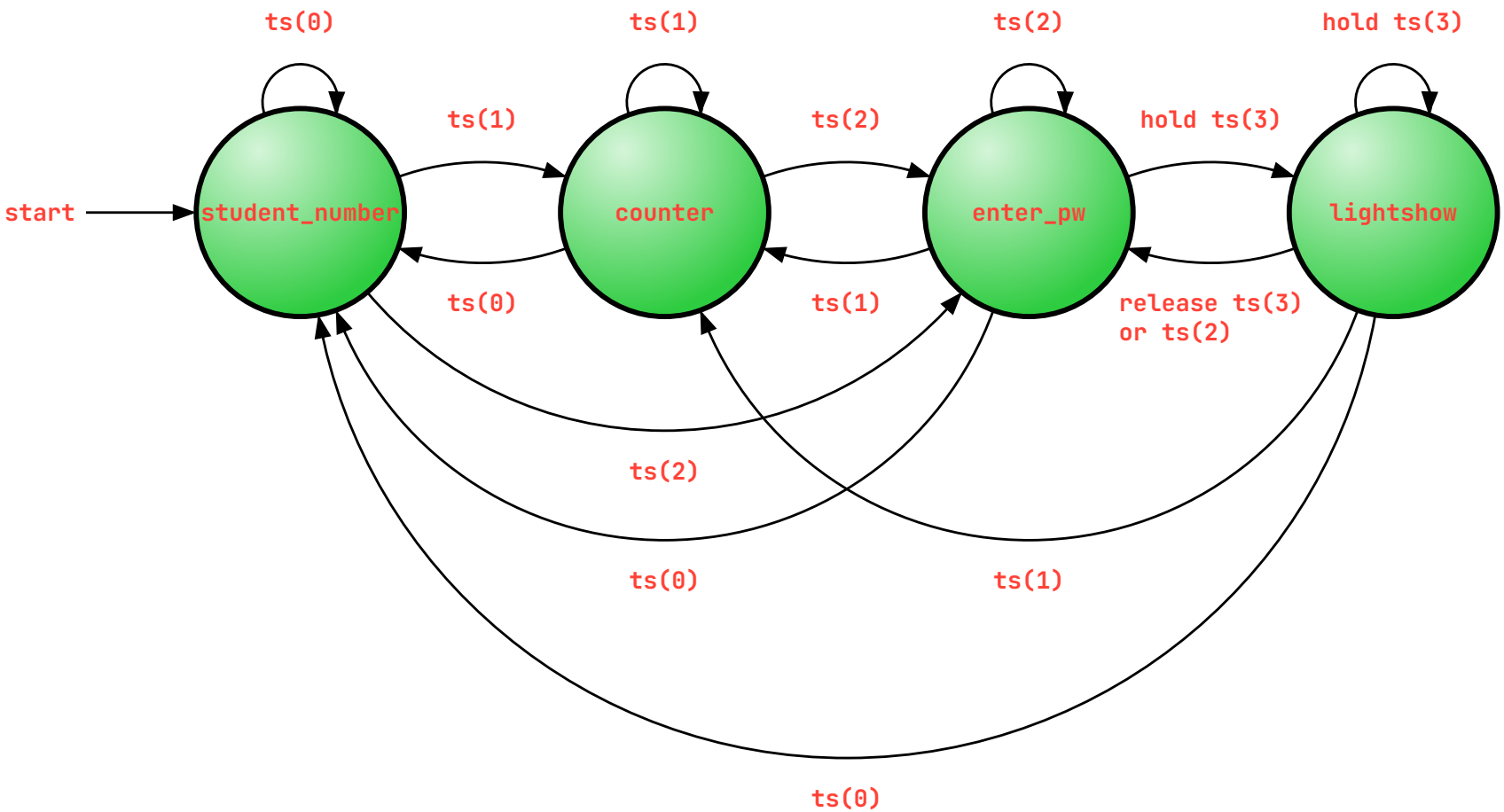
```
15          seg_slots_next(1) <= std_logic_vector(to_unsigned(0, 5)); -- two leftmost 7seg are 0
16          seg_slots_next(0) <= std_logic_vector(to_unsigned(0, 5)); -- two leftmost 7seg are 0
17        when enter_pw =>
18          if correct_pw = '1' then -- show "PASS"
19            seg_slots_next(3) <= std_logic_vector(to_unsigned(24, 5)); -- "S"
20            seg_slots_next(2) <= std_logic_vector(to_unsigned(24, 5)); -- "S"
21            seg_slots_next(1) <= std_logic_vector(to_unsigned(19, 5)); -- "A"
22            seg_slots_next(0) <= std_logic_vector(to_unsigned(14, 5)); -- "P"
23          else -- show "-"
24            seg_slots_next <= (others => std_logic_vector(to_unsigned(30, 5))); -- "-"
25          end if;
26        when lightshow =>
27          lightshow_s := lightshow_step; -- integer range 0 to 4 for five different digit "frame"
28          seg_slots_next(3) <= std_logic_vector(to_unsigned(lightshow_s + 25, 5));
29          seg_slots_next(2) <= std_logic_vector(to_unsigned(lightshow_s + 20, 5));
30          seg_slots_next(1) <= std_logic_vector(to_unsigned(lightshow_s + 15, 5));
31          seg_slots_next(0) <= std_logic_vector(to_unsigned(lightshow_s + 10, 5));
32      end case;
33    end process;
```

6.3 set_transitions_proc Process

3. Another synchronous process referred to as **set_transitions_proc**, which determines the **next_state** transitions based on the current state (**cr_state**) and the synchronized input from the push buttons (**ts_sync_1**) and (**ts**).

```vhdl
set_transitions_proc: process (clk, cr_state, ts)
  begin
    if rising_edge(clk) then
      if ts_sync_1(0) = '0' then next_state <= student_number;
      elsif ts_sync_1(1) = '0' then next_state <= counter;
      elsif ts_sync_1(2) = '0' then next_state <= enter_pw;
      elsif ts = "0111" then -- Holding ts(3)
        if cr_state = enter_pw and correct_pw = '1' then next_state <= lightshow;
        elsif cr_state = lightshow and correct_pw = '0' then next_state <= enter_pw;
        end if;
      elsif ts = "1111" and cr_state = lightshow then  next_state <= enter_pw; -- Released ts(3) in lightshow state
      end if;
    end if;
  end process;
```

6.4 State Diagram



7. Processes and their Responsibilities

The design is organized into a set of clearly scoped processes. Each process responsibility is described in it’s subsection.

7.1 Display Refresh Process (refresh_prc)

Multiplex refresh and blanking when blinking

- Synchronous to `clk`. Uses `refresh_cnt` to pace digit scanning at `TOTAL_SCAN_HZ`.
- `active_slot` cycles 0 → 1 → 2 → 3 to multiplex digits. If `blinker_state = '1'`, the code sets `active_slot <= -1` and `refresh_cnt <= 0`, which leads to blank the display during blink.

7.1 Display Refresh Process (refresh_prc) (cont.)

```vhdl
refresh_prc: process (clk, blinker_state)
    variable blink_s : std_logic;
  begin
    if rising_edge(clk) then
      blink_s := blinker_state;
      if blink_s = '1' then
        active_slot <= - 1;
        refresh_cnt <= 0;
      else
        if refresh_cnt < DIV_TICKS - 1 then
          refresh_cnt <= refresh_cnt + 1;
        else
          refresh_cnt <= 0;
          if active_slot = 3 then
            active_slot <= 0;
          else
            active_slot <= active_slot + 1;
          end if;
        end if;
      end if;
    end if;
  end process;
```

7.2 Button Synchronizer (sync_proc)

Two-stage synchronizer is implemented for asynchronous push button inputs (`ts`). The second stage output (`ts_sync_1`) is the only signal used in combinational decision logic.

- Captures `ts(2 downto 0)` into `ts_sync_0` then `ts_sync_1` one clock later.

```vhdl
sync_proc: process (clk)
  begin
    if rising_edge(clk) then
      ts_sync_0 <= ts(2 downto 0); -- capture user push buttons input except ts(3) because it is used as push button not as a single pulse (toggle) so it doesn't need synchronization
      ts_sync_1 <= ts_sync_0; -- stable in clk domain
    end if;
  end process;
```

7.3 Register Current State (state_update_proc)

A single clocked process updates `cr_state` from `next_state` on the rising edge of `clk`.

```vhdl
state_update_proc: process (clk) is
  begin
    if rising_edge(clk) then
      cr_state <= next_state;
    end if;
  end process;
```

7.4 Next State Logic (set_transitions_proc)

- Uses `if rising_edge(clk) then` and checks `ts_sync_1` bits to set `next_state`.
- Reads `ts` (unsynchronized) with patterns “0111” (for holding `ts(3)`) and “1111” on lightshow state (for releasing `ts(3)`).

```vhdl
set_transitions_proc: process (clk, cr_state, ts)
  begin
    if rising_edge(clk) then
      if ts_sync_1(0) = '0' then next_state <= student_number;
      elsif ts_sync_1(1) = '0' then next_state <= counter;
      elsif ts_sync_1(2) = '0' then next_state <= enter_pw;
      elsif ts = "0111" then -- Holding ts(3)
        if cr_state = enter_pw and correct_pw = '1' then next_state <= lightshow;
        elsif cr_state = lightshow and correct_pw = '0' then next_state <= enter_pw;
        end if;
      elsif ts = "1111" and cr_state = lightshow then  next_state <= enter_pw; -- Released ts(3) in lightshow state
      end if;
    end if;
  end process;
```

7.5 Store Next Segment Indexes (seg_slots_reg_update_proc)

A synchronous process that updates `seg_slots_reg` based on `seg_slots_next` on each rising edge of `clk`.

- `seg_slots_reg <= seg_slots_next;` implements double-buffering, a combinational process computes `seg_slots_next` based on current state, then the register latches it each clock.

```vhdl
seg_slots_reg_update_proc: process (clk)
  begin
    if rising_edge(clk) then
      seg_slots_reg <= seg_slots_next;
    end if;
  end process;
```

7.6 One-second Counter (counter_state)

A synchronous process that increments `counter_val` (range 0 to 15) by 1 each second.

- When `cr_state = counter` it counts `sec_count` up to `CLK_FREQ`, then increments `counter_val`.
- If not in `counter` state, `sec_count` is reset to 0.

7.6 One-second Counter (counter_state) (cont.)

```vhdl
counter_state: process (clk, cr_state) is
begin
  if cr_state = counter then
    if rising_edge(clk) then
      if sec_count < CLK_FREQ - 1 then
        sec_count <= sec_count + 1;
      else
        sec_count <= 0;
        if counter_val = 15 then counter_val <= 0;
        else counter_val <= counter_val + 1;
        end if;
      end if;
    end if;
  else
    sec_count <= 0;
  end if;
end process;
```

7.7 Compute Next Segments (seg_slots_next_set_proc)

- This is the combinational driver for what digits should show for each state.
- For `student_number` it sets (hardcoded) slots to show last two digits of student A's student number and last two digits of student B's student number on four 7-segments.
- For `counter` it uses `counter_val` to compute tens/units and places them into 7-segment slots.
- For `enter_pw` it shows either the "PASS" pattern (indices 14,19,24,24) `if correct_pw = '1'` or the `blank pattern (index 30)` for all slots if wrong.
- For `lightshow` it maps `lightshow_step` (range 0 to 5) into offsets to select patterns from `seg_display`.

For the code of this process see Section 6.2.

7.8 Password Comparator (pw_check_proc)

- Synchronously compares `pw_in` to constant `pw` each rising edge and sets `correct_pw`.

```vhdl
pw_check_proc: process (clk)
  begin
    if rising_edge(clk) then
      if pw_in = pw then
        correct_pw <= '1';
      else
        correct_pw <= '0';
      end if;
    end if;
  end process;
```

7.9 Animated Light Show (light_show_proc)

- When in `lightshow` state, counts `lightshow_count` up to `light_show_hz` then increments or decrements `lightshow_step` to create oscillating animation.
- Uses variable `lightshow_step_dir` to reverse direction when animation ends and rewind animation.

```vhdl
light_show_proc: process (clk, cr_state)
    variable lightshow_step_dir : std_logic := '0';
  begin
    if cr_state = lightshow then
      if rising_edge(clk) then
        if lightshow_count < light_show_hz - 1 then
          lightshow_count <= lightshow_count + 1;
        else
          lightshow_count <= 0;
          if lightshow_step = 4 then
```

7.9 Animated Light Show (light_show_proc) (cont.)

```
11              lightshow_step_dir := '1';
12            elsif lightshow_step = 0 then
13              lightshow_step_dir := '0';
14            end if;
15            if lightshow_step_dir = '0' then
16              lightshow_step <= lightshow_step + 1;
17            else
18              lightshow_step <= lightshow_step - 1;
19            end if;
20          end if;
21        end if;
22      else
23        lightshow_count <= 0;
24        lightshow_step <= 0;
25      end if;
26    end process;
```

7.10 Blink on Wrong Password (wrong_pw_seg_blink_proc)

- When `cr_state = enter_pw` and `correct_pw = '0'` it increments `blinker_count` up to `blinker_hz` and toggles `blinker_state`.
- `blinker_state` is used in `refresh_prc` to blank the display (by setting `active_slot = -1`) when `blinker_state = '1'`.

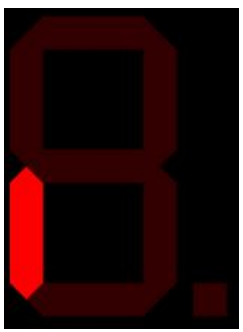
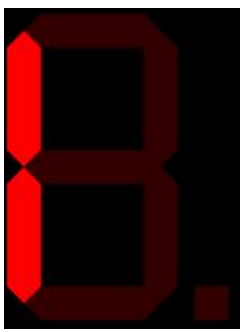
```vhdl
wrong_pw_seg_blink_proc: process (clk, cr_state, correct_pw)
begin
  if cr_state = enter_pw and correct_pw = '0' then
    if rising_edge(clk) then
      if blinker_count < blinker_hz - 1 then
        blinker_count <= blinker_count + 1;
      else
        blinker_state <= not blinker_state;
        blinker_count <= 0;
      end if;
    end if;
  else
    blinker_state <= '0';
    blinker_count <= 0;
  end if;
end process;
```

8. Animation Logic

Each character is animated in five steps, and their corresponding indices are arranged in a row. For example, the character ‘A’, which has a 7-segment mapping of `0001000` (active low, representing segments g to a), has an index of 19 in the `seg_display`. Therefore, the indices 15, 16, 17, 18, and 19 (from 15 to 19) are associated with the animation of character ‘A’. As the indices increase, the character appears more complete. Check next page’s example.

8. Animation Logic (cont.)

Character ‘A’s starting animation index = 15 → let index = 15

`step = 0` +	`step = 1` +	`step = 2` +	`step = 3` +	`step = 4` +
index = 15	index = 16	index = 17	index = 18	index = 19
				
`1101111`	`1001111`	`0001110`	`0001100`	`0001000`

Pictures Source: https://jasonacox.github.io/TM1637TinyDisplay/examples/7-segment-animator.html (Accesed 2025/12/13)

8. Animation Logic (cont.)

We know from Section 7.9 that **lightshow_step** ranges from 0 to 4. To implement each character's animation, we use the following code:

```vhdl
case cr_state is
  when lightshow =>
    lightshow_s := lightshow_step; -- integer range 0 to 4 for five different digit "frame"
          seg_slots_next(3) <= std_logic_vector(to_unsigned(lightshow_s + 25, 5));
          seg_slots_next(2) <= std_logic_vector(to_unsigned(lightshow_s + 20, 5));
          seg_slots_next(1) <= std_logic_vector(to_unsigned(lightshow_s + 15, 5));
          seg_slots_next(0) <= std_logic_vector(to_unsigned(lightshow_s + 10, 5));
end case;
```

This code snippet comes from Section 6.2. As you can see, we can add **lightshow_step** to the character's starting animation index. When **lightshow_step** is 4, the complete character is displayed.

Contact

For any questions or feedback, please feel free to reach out:

- **Name**: Mohammadhossein Keyvanfar
- **Email**: mohammadhossein.kv@gmail.com
- **Github**: MohammadHosseinKv
- **LinkedIn**: Mohammadhossein-keyvanfar
- **Telegram**: Mhmmd_Kv