HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA KỸ THUẬT ĐIỆN TỬ I

---o0o---

BÀI TẬP LỚN

THIẾT KẾ LOGIC SỐ

ĐỀ TÀI:

THIẾT KẾ BỘ ĐIỀU KHIỂN BÀN PHÍM MA TRẬN (KEYPAD CONTROLLER) DỰA TRÊN NỀN TẢNG FPGA CYCLONE IV

Giảng viên hướng dẫn: **TS. Vũ Anh Đào**

Lớp học phần: **D22DTMT01**

Nhóm sinh viên thực hiện: **Lý Trọng Nghĩa – B22DCDT222**

Nguyễn Đức Nam – B22DCDT206

Nguyễn Minh Đức – B22DCDT094

Dương Sơn Quang – B22DCDT238

Hà Nội – 11/2025

Mục lục

Danh sách hình vẽ

Danh sách bảng

CHƯƠNG I

GIỚI THIỆU

1.1 Đề bài

Bài 11: Bộ điều khiển bàn phím ma trận (Keypad Controller)

Yêu cầu:

- Quét bàn phím ma trận 4×4, phát hiện phím nhấn
- Kết hợp FSM (Finite State Machine) để nhập mật khẩu
- Hiển thị trên LED 7 đoạn 4 chữ số

1.2 Mô tả hệ thống

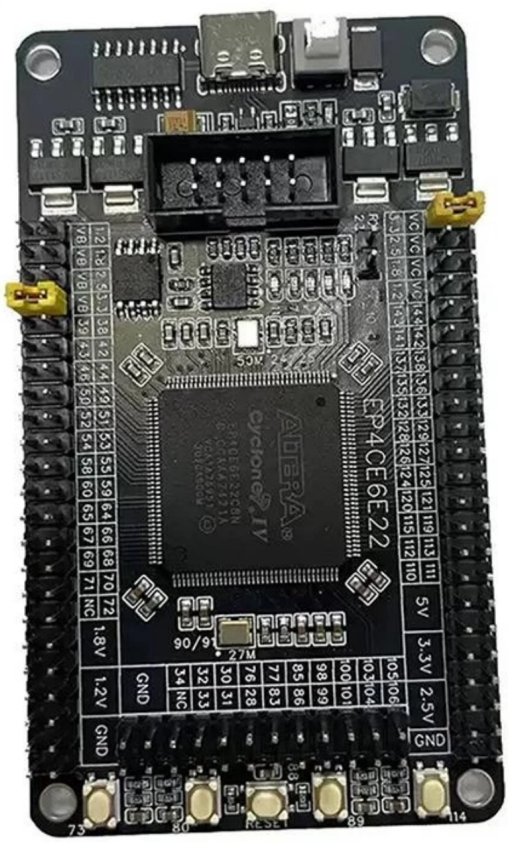
Hệ thống nhập mật khẩu sử dụng bàn phím ma trận 4×4 là một ứng dụng điển hình của thiết kế số trên FPGA. Hệ thống cho phép người dùng nhập mật khẩu 4 chữ số thông qua bàn phím, sau đó kiểm tra tính đúng đắn và hiển thị kết quả trên LED 7 đoạn.

Các chức năng chính:

1. Quét bàn phím ma trận 4×4 liên tục
2. Xử lý chống dội phím (debouncing)
3. Nhập mật khẩu 4 chữ số
4. So sánh với mật khẩu đã lưu (1234)
5. Hiển thị kết quả: ”PASS” hoặc ”FAIL”
6. Hỗ trợ phím chức năng: Clear (C) và Enter (E)

1.3 Các linh kiện sử dụng

1.3.1 FPGA Altera Cyclone IV (EP4CE6E22C8N)



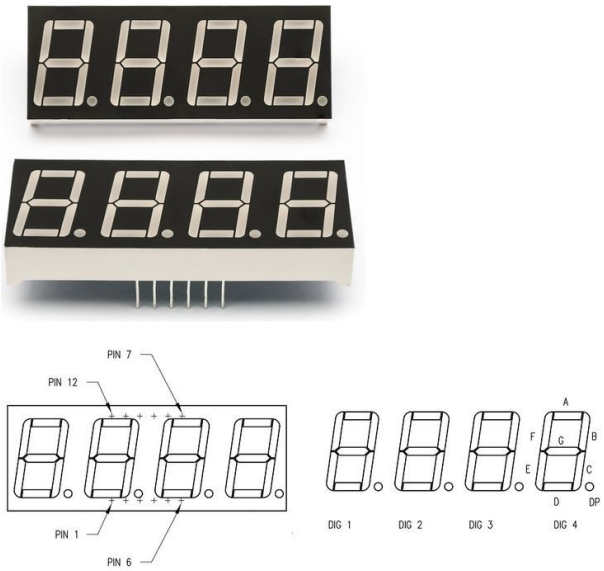
Hình 1.1: FPGA Altera Cyclone IV

EP4CE6E22C8N là chip FPGA thuộc dòng Cyclone IV E của Altera (Intel), được thiết kế cho các ứng dụng tiêu thụ điện năng thấp và chi phí hợp lý.

Thông số kỹ thuật:

- **Logic Elements (LE):** 6,272 LE
- **Bộ nhớ:** 270 Kb (30 khối M9K RAM)
- **Khối nhân:** 15 khối nhân 18×18 bit
- **I/O Pins:** 144 pins tối đa
- **Clock:** Hỗ trợ clock lên đến 400 MHz

1.3.2 LED 7 đoạn 4 chữ số - 5641AS (Common Cathode)



Hình 1.2: LED 7 đoạn 4 digit

LED 7 đoạn 5641AS là loại **Common Cathode**, nghĩa là:

- Tất cả các cathode của các đoạn LED được nối chung với GND
- Để bật một đoạn: đưa tín hiệu **HIGH (1)** vào anode
- Để tắt một đoạn: đưa tín hiệu **LOW (0)** vào anode

Cấu trúc:

- 8 chân segment: A, B, C, D, E, F, G, DP
- 4 chân common cathode: DIG1, DIG2, DIG3, DIG4
- Tần số quét: 100-200 Hz (để tránh nhấp nháy)

1.3.3 Bàn phím ma trận 4×4

Hình 1.3: Keypad nhựa 4x4

Bàn phím ma trận 4×4 gồm 16 nút bấm được sắp xếp thành 4 hàng và 4 cột.

Layout chuẩn:

```
      Col0  Col1  Col2  Col3
Row0  [1]   [2]   [3]   [A]
Row1  [4]   [5]   [6]   [B]
Row2  [7]   [8]   [9]   [C]
Row3  [*]   [0]   [#]   [D]
```

Chức năng phím:

- **0-9:** Nhập chữ số
- **C (phím C):** Xóa và bắt đầu lại
- **E (phím *):** Xác nhận mật khẩu (Enter)

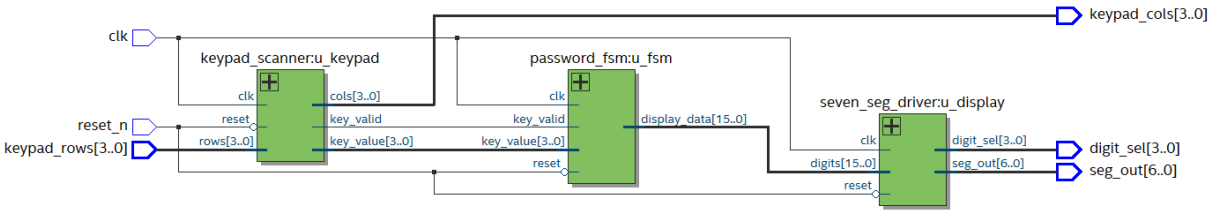
Nguyên lý quét ma trận:

1. **Kích hoạt cột:** Lần lượt đưa từng cột xuống mức LOW
2. **Đọc hàng:** Đọc trạng thái của 4 hàng (có pull-up resistor)
3. **Xác định phím:** Nếu hàng nào có mức LOW → phím tại (hàng, cột) đó được nhấn
4. **Chống dội:** Sử dụng bộ đếm 10ms để xác nhận phím nhấn hợp lệ

CHƯƠNG II

PHÂN TÍCH VÀ THIẾT KẾ HỆ THỐNG

2.1 Sơ đồ khối tổng thể



Hình 2.1: Sơ đồ khối tổng quát của hệ thống

Hệ thống được chia thành 5 module chính:

1. **keypad_password_system:** Module top-level, kết nối các module con
2. **keypad_scanner:** Quét bàn phím, chống dội
3. **password_fsm:** Máy trạng thái xử lý logic mật khẩu
4. **seven_seg_driver:** Điều khiển quét LED 7 đoạn
5. **hex_to_7seg:** Giải mã hex sang mã 7 đoạn

2.2 Module keypad_scanner

2.2.1 Chức năng

Module này thực hiện quét bàn phím ma trận 4×4, xử lý chống dội và phát hiện sự kiện nhấn phím với cơ chế phát hiện thông minh.

2.2.2 Interface

```
module keypad_scanner (
    input wire clk,              // 50MHz clock
    input wire reset,            // Active high reset
    input wire [3:0] rows,       // Row inputs (pulled-up)
    output reg [3:0] cols,       // Column outputs (active low)
    output reg [3:0] key_value,  // Detected key code (0-15)
    output reg key_valid         // One-cycle pulse when key pressed
);
```

Listing 2.1: Interface của keypad_scanner

2.2.3 Kiến trúc quét cột

Module sử dụng bộ đếm `scan_timer[15:0]` để tạo xung quét mỗi 1ms:

- Tần số clock: 50MHz
- Chu kỳ quét: 50,000 cycles = 1ms
- Mỗi cột được quét trong 1ms
- Tổng thời gian quét 4 cột: 4ms

```
reg [15:0] scan_timer;
reg [1:0] scan_index;
wire scan_tick;

assign scan_tick = (scan_timer == 16'd49999);

always @(posedge clk or posedge reset) begin
    if (reset) begin
        scan_timer <= 16'd0;
        scan_index <= 2'd0;
    end else if (scan_tick) begin
        scan_timer <= 16'd0;
        scan_index <= scan_index + 2'd1;
    end else begin
        scan_timer <= scan_timer + 16'd1;
    end
end
```

Listing 2.2: Logic quét cột

2.2.4 Logic sinh tín hiệu cột

```
always @(*) begin
    case (scan_index)
        2'b00: cols = 4'b1110; // Scan column 0
        2'b01: cols = 4'b1101; // Scan column 1
        2'b10: cols = 4'b1011; // Scan column 2
        2'b11: cols = 4'b0111; // Scan column 3
        default: cols = 4'b1111;
    endcase
end
```

Listing 2.3: Column output generation

2.2.5 Cơ chế chống dội nâng cao

Tầng 1: Đồng bộ hóa 2 stage (Synchronizer)

```
reg [3:0] row_sync1, row_sync2;

always @(posedge clk) begin
    row_sync1 <= rows;
    row_sync2 <= row_sync1;
end
```

Listing 2.4: Đồng bộ 2 tầng để tránh metastability

Tín hiệu rows (input) được chốt vào Flip-Flop tầng 1 (row_sync1) tại cạnh lên của xung clock. Nếu tín hiệu input thay đổi đúng thời điểm này, row_sync1 có thể rơi vào trạng thái Metastability (điện áp lửng lơ, không rõ 0/1).

Tín hiệu từ row_sync1 tiếp tục được chốt vào Flip-Flop tầng 2 (row_sync2) ở xung clock tiếp theo (20ns sau).

Khoảng thời gian 1 chu kỳ clock cho phép row_sync1 tự ổn định về mức logic xác định trước khi row_sync2 lấy mẫu, đảm bảo tín hiệu đi vào hệ thống luôn sạch và an toàn.

Tầng 2: Debouncing với bộ đếm 100us

```
reg [19:0] debounce_counter;
    reg [3:0] row_stable;
    reg [3:0] rows_debounced;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            debounce_counter <= 20'd0;
            row_stable <= 4'b1111;
            rows_debounced <= 4'b1111;
        end else begin
            if (row_sync2 != row_stable) begin
                debounce_counter <= 20'd0;
                row_stable <= row_sync2;
            end else begin
                if (debounce_counter < 20'd4999) begin
                    debounce_counter <= debounce_counter + 20'd1;
                end else begin
                    rows_debounced <= row_stable;
                end
            end
        end
    end
```

Listing 2.5: Debouncing logic

Debounce giúp loại bỏ nhiễu Rung phím cơ học (Mechanical Bouncing) xảy ra khi tiếp điểm kim loại va chạm, gây ra chuỗi xung ngẫu nhiên trước khi ổn định.

- Nguyên lý:
  - Sử dụng một bộ đếm thời gian (debounce_counter).
  - Khi tín hiệu từ bộ đồng bộ thay đổi trạng thái, bộ đếm bị reset về 0.
  - Bộ đếm chỉ tăng dần khi tín hiệu đầu vào giữ nguyên trạng thái ổn định.
  - Nếu tín hiệu ổn định liên tục trong khoảng thời gian quy định (100us), trạng thái đó mới được xác nhận là phím nhấn hợp lệ và cập nhật vào hệ thống.
- **Giải thích thời gian debouncing:**
  - Clock 50MHz → Chu kỳ = 20ns
  - Thời gian debounce = 5000 cycles × 20ns = 100us
  - Đủ để loại bỏ nhiễu cơ học của nút nhấn

2.2.6 Bảng tra cứu (LUT) giải mã phím

Module sử dụng bảng tra cứu để ánh xạ từ (hàng, cột) sang mã phím:

Row/Col	Col 0	Col 1	Col 2	Col 3
Row 0	1 (0x1)	2 (0x2)	3 (0x3)	A (0xA)
Row 1	4 (0x4)	5 (0x5)	6 (0x6)	B (0xB)
Row 2	7 (0x7)	8 (0x8)	9 (0x9)	C (0xC)
Row 3	E (0xE) *	0 (0x0)	F (0xF) #	D (0xD)

Bảng 2.1: *Bảng ánh xạ phím*

** Phím '*' được ánh xạ thành 0xE (Enter)*

```
reg [3:0] key_code_reg;

always @(*) begin
    key_code_reg = 4'hF; // Default: No key

    case (scan_index)
        2'b00: begin // Column 0
            if (rows_debounced == 4'b1110)
                key_code_reg = 4'h1; // '1'
            else if (rows_debounced == 4'b1101)
                key_code_reg = 4'h4; // '4'
            else if (rows_debounced == 4'b1011)
                key_code_reg = 4'h7; // '7'
            else if (rows_debounced == 4'b0111)
                key_code_reg = 4'hE; // '*' (ENTER)
        end

        2'b01: begin // Column 1
            if (rows_debounced == 4'b1110)
                key_code_reg = 4'h2;
            else if (rows_debounced == 4'b1101)
                key_code_reg = 4'h5;
            else if (rows_debounced == 4'b1011)
                key_code_reg = 4'h8;
            else if (rows_debounced == 4'b0111)
                key_code_reg = 4'h0;
        end

        2'b10: begin // Column 2
            if (rows_debounced == 4'b1110)
                key_code_reg = 4'h3;
            else if (rows_debounced == 4'b1101)
                key_code_reg = 4'h6;
            else if (rows_debounced == 4'b1011)
                key_code_reg = 4'h9;
            else if (rows_debounced == 4'b0111)
                key_code_reg = 4'hF;
        end

```

```
        2'b11: begin // Column 3
            if (rows_debounced == 4'b1110)
                key_code_reg = 4'hA;
            else if (rows_debounced == 4'b1101)
                key_code_reg = 4'hB;
            else if (rows_debounced == 4'b1011)
                key_code_reg = 4'hC;
            else if (rows_debounced == 4'b0111)
                key_code_reg = 4'hD;
        end
    endcase
end

assign key_code = key_code_reg;
```

Listing 2.6: Key detection logic

2.2.7 Máy trạng thái phát hiện nhấn/nhả phím

Module sử dụng FSM 3 trạng thái để phát hiện sự kiện nhấn phím:

```
localparam IDLE = 2'b00;
    localparam KEY_DETECTED = 2'b01;
    localparam WAIT_RELEASE = 2'b10;

    reg [1:0] state;
    reg [3:0] detected_key;
    reg [23:0] release_timer;

    always @(posedge clk or posedge reset) begin
        if (reset) begin
            state <= IDLE;
            key_valid <= 1'b0;
            key_value <= 4'hF;
            detected_key <= 4'hF;
            release_timer <= 24'd0;
        end else begin
            key_valid <= 1'b0;  // Default

            case (state)
                IDLE: begin
                    // Wait for valid key press
                    if (key_code != 4'hF) begin
                        detected_key <= key_code;
                        state <= KEY_DETECTED;
                        release_timer <= 24'd0;
                    end
                end

```

```
                KEY_DETECTED: begin
                    // Assert key_valid for 1 cycle only
                    key_valid <= 1'b1;
                    key_value <= detected_key;
                    state <= WAIT_RELEASE;
                end

                WAIT_RELEASE: begin
                    // Wait for key release
                    if (key_code == 4'hF) begin
                        if (release_timer < 24'd2500000) begin  // 50ms
                            release_timer <= release_timer + 24'd1;
                        end else begin
                            // Key released stable, return to IDLE
                            state <= IDLE;
                            detected_key <= 4'hF;
                            release_timer <= 24'd0;
                        end
                    end else begin
                        // Key still pressed, reset counter
                        release_timer <= 24'd0;
                    end
                end

                default: state <= IDLE;
            endcase
        end
    end
```

Listing 2.7: Key press detection state machine

Đoạn code trên mô tả máy trạng thái hữu hạn (FSM - Finite State Machine) bên trong module keypad_scanner. Mục đích chính của nó là xử lý sự kiện nhấn phím để tạo ra một tín hiệu đầu ra ”sạch” và duy nhất (key_valid và key_value) cho mỗi lần nhấn, bất kể ta giữ phím đó lâu hay nhanh.

Máy trạng thái này có 3 trạng thái hoạt động:

1. **IDLE (Trạng thái Chờ):**
   - Ý nghĩa: Hệ thống đang nghỉ ngơi, không có phím nào được nhấn, hoặc đang chờ một phím mới.
   - **Logic:**
     - Nó liên tục kiểm tra tín hiệu key_code. Tín hiệu này đến từ bộ giải mã cột/hàng.
     - Nếu key_code != 4’hF (nghĩa là có một phím đang được nhấn), nó sẽ:
       * Lưu mã phím đó vào biến detected_key.
       * Chuyển trạng thái sang KEY_DETECTED.

∗ Reset bộ đếm thời gian release_timer.

2. **KEY_DETECTED (Trạng thái Đã Phát Hiện Phím):**

- **Ý nghĩa:** Đây là trạng thái ”tức thời” để thông báo cho hệ thống biết vừa có một phím được nhấn.
- **Logic:**
  - key_valid <= 1’b1: Bật cờ báo hiệu ”có phím hợp lệ”. Cờ này chỉ bật trong đúng 1 chu kỳ xung nhịp (clock cycle), tạo ra một xung đơn (pulse). Đây là kỹ thuật quan trọng để tránh việc hệ thống xử lý lặp lại một phím nhiều lần (ví dụ nhấn số 1 mà ra 1111).
  - key_value <= detected_key: Đưa mã phím đã lưu ra ngõ ra để các module khác sử dụng.
  - state <= WAIT_RELEASE: Ngay lập tức chuyển sang trạng thái chờ nhả phím.

3. **WAIT_RELEASE (Trạng thái Chờ Nhả Phím):**

- **Ý nghĩa:** Hệ thống buộc người dùng phải nhả phím ra trước khi chấp nhận một lần nhấn mới. Điều này ngăn chặn hiện tượng ”lặp phím” (key repeat) nếu người dùng giữ phím quá lâu.
- **Logic:**
  - Nó kiểm tra key_code == 4’hF (nghĩa là phím đã được nhả ra, không còn hàng nào được nối với cột).
  - Cơ chế Debounce khi nhả (Release Debounce): Khi nhả tay, tiếp điểm cơ khí có thể bị rung, làm tín hiệu chập chờn. Code sử dụng release_timer để đảm bảo phím đã thực sự được nhả ổn định. Nếu phím được nhả (key_code == 4’hF), bộ đếm release_timer bắt đầu chạy.
  - Nó phải đếm đến 2,500,000 chu kỳ (tương đương 50ms với xung nhịp 50MHz).
  - Nếu trong quá trình đếm mà tín hiệu lại nhảy về ”có nhấn” (do rung phím), bộ đếm sẽ bị reset về 0.
  - Chỉ khi phím được nhả ổn định liên tục trong 50ms, hệ thống mới quay về trạng thái IDLE để sẵn sàng nhận phím tiếp theo.

4. **Ưu điểm của thiết kế:**

- Tạo xung `key_valid` chỉ một chu kỳ clock
- Chống nhấn liên tục (key repeat)
- Yêu cầu phím được thả ổn định 50ms trước khi chấp nhận nhấn mới
- Đảm bảo mỗi lần nhấn chỉ tạo một sự kiện

2.3 Module password_fsm

2.3.1 Chức năng

Module password_fsm.v là ”bộ não” của hệ thống. Nó thực hiện các nhiệm vụ:

- Nhận dữ liệu phím nhấn từ bàn phím.
- Quản lý quá trình nhập mật khẩu (tối đa 4 chữ số).
- So sánh mật khẩu nhập vào với mật khẩu được cài đặt sẵn (mặc định là 1234).
- Quyết định trạng thái hệ thống: Chờ, Đang nhập, Mở khóa thành công (PASS), hoặc Sai mật khẩu (FAIL).
- Điều khiển dữ liệu hiển thị trên màn hình LED 7 đoạn.

2.3.2 Interface

```
module password_fsm (
    input wire clk,
    input wire reset,
    input wire [3:0] key_value,
    input wire key_valid,
    output reg [15:0] display_data
);
```

Listing 2.8: Interface của password_fsm

2.3.3 Định nghĩa các hằng số

```
// ===== Key Definitions =====
localparam KEY_C = 4'hC;  // Clear key
localparam KEY_E = 4'hE;  // Enter key

// ===== Display Character Codes =====
localparam CHAR_A    = 4'hA;
localparam CHAR_P    = 4'hB;
localparam CHAR_S    = 4'hC;
localparam CHAR_F    = 4'hD;
localparam CHAR_L    = 4'hE;
localparam CHAR_I    = 4'h1;
localparam CHAR_DASH = 4'hF;

localparam DISP_FAIL = {CHAR_F, CHAR_A, CHAR_I, CHAR_L};  // "FAIL"
localparam DISP_PASS = {CHAR_P, CHAR_A, CHAR_S, CHAR_S};  // "PASS"
localparam DISP_IDLE = {CHAR_DASH, CHAR_DASH, CHAR_DASH, CHAR_DASH}; //
    "----"
```

```
17
18 // ===== Password Configuration =====
19 localparam [15:0] PASSWORD = 16'h1234;  // Default: 1234
```

Listing 2.9: Key definitions và display character codes

2.3.4 Các trạng thái FSM

```
1 localparam S_IDLE     = 2'b00;
2 localparam S_ENTERING = 2'b01;
3 localparam S_SUCCESS  = 2'b10;
4 localparam S_FAIL     = 2'b11;
5
6 reg [1:0] state, next_state;
```

Listing 2.10: Định nghĩa các trạng thái

Hệ thống sử dụng mô hình FSM kiểu Moore với 4 trạng thái chính, được mã hóa bằng 2 bit:

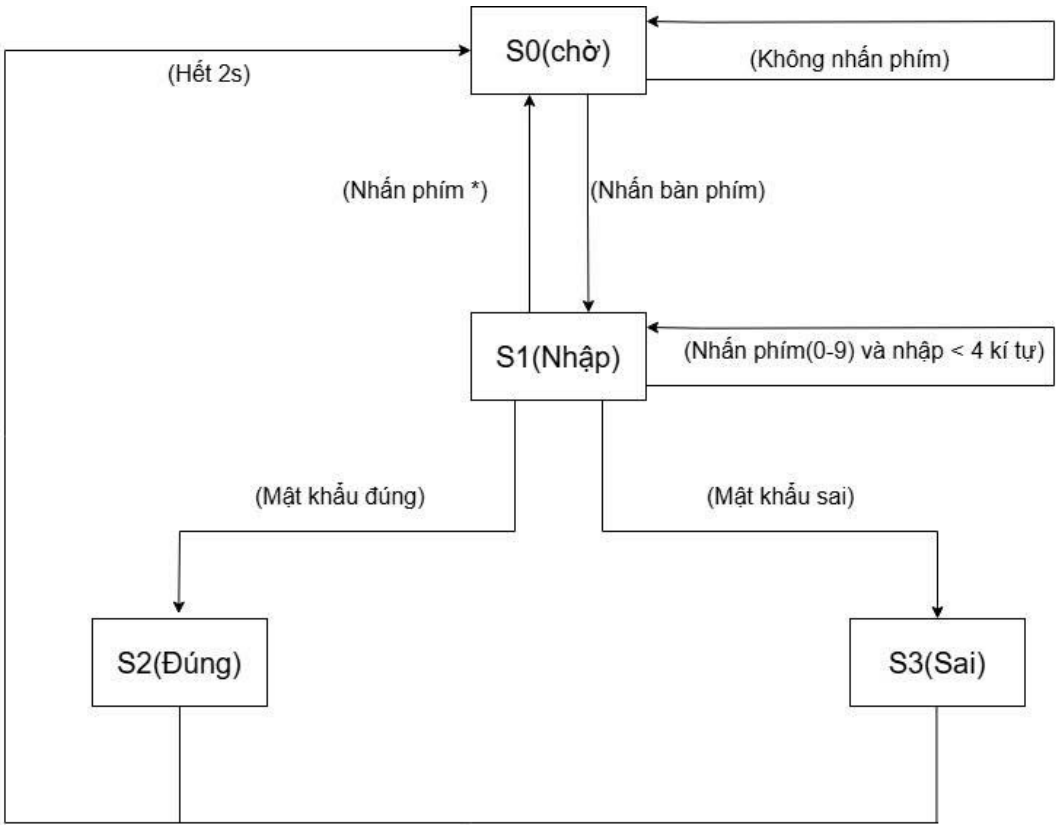
1. **S_IDLE (2'b00) - Trạng thái Chờ:**
   - Đây là trạng thái mặc định khi khởi động hoặc sau khi reset.
   - Màn hình hiển thị —- (DISP_IDLE).
   - Hệ thống chờ người dùng nhấn một phím số (0-9) để bắt đầu quá trình nhập.

2. **S_ENTERING (2'b01) - Trạng thái Đang Nhập:**
   - Người dùng đang nhập các chữ số của mật khẩu.
   - Hệ thống lưu trữ các số đã nhập vào thanh ghi entered_digits (dịch trái và thêm số mới vào).
   - Màn hình hiển thị các số vừa nhập.
   - Cho phép nhấn phím KEY_C (Clear) để hủy bỏ và quay về S_IDLE.
   - Cho phép nhấn KEY_E (Enter) để xác nhận mật khẩu ngay lập tức (nếu nhập ít hơn 4 số).
   - Tự động kiểm tra mật khẩu ngay khi nhập đủ 4 chữ số.

3. **S_SUCCESS (2'b10) - Trạng thái Thành Công:**
   - Được kích hoạt khi mật khẩu nhập vào trùng khớp với PASSWORD (1234).
   - Màn hình hiển thị PASS (DISP_PASS).
   - Duy trì trạng thái này trong 2 giây nhờ bộ đếm thời gian (timer).

4. **S_FAIL (2'b11) - Trạng thái Thất Bại:**
   - Được kích hoạt khi mật khẩu nhập vào KHÔNG trùng khớp.
   - Màn hình hiển thị FAIL (DISP_FAIL).
   - Duy trì trạng thái này trong 2 giây rồi quay về S_IDLE.

2.3.5 Đồ hình trạng thái



Hình 2.2: Sơ đồ trạng thái của FSM

2.3.6 Các thanh ghi dữ liệu

```
reg [15:0] entered_digits;
reg [1:0] digit_count;

// Timer for PASS/FAIL Display (2 seconds)
// 50MHz * 2s = 100,000,000 cycles
reg [26:0] timer;
wire timer_done;
assign timer_done = (timer == 27'd99999999);
```

Listing 2.11: Data registers

2.3.7 Logic chuyển trạng thái (Combinational)

```
always @(*) begin
    next_state = state;

    case (state)
        S_IDLE: begin
            if (key_valid && (key_value <= 4'd9))
                next_state = S_ENTERING;
        end

        S_ENTERING: begin
            if (key_valid) begin
                if (key_value == KEY_C)
                    next_state = S_IDLE;
                else if (key_value == KEY_E)
                    next_state = (entered_digits == PASSWORD)
                                ? S_SUCCESS : S_FAIL;
                else if (key_value <= 4'd9 && digit_count == 2'd3)
                    next_state = (({entered_digits[11:0], key_value})
                                == PASSWORD) ? S_SUCCESS : S_FAIL;
            end
        end

        S_SUCCESS: begin
            if (timer_done)
                next_state = S_IDLE;
        end

        S_FAIL: begin
            if (timer_done)
                next_state = S_IDLE;
        end
    endcase
end
```

Listing 2.12: State transition logic

Phân tích logic chuyển trạng thái:

- Từ `S_IDLE`: Chuyển sang `S_ENTERING` khi nhấn số (0-9)
- Từ `S_ENTERING`:
  - Nhấn C → Về `S_IDLE`
  - Nhấn E → Kiểm tra mật khẩu → `S_SUCCESS` hoặc `S_FAIL`
  - Nhập đủ 4 số → Tự động kiểm tra → `S_SUCCESS` hoặc `S_FAIL`
- Từ `S_SUCCESS`/`S_FAIL`: Sau 2 giây → Về `S_IDLE`

2.3.8 Logic cập nhật dữ liệu (Sequential)

```
always @(posedge clk or posedge reset) begin
    if (reset) begin
        state <= S_IDLE;
        entered_digits <= 16'd0;
        digit_count <= 2'd0;
        display_data <= DISP_IDLE;
        timer <= 27'd0;
    end else begin
        state <= next_state;

        // Timer management
        if (next_state != state)
            timer <= 27'd0;
        else if (!timer_done)
            timer <= timer + 27'd1;

        // State-dependent data updates
        case (next_state)
            S_IDLE: begin
                display_data <= DISP_IDLE;
                entered_digits <= 16'd0;
                digit_count <= 2'd0;
            end

            S_ENTERING: begin
                if (state == S_IDLE && key_valid
                    && (key_value <= 4'd9)) begin
                    // First digit entered
                    entered_digits <= {12'hFFF, key_value};
                    display_data <= {12'hFFF, key_value};
                    digit_count <= 2'd1;
                end else if (state == S_ENTERING && key_valid
                    && (key_value <= 4'd9) && (digit_count < 3)) begin
                    // Subsequent digits
                    entered_digits <= {entered_digits[11:0], key_value};
                    display_data <= {entered_digits[11:0], key_value};
                    digit_count <= digit_count + 2'd1;
                end
            end

            S_SUCCESS: begin
                display_data <= DISP_PASS;
                entered_digits <= 16'd0;
                digit_count <= 2'd0;
            end

            S_FAIL: begin
```

```
                display_data <= DISP_FAIL;
                entered_digits <= 16'd0;
                digit_count <= 2'd0;
            end
        endcase
    end
end
```

Listing 2.13: State update and data processing

Cơ chế nhập mật khẩu:

- Chữ số đầu tiên: `entered_digits = 16'hFFFX` (X là số nhập)
- Chữ số thứ hai: `entered_digits = 16'hFFXY`
- Chữ số thứ ba: `entered_digits = 16'hFXYZ`
- Chữ số thứ tư: `entered_digits = 16'hXYZW` → Tự động kiểm tra

2.4 Module seven_seg_driver

2.4.1 Chức năng

Module này thực hiện kỹ thuật quét LED (multiplexing) để hiển thị 4 ký tự lên 4 LED 7 đoạn với kiến trúc pipeline 4 tầng.

2.4.2 Interface

```
module seven_seg_driver (
    input wire clk,              // 50MHz clock
    input wire reset,            // Active high reset
    input wire [15:0] digits,    // 4 hex digits [D3, D2, D1, D0]
    output reg [6:0] seg_out,    // Segment outputs {g,f,e,d,c,b,a}
    output reg [3:0] digit_sel   // Digit select (Active-LOW)
);
```

Listing 2.14: Interface của seven_seg_driver

2.4.3 Kiến trúc Pipeline 4 tầng

Trong module điều khiển hiển thị LED 7 đoạn (`seven_seg_driver`), ta áp dụng kỹ thuật Pipelining (đường ống) 4 tầng. Mục đích của thiết kế này là loại bỏ hiện tượng "trôi dữ liệu" (data bleeding) và đảm bảo tính đồng bộ tuyệt đối giữa tín hiệu chọn đèn (`digit_sel`) và dữ liệu hiển thị (`seg_out`) tại tần số hoạt động cao của FPGA.

Nếu không sử dụng Pipeline, tín hiệu điều khiển đèn (`digit_sel`) thường đến chân xuất nhanh hơn tín hiệu dữ liệu (`seg_out`) do dữ liệu phải đi qua khối logic giải mã phức tạp. Sự chênh lệch này (khoảng vài nano-giây) gây ra hiện tượng "bóng ma" (ghosting), nơi đèn tiếp theo bật lên nhưng vẫn hiển thị dữ liệu của đèn trước đó trong một khoảnh khắc ngắn.

Kỹ thuật Pipeline buộc tín hiệu nhanh phải "chờ" tín hiệu chậm thông qua các thanh ghi trung gian, đảm bảo chúng đến đích cùng một lúc, mang lại chất lượng hiển thị sắc nét và ổn định.

Stage 1: Refresh Counter

- **Chức năng:** Tạo ra tín hiệu định thời gian để quét qua các đèn LED.
- **Hoạt động:** Sử dụng một bộ đếm 18-bit. Hai bit cao nhất (`refresh_counter[17:16]`) được trích xuất để làm tín hiệu chọn đèn (`current_digit`).
- **Kết quả:** Giá trị chọn đèn được lưu vào thanh ghi `current_digit_s1`.

```
reg [17:0] refresh_counter;
reg [1:0] current_digit_s1;

always @(posedge clk or posedge reset) begin
    if (reset) begin
        refresh_counter <= 18'd0;
        current_digit_s1 <= 2'b0;
    end else begin
        refresh_counter <= refresh_counter + 18'd1;
        current_digit_s1 <= refresh_counter[17:16];
    end
end
```

Listing 2.15: Stage 1 - Refresh counter

Tính toán tần số quét:

- Counter 18-bit: 2^{18} = 262,144 cycles
- Tần số quét tổng: 50MHz / 262,144 ≈ 190.7 Hz
- Tần số quét mỗi LED: 190.7Hz / 4 ≈ 47.7 Hz
- Hoàn toàn đủ để tránh nhấp nháy (>30Hz)

Stage 2: Hex MUX

- **Chức năng:** Chọn dữ liệu 4-bit (số Hex) tương ứng với đèn đang được kích hoạt.
- **Hoạt động:** Dựa vào tín hiệu `current_digit_s1` từ Stage 1, bộ MUX sẽ chọn 1 trong 4 luồng dữ liệu đầu vào (`digits[15:0]`).
- **Kết quả:** Dữ liệu Hex được chọn lưu vào thanh ghi `current_hex_s2`. Đồng thời, tín hiệu chọn đèn được truyền tiếp sang thanh ghi đệm `current_digit_s2`.

```
reg [3:0] current_hex_s2;

always @(posedge clk or posedge reset) begin
    if (reset) begin
        current_hex_s2 <= 4'hF;
    end else begin
        case (current_digit_s1)
            2'b00: current_hex_s2 <= digits[15:12]; // D3
            2'b01: current_hex_s2 <= digits[11:8];  // D2
            2'b10: current_hex_s2 <= digits[7:4];   // D1
            2'b11: current_hex_s2 <= digits[3:0];   // D0
        endcase
    end
end
```

Listing 2.16: Stage 2 - Hex multiplexer

Stage 3: Hex-to-Segment Decoder

Đây là tầng quan trọng nhất để xử lý độ trễ lan truyền (propagation delay).

- **Nhánh Dữ liệu:** Số Hex từ Stage 2 đi qua bộ giải mã tổ hợp `hex_to_7seg` để chuyển thành mã 7 đoạn. Quá trình này tốn thời gian xử lý vật lý. Kết quả sau đó được chốt vào thanh ghi `decoded_segments_s3`.
- **Nhánh Điều khiển:** Tín hiệu chọn đèn (`current_digit_s2`) không cần xử lý, nhưng được đưa qua một thanh ghi đệm (`current_digit_s3`) để tạo độ trễ tương ứng.
- **Mục đích:** Đảm bảo cả dữ liệu hiển thị và tín hiệu chọn đèn đều sẵn sàng tại cùng một thời điểm ở cuối chu kỳ xung nhịp.

```
reg [6:0] decoded_segments_s3;

wire [6:0] decoded_segments_comb;
hex_to_7seg decoder_inst (
    .hex_in(current_hex_s2),
    .seg_out(decoded_segments_comb)
);

always @(posedge clk or posedge reset) begin
    if (reset) begin
        decoded_segments_s3 <= 7'b0000000;
    end else begin
        decoded_segments_s3 <= decoded_segments_comb;
    end
end
```

Listing 2.17: Stage 3 - Decoder với register

Pipeline Delay cho Digit Select

```
reg [1:0] current_digit_s2;
reg [1:0] current_digit_s3;

always @(posedge clk or posedge reset) begin
    if (reset) begin
        current_digit_s2 <= 2'b0;
        current_digit_s3 <= 2'b0;
    end else begin
        current_digit_s2 <= current_digit_s1; // S1 -> S2
        current_digit_s3 <= current_digit_s2; // S2 -> S3
    end
end
```

Listing 2.18: Pipeline synchronization

Stage 4: Output Registers

- **Chức năng:** Đưa tín hiệu sạch ra chân vật lý của FPGA.
- **Hoạt động:** Tại cạnh lên của xung nhịp, dữ liệu từ thanh ghi Stage 3 được đẩy ra `seg_out` và `digit_sel`.
- **Kết quả:** Tín hiệu ngõ ra hoàn toàn đồng bộ, loại bỏ mọi nhiễu (glitch) do sự chênh lệch thời gian xử lý giữa các luồng logic.

```
always @(posedge clk or posedge reset) begin
    if (reset) begin
        seg_out <= 7'b0000000;
        digit_sel <= 4'b1111;  // All OFF (Active-LOW)
    end else begin
        seg_out <= decoded_segments_s3;

        case (current_digit_s3)
            2'b00: digit_sel <= 4'b0111;  // Enable digit 3 (left)
            2'b01: digit_sel <= 4'b1011;  // Enable digit 2
            2'b10: digit_sel <= 4'b1101;  // Enable digit 1
            2'b11: digit_sel <= 4'b1110;  // Enable digit (right)
        endcase
    end
end
```

Listing 2.19: Stage 4 - Output registers (Active-LOW CC)

2.4.4 Ưu điểm của kiến trúc Pipeline

1. **Timing tốt hơn:**
   - Mỗi stage đơn giản hơn
   - Giảm critical path

- Tăng tần số clock tối đa

2. **Giảm glitch:**
   - Segment và digit select được đồng bộ
   - Không có hiện tượng ”ghost segment”
   - Hiển thị ổn định hơn
3. **Dễ debug:**
   - Mỗi stage có register riêng
   - Có thể quan sát tín hiệu từng tầng
   - Dễ xác định lỗi

2.5 Module hex_to_7seg

2.5.1 Chức năng

Module này thực hiện giải mã từ mã hex 4-bit sang mã 7 đoạn 7-bit cho LED Common Cathode.

2.5.2 Interface

```
module hex_to_7seg (
    input [3:0] hex_in,
    output reg [6:0] seg_out  // {g,f,e,d,c,b,a}
);
```

Listing 2.20: Interface của hex_to_7seg

2.5.3 Bảng mã

Hex	Ký tự	Mã 7-bit	Segments ON
0x0	0	7'b0111111	a,b,c,d,e,f
0x1	1	7'b0000110	b,c
0x2	2	7'b1011011	a,b,g,e,d
0x3	3	7'b1001111	a,b,g,c,d
0x4	4	7'b1100110	f,g,b,c
0x5	5	7'b1101101	a,f,g,c,d
0x6	6	7'b1111101	a,f,g,e,d,c
0x7	7	7'b0000111	a,b,c
0x8	8	7'b1111111	all segments
0x9	9	7'b1101111	a,b,c,d,f,g
0xA	A	7'b1110111	a,b,c,e,f,g
0xB	P	7'b1110011	a,b,e,f,g
0xC	S	7'b1101101	a,f,g,c,d
0xD	F	7'b1110001	a,e,f,g
0xE	L	7'b0111000	d,e,f
0xF	-	7'b1000000	g only (dash)

Bảng 2.2: *Bảng mã hex sang 7 đoạn (Common Cathode)*

2.5.4 Code implementation

```
module hex_to_7seg (
    input [3:0] hex_in,
    output reg [6:0] seg_out  // {g,f,e,d,c,b,a}
);

    // Common Cathode encoding (1 = segment ON, 0 = segment OFF)
    always @(*) begin
        case (hex_in)
            // Numeric digits 0-9
            4'h0: seg_out = 7'b0111111;  // 0: a,b,c,d,e,f
            4'h1: seg_out = 7'b0000110;  // 1: b,c
            4'h2: seg_out = 7'b1011011;  // 2: a,b,g,e,d
            4'h3: seg_out = 7'b1001111;  // 3: a,b,g,c,d
            4'h4: seg_out = 7'b1100110;  // 4: f,g,b,c
            4'h5: seg_out = 7'b1101101;  // 5: a,f,g,c,d
            4'h6: seg_out = 7'b1111101;  // 6: a,f,g,e,d,c
            4'h7: seg_out = 7'b0000111;  // 7: a,b,c
            4'h8: seg_out = 7'b1111111;  // 8: all segments
            4'h9: seg_out = 7'b1101111;  // 9: a,b,c,d,f,g

```

```
            // Special characters for PASS/FAIL display
            4'hA: seg_out = 7'b1110111;  // A: a,b,c,e,f,g
            4'hB: seg_out = 7'b1110011;  // P: a,b,e,f,g
            4'hC: seg_out = 7'b1101101;  // S: a,f,g,c,d (same as 5)
            4'hD: seg_out = 7'b1110001;  // F: a,e,f,g
            4'hE: seg_out = 7'b0111000;  // L: d,e,f
            4'hF: seg_out = 7'b1000000;  // - (dash): g only

            default: seg_out = 7'b0000000;  // All segments OFF
        endcase
    end

endmodule
```

Listing 2.21: Module hex_to_7seg - Full implementation

Lưu ý về ký tự đặc biệt:

- **0xB → "P":** Dùng cho chữ "PASS"
- **0xC → "S":** Dùng cho chữ "PASS"
- **0xD → "F":** Dùng cho chữ "FAIL"
- **0xE → "L":** Dùng cho chữ "FAIL"
- **0x1 → "I":** Dùng cho chữ "FAIL" (giống số 1)

2.6 Module top-level: keypad_password_system

2.6.1 Chức năng

Module top-level kết nối tất cả các module con và quản lý tín hiệu I/O.

2.6.2 Interface

```
module keypad_password_system (
    // Clock and Reset
    input wire clk,           // 50MHz external clock
    input wire reset_n,       // Active-low reset button

    // 4x4 Keypad Interface
    input wire [3:0] keypad_rows,    // Row inputs (pulled-up)
    output wire [3:0] keypad_cols,   // Column outputs (active low)

    // 7-Segment Display (5641AS Common Cathode)
    output wire [6:0] seg_out,       // Segments {g,f,e,d,c,b,a}
    output wire [3:0] digit_sel      // Digit select (Active-LOW)
);
```

Listing 2.22: Top-level interface

2.6.3 Kết nối các module

```
// Internal reset (active high for internal logic)
wire reset;
assign reset = ~reset_n;

// Interconnect wires
wire [3:0] w_key_value;
wire w_key_valid;
wire [15:0] w_display_data;

// ===== Keypad Scanner =====
keypad_scanner u_keypad (
    .clk(clk),
    .reset(reset),
    .rows(keypad_rows),
    .cols(keypad_cols),
    .key_value(w_key_value),
    .key_valid(w_key_valid)
);

// ===== Password FSM =====
password_fsm u_fsm (
    .clk(clk),
    .reset(reset),
    .key_value(w_key_value),
    .key_valid(w_key_valid),
    .display_data(w_display_data)
);

// ===== Seven Segment Display Driver =====
seven_seg_driver u_display (
    .clk(clk),
    .reset(reset),
    .digits(w_display_data),
    .seg_out(seg_out),
    .digit_sel(digit_sel)
);
```

Listing 2.23: Module interconnections

Luồng dữ liệu:

1. `keypad_scanner` đọc phím → xuất `w_key_value` và `w_key_valid`
2. `password_fsm` nhận key → xử lý logic → xuất `w_display_data`

3. `seven_seg_driver` nhận display data → quét LED 7 đoạn

2.6.4 Bảng gán chân (Pin Assignment)

Bảng dưới đây mô tả chi tiết việc gán chân giữa các tín hiệu của hệ thống với các chân vật lý trên FPGA Cyclone IV EP4CE6E22C8N.

Tín hiệu	Hướng	Chân	I/O Bank	Chuẩn I/O	Ghi chú
Clock và Reset					
clk	Input	PIN_25	2	3.3V LVTTL	Clock 50MHz
reset_n	Input	PIN_73	5	3.3V LVTTL	Pull-up enabled
Keypad Rows (Input)					
keypad_rows[3]	Input	PIN_135	8	3.3V LVTTL	Pull-up enabled
keypad_rows[2]	Input	PIN_136	8	3.3V LVTTL	Pull-up enabled
keypad_rows[1]	Input	PIN_137	8	3.3V LVTTL	Pull-up enabled
keypad_rows[0]	Input	PIN_138	8	3.3V LVTTL	Pull-up enabled
Keypad Columns (Output)					
keypad_cols[3]	Output	PIN_141	8	3.3V LVTTL	Active-low
keypad_cols[2]	Output	PIN_142	8	3.3V LVTTL	Active-low
keypad_cols[1]	Output	PIN_143	8	3.3V LVTTL	Active-low
keypad_cols[0]	Output	PIN_144	8	3.3V LVTTL	Active-low
7-Segment Display - Segments					
seg_out[6]	Output	PIN_55	4	3.3V LVTTL	Segment g
seg_out[5]	Output	PIN_58	4	3.3V LVTTL	Segment f
seg_out[4]	Output	PIN_59	4	3.3V LVTTL	Segment e
seg_out[3]	Output	PIN_60	4	3.3V LVTTL	Segment d
seg_out[2]	Output	PIN_64	4	3.3V LVTTL	Segment c
seg_out[1]	Output	PIN_65	4	3.3V LVTTL	Segment b
seg_out[0]	Output	PIN_66	4	3.3V LVTTL	Segment a
7-Segment Display - Digit Select					
digit_sel[3]	Output	PIN_38	3	3.3V LVTTL	Digit 3 (trái)
digit_sel[2]	Output	PIN_39	3	3.3V LVTTL	Digit 2
digit_sel[1]	Output	PIN_42	3	3.3V LVTTL	Digit 1
digit_sel[0]	Output	PIN_43	3	3.3V LVTTL	Digit 0 (phải)

Bảng 2.3: *Bảng gán chân FPGA*

CHƯƠNG III

MÔ PHỎNG VÀ TRIỂN KHAI THỰC TẾ

3.1 Môi trường mô phỏng

Phần mềm: ModelSim (Intel/Altera Edition)

Testbench: File `tb_keypad_password.v` được thiết kế để:

- Giả lập tín hiệu clock 50MHz
- Giả lập nhấn phím thông qua task `press_key`
- Theo dõi và hiển thị kết quả ra console
- Kiểm tra tất cả các chức năng của hệ thống

3.1.1 Cơ chế giả lập nhấn phím (Task press_key)

Để mô phỏng hành vi nhấn phím trên bàn phím ma trận, testbench sử dụng một `task` trong Verilog có tên `press_key`. Task này tái tạo chính xác trình tự tín hiệu điện xảy ra khi người dùng nhấn một phím vật lý.

```
task press_key(input [1:0] row, input [1:0] col);
    begin
        @(negedge keypad_cols[col]);  // Cho cot duoc quet
        #1000;                        // Tre nho (1us)
        keypad_rows = ~(4'b0001 << row); // Kich hoat hang
        #50000000;                    // Giu phim 50ms
        keypad_rows = 4'b1111;        // Tha phim
        #50000000;                    // Cho truoc phim tiep
    end
endtask
```

Listing 3.1: Task giả lập nhấn phím

Giải thích chi tiết từng bước:

1. **Đồng bộ với chu kỳ quét cột:**

   ```
   @(negedge keypad_cols[col]);
   ```

   Task chờ cho đến khi cột tương ứng (`col`) được module `keypad_scanner` kích hoạt (chuyển từ HIGH xuống LOW). Điều này đảm bảo phím được ”nhấn” đúng thời điểm cột đó đang được quét.

2. **Độ trễ ổn định (1µs):**

```
#1000;
```

Một khoảng trễ nhỏ (1000ns = 1µs) để đảm bảo tín hiệu cột đã ổn định trước khi kích hoạt hàng.

3. **Kích hoạt tín hiệu hàng (Active-LOW):**

```
keypad_rows = ~(4'b0001 << row);
```

Tạo tín hiệu hàng tương ứng với phím cần nhấn. Ví dụ:

- `row = 0`: `keypad_rows = ~(0001) = 1110` (Hàng 0 LOW)
- `row = 1`: `keypad_rows = ~(0010) = 1101` (Hàng 1 LOW)
- `row = 2`: `keypad_rows = ~(0100) = 1011` (Hàng 2 LOW)
- `row = 3`: `keypad_rows = ~(1000) = 0111` (Hàng 3 LOW)

4. **Giữ phím trong 50ms:**

```
#50000000;
```

Thời gian 50ms (50,000,000ns) mô phỏng khoảng thời gian người dùng giữ phím. Giá trị này đủ lớn để vượt qua ngưỡng debounce (100µs) của module `keypad_scanner`.

5. **Thả phím:**

```
keypad_rows = 4'b1111;
```

Đưa tất cả các hàng về mức HIGH, mô phỏng trạng thái không có phím nào được nhấn (các hàng được kéo lên bởi điện trở pull-up).

6. **Chờ trước khi nhấn phím tiếp theo (50ms):**

```
#50000000;
```

Khoảng nghỉ giữa các lần nhấn phím để đảm bảo module `keypad_scanner` hoàn tất chu trình phát hiện nhả phím và sẵn sàng cho lần nhấn tiếp theo.

Ví dụ sử dụng:

```
// Nhan phim '1' (Hang 0, Cot 0)
press_key(0, 0);

// Nhan phim '5' (Hang 1, Cot 1)
press_key(1, 1);

// Nhan phim 'C' - Clear (Hang 2, Cot 3)
press_key(2, 3);

// Nhan phim '*' - Enter (Hang 3, Cot 0)
press_key(3, 0);
```

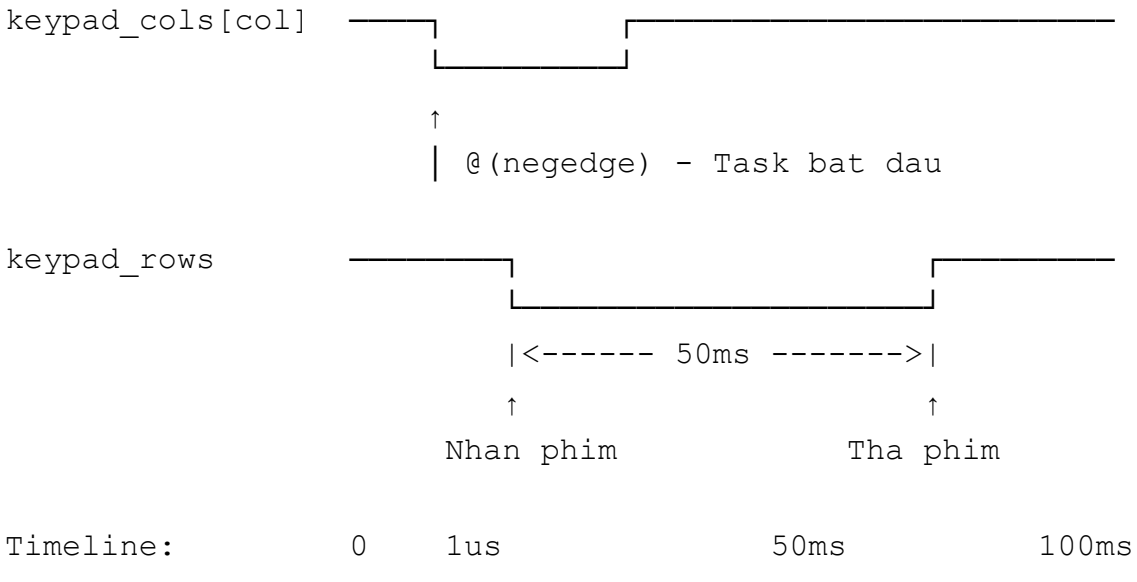
Listing 3.2: Ví dụ gọi task press_key

Bảng ánh xạ tham số với phím:

row \col	0	1	2	3
0	1	2	3	A
1	4	5	6	B
2	7	8	9	C (Clear)
3	* (Enter)	0	#	D

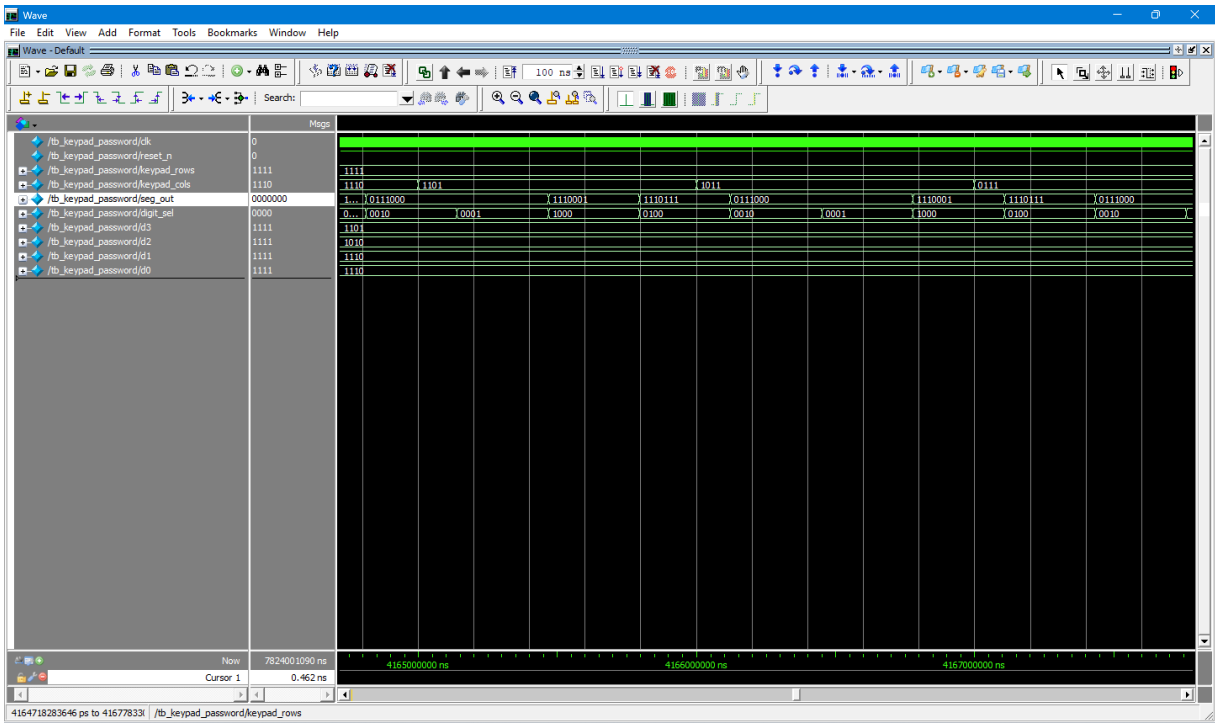
Bảng 3.1: *Ánh xạ tham số (row, col) với phím trên bàn phím 4x4*

Biểu đồ thời gian của một lần nhấn phím:



Hình 3.1: Biểu đồ thời gian mô phỏng nhấn phím

3.2 Dạng sóng mô phỏng



Hình 3.2: Dạng sóng đầu ra của hệ thống

3.2.1 Phân tích trạng thái hệ thống

Trước khi nhấn phím:

- `reset_n = 1`: Hệ thống hoạt động bình thường
- FSM ở trạng thái `S_IDLE`
- Display data: `d3=d2=d1=d0=4'hF` (hiển thị ”—-”)
- `digit_sel` quét liên tục: 0001 → 0010 → 0100 → 1000
- `seg_out = 7'b1000000` (mã cho dấu gạch ngang)

Sự kiện nhấn phím:

- `keypad_cols` quét: 1110 → 1101 → 1011 → 0111
- Tại thời điểm ≈416,742,000 ns: `keypad_cols = 1101` (Cột 1)
- Testbench kéo `keypad_rows` từ 1111 xuống 1101 (Hàng 1)
- Phím được phát hiện: (Hàng 1, Cột 1) = Phím ’5’ (4’h5)

3.3 Các kịch bản kiểm thử

3.3.1 Kịch bản 1: Nhập đúng mật khẩu ”1234”

Mô tả: Giả lập nhấn lần lượt 4 phím ”1”, ”2”, ”3”, ”4”

Kết quả mong đợi:

1. Hiển thị lần lượt: ”—1” → ”–12” → ”-123” → ”1234”
2. FSM chuyển sang S_SUCCESS
3. Hiển thị ”PASS” trong 2 giây
4. Tự động quay về ”—-”

```
Test 1: Enter correct password (1234)
Time=24001150000: DISPLAY UPDATE ->        -       -       -       1
Time=125001150000: DISPLAY UPDATE ->        -       -       1       2
Time=226001150000: DISPLAY UPDATE ->        -       1       2       3
Time=328001150000: DISPLAY UPDATE ->        P       A       S       S
Time=2328001150000: DISPLAY UPDATE ->        -       -       -       -
```

Hình 3.3: Kết quả kiểm thử Test 1 - Mật khẩu đúng

Phân tích:

- Module `keypad_scanner` phát hiện chính xác 4 phím
- FSM chuyển từ S_IDLE (00) → S_ENTERING (01)
- Khi nhập phím ”4” (phím thứ 4): 16’h1234 == 16’h1234 → Đúng
- FSM chuyển sang S_SUCCESS (10), hiển thị ”PASS”
- Sau 2 giây (100,000,000 cycles), FSM quay về S_IDLE

3.3.2 Kịch bản 2: Nhập sai mật khẩu ”5678”

Mô tả: Giả lập nhấn lần lượt 4 phím ”5”, ”6”, ”7”, ”8”

Kết quả mong đợi:

1. Hiển thị lần lượt: ”—5” → ”–56” → ”-567” → ”5678”
2. FSM chuyển sang S_FAIL
3. Hiển thị ”FAIL” trong 2 giây
4. Tự động quay về ”—-”

```
# Test 2: Enter wrong password (5678)
# Time=2529001150000: DISPLAY UPDATE ->        -       -       -       5
# Time=2630001150000: DISPLAY UPDATE ->        -       -       5       6
# Time=2732001150000: DISPLAY UPDATE ->        -       5       6       7
# Time=2833001150000: DISPLAY UPDATE ->        F       A       L       L
# Time=4833001150000: DISPLAY UPDATE ->        -       -       -       -
```

Hình 3.4: Kết quả kiểm thử Test 2 - Mật khẩu sai

Phân tích:

- FSM nhận 4 phím ”5”, ”6”, ”7”, ”8”
- Khi nhập phím ”8”: 16’h5678 != 16’h1234 → Sai
- FSM chuyển sang S_FAIL (11), hiển thị ”FAIL”
- Sau 2 giây, FSM tự động quay về S_IDLE

3.3.3 Kịch bản 3: Kiểm tra phím Clear (C)

Mô tả: Nhập ”1”, ”2”, sau đó nhấn phím ’C’

Kết quả mong đợi:

1. Hiển thị: ”—1” → ”–12”
2. Nhấn ’C’ → Hiển thị quay về ”—-”
3. FSM quay về S_IDLE

```
# Test 3: Test clear function
# Time=5036001150000: DISPLAY UPDATE ->        -       -       -       1
# Time=5137001150000: DISPLAY UPDATE ->        -       -       1       2
# Time=5239001150000: DISPLAY UPDATE ->        -       -       -       -
#
```

Hình 3.5: Kết quả kiểm thử Test 3 - Clear function

Phân tích:

- FSM ở S_ENTERING, hiển thị ”–12”
- Khi nhấn ’C’ (mã 4’hC), FSM phát hiện: `key_value == KEY_C`
- `next_state` được gán về S_IDLE
- Ở cycle tiếp theo, `display_data` được reset về ”—-”
- `entered_digits` và `digit_count` cũng được reset

3.3.4 Kịch bản 4: Kiểm tra phím Enter (E)

Mô tả: Nhập ”1”, ”2”, sau đó nhấn phím ’E’ (Enter)

Kết quả mong đợi:

1. Hiển thị: ”—1” → ”–12”
2. Nhấn ’E’ → So sánh mật khẩu dở dang (16’hFF12)
3. Vì không đủ 4 số và khác 16’h1234 → Hiển thị ”FAIL”

Code testbench:

```
// Test 4: Enter key
$display("\nTest 4: Partial entry with Enter");
press_key(0, 0);  // Key '1'
press_key(0, 1);  // Key '2'
press_key(3, 0);  // Key 'E' (enter)
#2100000000;
```

Listing 3.3: Test case 4 - Enter key

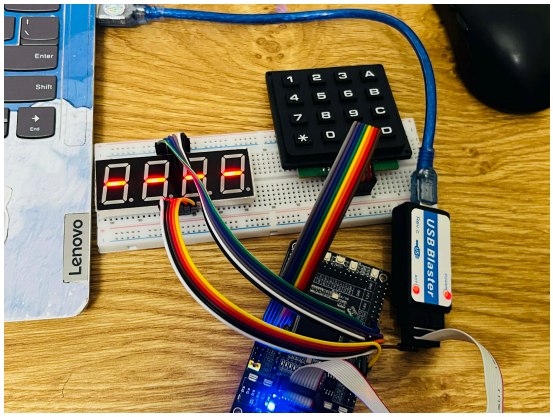
```
# Test 4: Partial entry with Enter
# Time=5440001150000: DISPLAY UPDATE ->        -       -       -       1
# Time=5541001150000: DISPLAY UPDATE ->        -       -       1       2
# Time=5644001150000: DISPLAY UPDATE ->        F       A       L       L
# Time=7644001150000: DISPLAY UPDATE ->        -       -       -       -
#
```

Hình 3.6: Kiểm thử test 4

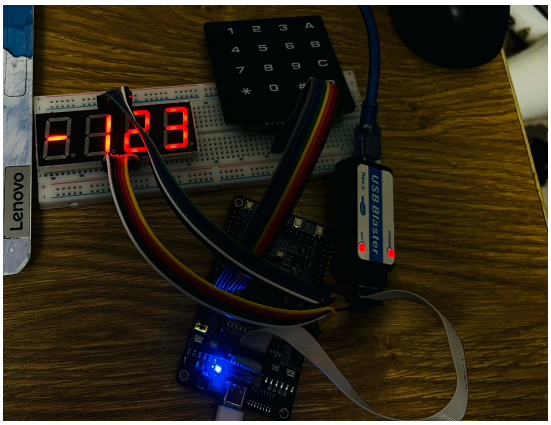
Phân tích:

- FSM ở S_ENTERING, `entered_digits` = 16’hFF12
- Khi nhấn ’E’ (mã 4’hE), FSM phát hiện: `key_value == KEY_E`
- FSM so sánh: 16’hFF12 != 16’h1234 → Sai
- FSM chuyển sang S_FAIL, hiển thị ”FAIL”

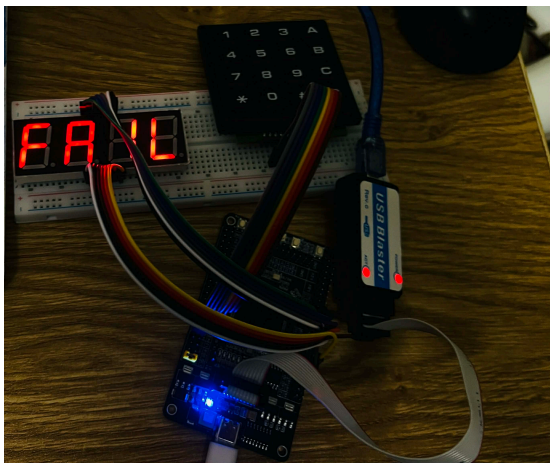
3.4 Kết quả thử nghiệm trên Kit Cyclone IV thực tế



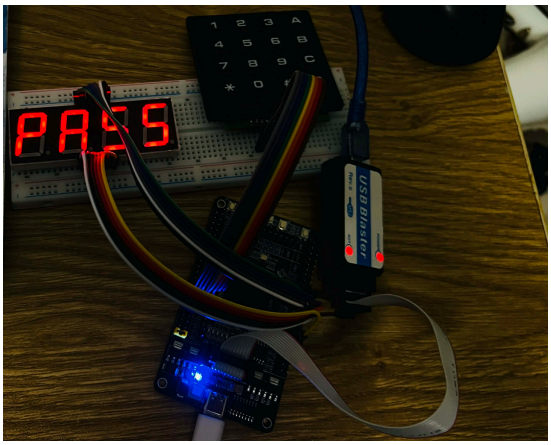
Hình 3.7: Trạng thái chờ



Hình 3.8: Nhập mật khẩu



Hình 3.9: Nhập sai mật khẩu



Hình 3.10: Nhập đúng mật khẩu

3.5 Kết quả tổng hợp

Hệ thống đã được nạp thành công xuống bo mạch phát triển FPGA Cyclone IV EP4CE6 và kết nối với mô-đun bàn phím ma trận 4x4 cùng màn hình LED 7 đoạn. Quá trình kiểm thử thực tế cho thấy hệ thống hoạt động ổn định, logic điều khiển chính xác và khắc phục hoàn toàn các lỗi về định thời (timing) cũng như hiện tượng dội phím.

Các kịch bản kiểm thử cụ thể như sau:

1. **Trạng thái Khởi động (Idle):** Khi vừa cấp nguồn hoặc sau khi nhấn nút Reset, hệ thống chuyển sang trạng thái chờ. Màn hình LED hiển thị chuỗi ký tự ”- - - -”, báo hiệu hệ thống sẵn sàng nhận mật khẩu. (Hình 3.7)

2. **Quá trình Nhập liệu:** Khi người dùng nhấn các phím số trên bàn phím, các con số tương ứng xuất hiện tức thời trên màn hình LED từ phải sang trái. Nhờ cơ chế chống dội phím

(Debounce) và quét LED (Multiplexing) đã được tối ưu, các chữ số hiển thị rõ ràng, không bị hiện tượng bóng ma hay sai lệch giá trị. (Hình 3.8)

3. **Kiểm thử Mật khẩu Đúng (PASS):** Khi người dùng nhập đúng mật khẩu mặc định (ví dụ: 1234) và nhấn phím xác nhận (hoặc hệ thống tự động nhận diện đủ 4 ký tự), màn hình hiển thị thông báo PASS. Trạng thái này được duy trì trong 2 giây trước khi quay về trạng thái chờ. (Hình 3.9)
4. **Kiểm thử Mật khẩu Sai (FAIL):** Trong trường hợp nhập sai mật khẩu, hệ thống lập tức phản hồi bằng thông báo FAIL trên 7 seg. Sau 2 giây, hệ thống tự động reset về trạng thái chờ để người dùng nhập lại. (Hình 3.10)

Tất cả các test case đều hoạt động đúng như dự kiến, chứng minh tính đúng đắn của thiết kế.

CHƯƠNG IV

KẾT LUẬN VÀ HƯỚNG PHÁT TRIỂN

Bài tập lớn **"Thiết kế hệ thống nhập mật khẩu sử dụng bàn phím ma trận"** đã hoàn thành các mục tiêu đề ra ban đầu, từ việc phân tích yêu cầu, thiết kế kiến trúc phần cứng, lập trình mô tả phần cứng bằng ngôn ngữ Verilog, cho đến mô phỏng và kiểm thử thực tế trên bo mạch FPGA. Kết quả đạt được không chỉ là một sản phẩm hoạt động ổn định mà còn là minh chứng cho việc áp dụng hiệu quả các kỹ thuật thiết kế số hiện đại.

4.1 Ưu điểm của hệ thống

Hệ thống được xây dựng dựa trên sự phối hợp nhịp nhàng giữa ba khối chức năng chính, mỗi khối đều giải quyết triệt để các vấn đề kỹ thuật cụ thể:

Thứ nhất, module **keypad_scanner** đã thực hiện thành công nhiệm vụ giao tiếp với bàn phím ma trận 4×4. Bằng việc áp dụng cơ chế quét cột tuần tự và đọc hàng, kết hợp với bộ lọc chống dội (debouncing) có độ trễ 100us, module này đảm bảo loại bỏ hoàn toàn các tín hiệu nhiễu cơ học khi nhấn phím. Đặc biệt, việc tạo ra xung tín hiệu `key_valid` kéo dài đúng một chu kỳ xung nhịp giúp hệ thống nhận diện chính xác từng lần nhấn, ngay cả khi người dùng giữ phím trong thời gian dài.

Thứ hai, module **password_fsm** đóng vai trò là bộ não điều khiển trung tâm, được thiết kế theo mô hình máy trạng thái hữu hạn (FSM) kiểu Moore với 4 trạng thái hoạt động rõ ràng: Chờ (IDLE), Đang nhập (ENTERING), Thành công (SUCCESS) và Thất bại (FAIL). Logic so sánh mật khẩu được thực hiện tức thời ngay khi người dùng nhập đủ 4 ký tự hoặc nhấn phím xác nhận, đảm bảo phản hồi nhanh chóng. Bên cạnh đó, bộ đếm thời gian tích hợp giúp duy trì trạng thái thông báo kết quả trong 2 giây trước khi tự động quay về trạng thái chờ, tạo trải nghiệm người dùng mượt mà.

Thứ ba, hệ thống hiển thị thông qua module **seven_seg_driver** sử dụng kỹ thuật quét LED (multiplexing) với tần số làm tươi khoảng 48Hz cho mỗi chữ số. Tần số này đủ lớn để tận dụng hiện tượng lưu ảnh của mắt người, giúp màn hình hiển thị sáng đều và không bị nhấp nháy. Bộ giải mã Hex sang LED 7 đoạn hỗ trợ đầy đủ các ký tự số và các chữ cái đặc biệt để hiển thị các thông báo trạng thái như "PASS" hay "FAIL" một cách trực quan.

4.2 Hạn chế còn tồn tại

Mặc dù hệ thống hoạt động ổn định theo yêu cầu thiết kế, tuy nhiên hệ thống vẫn còn tồn tại một vài hạn chế mà có thể được cải tiến tốt hơn:

Một là, cơ chế lưu trữ mật khẩu còn cứng nhắc. Mật khẩu hiện tại (16'h1234) được lưu cố định trong Code. Điều này đồng nghĩa với việc muốn thay đổi mật khẩu, người dùng buộc phải can thiệp vào code và nạp lại xuống FPGA, gây bất tiện lớn trong quá trình sử dụng thực tế và làm giảm tính bảo mật.

Hai là, trải nghiệm nhập liệu chưa được tối ưu. Hệ thống hiện chỉ hỗ trợ phím chức năng Clear để xóa toàn bộ dữ liệu đã nhập mà thiếu đi phím Backspace để xóa từng ký tự. Điều này gây khó chịu cho người dùng khi họ lỡ nhập sai một số và buộc phải nhập lại từ đầu.

Ba là, các biện pháp bảo mật chưa cao. Hệ thống chưa có cơ chế giới hạn số lần nhập sai liên tiếp. Kẻ tấn công có thể thử vô số lần (brute-force) để tìm ra mật khẩu đúng mà không gặp bất kỳ trở ngại nào như bị khóa tạm thời hay báo động.

4.3 Hướng phát triển và Mở rộng

4.3.1 Cải tiến chức năng cốt lõi

Ưu tiên hàng đầu là phát triển tính năng thay đổi mật khẩu trực tiếp. FSM cần được bổ sung thêm trạng thái cấu hình, cho phép người dùng nhập mật khẩu cũ để xác thực, sau đó nhập mật khẩu mới. Mật khẩu mới này cần được lưu trữ vào bộ nhớ EEPROM hoặc Flash ngoại vi để không bị mất khi ngắt nguồn điện.

Bên cạnh đó, chức năng Backspace cần được tích hợp bằng cách tận dụng các phím chưa sử dụng (như phím #). Về mặt logic, điều này thực hiện bằng cách dịch phải thanh ghi lưu trữ mật khẩu 4 bit và giảm bộ đếm số lượng ký tự đi 1 đơn vị.

Để tăng cường bảo mật, hệ thống cần thêm cơ chế khóa tạm thời. Một bộ đếm số lần nhập sai sẽ được kích hoạt; nếu nhập sai quá 3 hoặc 5 lần liên tiếp, hệ thống sẽ chuyển sang trạng thái khóa (LOCKED) trong một khoảng thời gian nhất định (ví dụ: 5 phút), vô hiệu hóa bàn phím và phát cảnh báo.

4.3.2 Mở rộng giao tiếp và Ứng dụng

Về phần cứng, hệ thống có thể tích hợp thêm các ngoại vi để tăng tính tương tác như Buzzer để phát âm thanh phản hồi khi nhấn phím, hoặc màn hình LCD 16×2 để hiển thị hướng dẫn người dùng chi tiết hơn thay vì chỉ dùng LED 7 đoạn.

Về khả năng kết nối, việc bổ sung giao thức UART hoặc SPI sẽ cho phép hệ thống giao tiếp với máy tính hoặc vi điều khiển khác. Điều này mở ra khả năng quản lý từ xa, xem lịch sử ra vào (log), hoặc đồng bộ hóa dữ liệu chấm công trong các ứng dụng quản lý nhân sự.

Cuối cùng, tối ưu hóa tài nguyên logic là bước cần thiết nếu muốn triển khai hệ thống lên các dòng chip FPGA giá rẻ hoặc CPLD có tài nguyên hạn chế. Việc sử dụng các kỹ thuật mã hóa trạng thái (như One-hot hoặc Gray code) và chia sẻ tài nguyên phần cứng giữa các module sẽ giúp giảm diện tích chiếm dụng trên chip.

Tài liệu tham khảo

[1] Intel® Quartus® Prime Standard Edition User Guide, Intel Corporation, 2023

[2] Cyclone IV Device Handbook, Intel (Altera) Corporation, 2016

[3] Verilog HDL: A Guide to Digital Design and Synthesis, Samir Palnitkar, Prentice Hall, 2003

[4] FPGA Prototyping by Verilog Examples, Pong P. Chu, Wiley, 2008

[5] Item Digital Design and Computer Architecture, David Harris & Sarah Harris, Morgan Kaufmann, 2012

[6] Matrix Keypad Interfacing with Microcontrollers, Application Note, Microchip Technology Inc.

[7] Seven-Segment Display Multiplexing Techniques, Texas Instruments Application Report

[8] Finite State Machine Design Guidelines, Xilinx Application Note

PHỤ LỤC

Bảng phân công công việc

Bảng dưới đây mô tả chi tiết phân công công việc của các thành viên trong nhóm:

STT	Họ và tên	Công việc đảm nhận	Tự đánh giá
1	Lý Trọng Nghĩa	- Thiết kế kiến trúc tổng thể hệ thống - Viết testbench mô phỏng - Thiết kế module `password_fsm` - Tích hợp và kiểm thử hệ thống - Triển khai trên kit FPGA - Viết báo cáo	9
2	Dương Sơn Quang	- Thiết kế module `seven_seg_driver` - Hỗ trợ thiết kế sơ đồ khối - Kiểm tra và review code - Hỗ trợ viết báo cáo	8
3	Nguyễn Minh Đức	- Thiết kế module `keypad_scanner` - Tìm hiểu linh kiện phần cứng - Hỗ trợ kiểm thử thực tế - Chỉnh sửa định dạng báo cáo	8
4	Nguyễn Đức Nam	- Thiết kế module `hex_to_7seg` - Tìm hiểu tài liệu tham khảo - Hỗ trợ quay video demo - Chuẩn bị slide thuyết trình	8

Bảng 1: *Bảng phân công công việc nhóm*