

逢甲大學學生報告 ePaper

報告題名： 極限記憶挑戰

Brain Flash

作者：吳姿樺、梁哲豪、張瑜珊、溫昀蓁

系級：電子工程學系二年乙班

學號：D1321629、D1350866、D1351217、D1351281

開課老師：楊鎮安老師

課程名稱：微處理機系統

開課系所：電子系

開課學年： 114 學年度 第二學期





中文摘要

- (1) 目的：本專題的目的是參考市面上經典的互動式益智玩具「Simon Game（西門遊戲）」，透過數位邏輯與硬體電路設計，在 FPGA 平台上實現一個能夠訓練玩家記憶力、專注力與即時反應能力的互動遊戲系統。
- (2) 過程及方法：在實作過程中，主要參考開源的 Simon game 相關設計邏輯與架構，並使用 Terasic DE0 (Cyclone III) 開發板作為核心處理與硬體控制設備。方法上透過硬體描述語言（如 Verilog HDL）來規劃有限狀態機（FSM）、時脈除頻器、開關與按鍵輸入處理、防彈跳電路、以及 7 段顯示器和 LED 燈號輸出控制，完成遊戲核心邏輯的編寫、接腳配置（Pin Assignment）、電路合成、模擬驗證與實機測試。
- (3) 結果：研究過程最終成功在 DE0 開發板上完整實現了 Simon 遊戲的燈光序列展示、玩家按鍵輸入、序列比對及錯誤判定等互動功能。此硬體實現成果不僅能作為數位邏輯實習與 FPGA 設計課程的優異實作範例，亦可推廣應用於嵌入式系統教學、互動娛樂裝置開發以及數位電路設計等實務領域。

關鍵字：（請填寫與本報告相關的重要名詞 3 至 5 個）

狀態機、記憶、遊戲、Flash、Verilog

Abstract

(1) Objective: The purpose of this project is to refer to the classic interactive puzzle game "Simon Game" available on the market, and through digital logic and hardware circuit design, implement an interactive game system on an FPGA platform that can train players' memory, concentration, and real-time reaction abilities.

(2) Process and Methods: During the implementation process, the design logic and architecture related to the open-source Simon game were primarily referenced, utilizing the Terasic DE0 (Cyclone III) development board as the core processing and hardware control equipment. The methodology involves using hardware description languages (such as Verilog HDL) to plan finite state machines (FSM), clock dividers, switch and button input processing, debouncing circuits, and 7-segment display and LED output control, thereby completing the core game logic coding, pin assignment, circuit synthesis, simulation verification, and physical testing.

(3) Results: The research process ultimately and successfully implemented the interactive functions of the Simon game on the DE0 development board, including light sequence display, player button input, sequence comparison, and error judgment. This hardware implementation result can not only serve as an excellent practical example for digital logic laboratory and FPGA design courses, but can also be widely applied to fields such as embedded system teaching, interactive entertainment device development, and digital circuit design.

Keyword :

Finite State Machine (FSM) 、Memory 、Game 、Flash 、Verilog

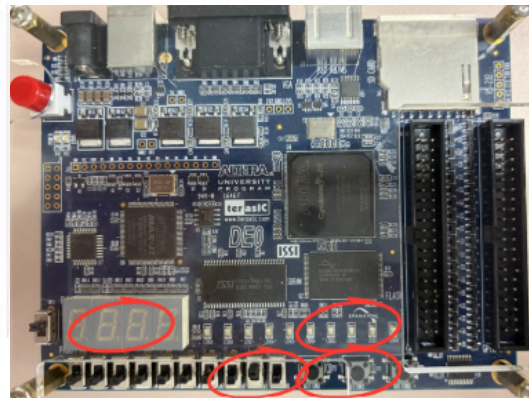
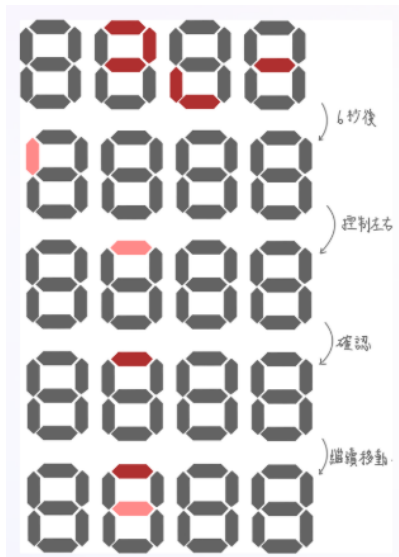
目 次

- 一、 遊戲規則
- 二、 原理流程
- 三、 程式碼
- 四、 波形呈現
- 五、 困難與解決
- 六、 影片

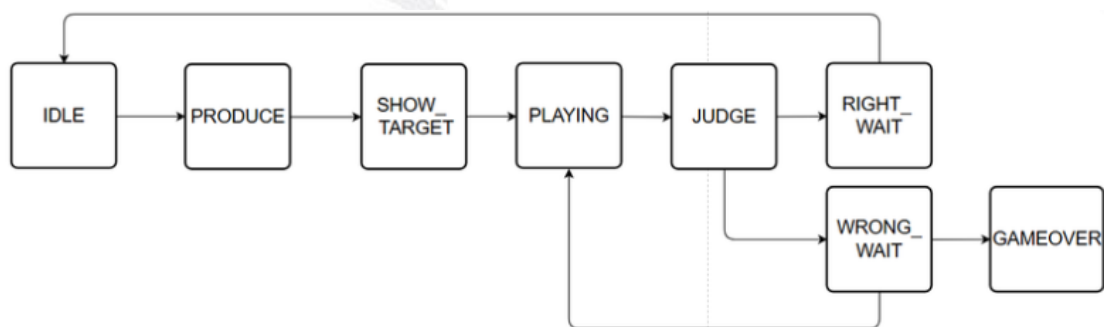


一、 遊戲規則

1. 將切換開關 sw[2] 拉起，即可正式啟動遊戲系統
2. 題目會隨機顯示在面板左側的四個七段顯示器中，並於六秒後消失。
3. 利用 button[0] 進行右移、button[1] 進行左移，將燈條移動至目標位置後，按下 button[2] 確認燈條位置，並重複此步驟直到完成所有位置設定
4. 若在答題過程中出現失誤，可拉起 sw[0] 清除燈條重新輸入；當燈條顯示與題目相符時，拉起 sw[1] 上傳答案
5. 回答正確將晉級至下一關，回答錯誤則會扣減 LED 顯示的生命值；若生命值歸零，七段顯示器將顯示「END」以表示遊戲結束。



二、 原理流程圖



idle：閒置狀態，沒有作動

Produce：利用亂數產生題目，並儲存

show_target：展示題目 6 秒

playing：玩家作答

judge：判斷玩家答案是否正確 & 遊戲是否結束

right_wait：七段閃爍 6 秒表示正確，再出下一題

wrong_wait：讓玩家重新答題

game_over：展示遊戲結束畫面

三、 程式碼

1. 亂數產生

這段程式碼使用 32 位元線性回授移位暫存器 (LFSR) 來產生亂數：當重置時會載入初始種子值 32'hACE1，並在系統處於閒置或產生狀態時，透過不斷向左位移與回授來產生新的亂數序列；同時，利用組合邏輯的迴圈與計數器來統計暫存器中數值為 1 的位元數量，以作為後續遊戲題目的數值來源。

```
always @(posedge clk or negedge rst_n) begin
    if (!rst_n) LFSR <= 32'hACE1;
    else if (state == idle || state == produce) begin
        LFSR <= {LFSR[30:0], LFSR_feedback};
    end
end

integer i;
always @(*) begin
    LFSR_count = 5'd0;
    for (i=0; i<28; i=i+1) begin
        if (LFSR[i] == 1'b1) LFSR_count = LFSR_count + 1;
    end
end
```

2. 七段亮燈解碼與游標

段程式碼主要用於控制 7 段顯示器上的游標或特定線段顯示：上方透過 case 敘述根據 seg_cursor 決定點亮哪一個 7 段顯示器的某個特定區段；下方則透過 assign 敘述與條件運算子，依據 hex_cursor 的數值，將解碼後的 7 段顯示資料放入 28 位元輸出訊號 cursor 的對應區段位置中，藉此在不同的顯示器上呈現如圖示中依序移動的高亮橫線或區段

```
always @(*) begin
    seg_decoded = 7'b0;
    case (seg_cursor)
        seg_a: seg_decoded[0] = 1'b1;
        seg_b: seg_decoded[1] = 1'b1;
        seg_c: seg_decoded[2] = 1'b1;
        seg_d: seg_decoded[3] = 1'b1;
        seg_e: seg_decoded[4] = 1'b1;
        seg_f: seg_decoded[5] = 1'b1;
        seg_g: seg_decoded[6] = 1'b1;
        default: seg_decoded = 7'b0;
    endcase
end

assign cursor = (hex_cursor == 2'b00)? {21'b0, seg_decoded}:
    ((hex_cursor == 2'b01)? {14'b0, seg_decoded, 7'b0}:
    ((hex_cursor == 2'b10)? {7'b0, seg_decoded, 14'b0}:
    {seg_decoded, 21'b0}));
```



3. 遊戲主程式

```

95 always @(posedge clk or negedge rst_n) begin
96     if (rst_n) begin
97         life
98         player_seg <= 28'b0;
99         target_seg <= 28'b0;
100        state <= idle;
101        timer <= 2'd0;
102        hex_cursor <= 2'b11;
103        seg_cursor <= seg_f;
104        updata_lock <= 1'b0;
105    end
106    else begin
107        case (state)
108            idle: state <= produce;
109            produce: begin
110                if (LFSR_count >= 5'd4 && LFSR_count <= 5'd15) begin
111                    target_seg <= LFSR[27:0];
112                    timer <= 2'd0;
113                    state <= show_target;
114                end
115                else state <= produce;
116            end
117            show_target: begin
118                if (clk_2hz_pos) begin
119                    if (timer == 2'd2) begin
120                        state <= playing;
121                        hex_cursor <= 2'b11;
122                        seg_cursor <= seg_f;
123                    end
124                    else timer <= timer + 1'b1;
125                end
126            end
127        end
128    end
129 end

```

```

playing: begin
    if (player_reset) begin
        player_seg <= 28'b0;
        hex_cursor <= 2'b11;
        seg_cursor <= seg_f;
    end
    else if (updata && !updata_lock) begin
        state <= judge;
        updata_lock <= 1'b1;
    end
    else begin
        if (!updata) updata_lock <= 1'b0;
        if (btn_right) begin
            if (seg_cursor == seg_c) begin
                seg_cursor <= seg_f;
                hex_cursor <= hex_cursor - 1'b1;
            end else begin
                seg_cursor <= seg_cursor + 1'b1;
            end
        end
    end
end

```

```

    else if (btn_left) begin
        if (seg_cursor == seg_f) begin
            seg_cursor <= seg_c;
            hex_cursor <= hex_cursor + 1'b1;
        end else begin
            seg_cursor <= seg_cursor - 1'b1;
        end
    end
    else if (btn_confirm) begin
        if ((cursor & player_seg) == 28'b0) begin
            player_seg <= player_seg | cursor;
        end
        else begin
            player_seg <= player_seg;
        end
    end
end
end

```

```

judge: begin
    if (player_seg == target_seg) begin
        player_seg    <= 28'b0;
        timer         <= 2'd0;
        state         <= right_wait;
    end
    else begin
        if(life == 5'b00001) begin
            life <= 5'b00000;
            state <= game_over;
        end
        else begin
            life <= life >> 1;
            state <= wrong_wait;
        end
    end
end
end

```

```

right_wait: begin
    if (clk_2hz_pos) begin
        life <= life;
        if (timer == 2'd2) begin
            if (!updata) begin
                state <= idle;
                updata_lock <= 1'b0;
            end
        end
        else timer <= timer + 1'b1;
    end
end
wrong_wait: begin
    if (!updata) begin
        updata_lock <= 1'b0;
        state <= playing;
    end
end
game_over: state <= game_over;
default: state <= idle;
endcase
end
end

```

```

always @(*) begin
    case (state)
        show_target: seg_output = target_seg;
        playing: seg_output = player_seg | (flash ? cursor : 28'b0);
        wrong_wait: seg_output = player_seg;
        right_wait: seg_output = clk_2hz ? 28'hFFF_FFFF : 28'b0;
        game_over: seg_output = 28'h01E6A5E;
        default: seg_output = player_seg;
    endcase
end
always @(*) begin
    hex0 = ~seg_output[ 6: 0];
    hex1 = ~seg_output[13: 7];
    hex2 = ~seg_output[20:14];
    hex3 = ~seg_output[27:21];
end
endmodule

```

4. Clk_gen & button_debounce

參考文獻

- ISmillex. n.d. "Simon-Game-FPGA." GitHub repository. Accessed September 30, 2026. <https://github.com/ISmillex/Simon-Game-FPGA>.

