

2025 Digital IC Design Homework 2

NAME	黃啟桓			
Student ID	P76134927			
Functional Simulation Result				
Pattern1	Pattern2	Pattern3	Pattern4	Pattern5
Pass	Pass	Pass	Pass	Pass
Pattern 1				
<pre> # All data have been generated successfully! # # // # / __ # / Congratulations !! / / 0.0 # / / # / Simulation PASS !! / /_____\ # / / # / / ^ ^ ^ ^ \ w # / / ^ ^ ^ ^ \ w # // \m__m_ _ # # ** Note: \$finish : E:/GG/Code/DIC_HW/HW2/file/testfixture.sv(162) # Time: 2355 ns Iteration: 0 Instance: /testfixture # </pre>				
Pattern 2				
<pre> # All data have been generated successfully! # # // # / __ # / Congratulations !! / / 0.0 # / / # / Simulation PASS !! / /_____\ # / / # / / ^ ^ ^ ^ \ w # / / ^ ^ ^ ^ \ w # // \m__m_ _ # # ** Note: \$finish : E:/GG/Code/DIC_HW/HW2/file/testfixture.sv(162) # Time: 2355 ns Iteration: 0 Instance: /testfixture # </pre>				
Pattern 3				
<pre> # All data have been generated successfully! # # // # / __ # / Congratulations !! / / 0.0 # / / # / Simulation PASS !! / /_____\ # / / # / / ^ ^ ^ ^ \ w # / / ^ ^ ^ ^ \ w # // \m__m_ _ # # ** Note: \$finish : E:/GG/Code/DIC_HW/HW2/file/testfixture.sv(162) # Time: 2355 ns Iteration: 0 Instance: /testfixture # </pre>				
Pattern 4				

```

# All data have been generated successfully!
#
#
#          ///////////////////////////////////////////////////
#          /               /               /  |__||
#          / Congratulations !! /         / 0.0 |
#          /               /         /_____|
#          / Simulation PASS !! /      / ^ ^ ^ \ |
#          /               /      / ^ ^ ^ ^ \w|
#          ///////////////////////////////////////////////////      \m__m_|_
#
#
# ** Note: $finish      : E:/GG/Code/DIC_HW/HW2/file/testfixture.sv(162)
#      Time: 2355 ns   Iteration: 0   Instance: /testfixture
#

```

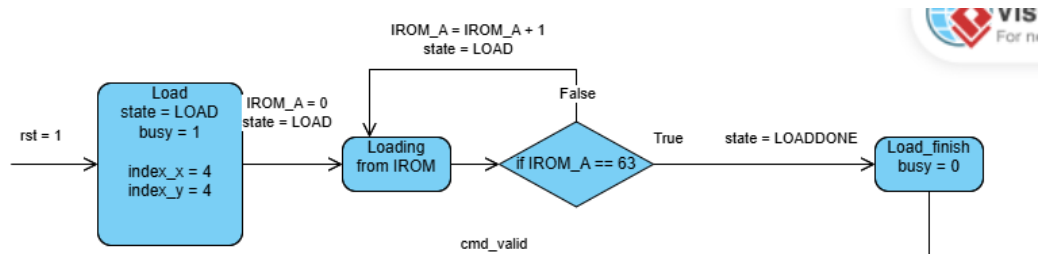
Pattern 5

```

# All data have been generated successfully!
#
#
#          ///////////////////////////////////////////////////
#          /               /               /  |__||
#          / Congratulations !! /         / 0.0 |
#          /               /         /_____|
#          / Simulation PASS !! /      / ^ ^ ^ \ |
#          /               /      / ^ ^ ^ ^ \w|
#          ///////////////////////////////////////////////////      \m__m_|_
#
#
# ** Note: $finish      : E:/GG/Code/DIC_HW/HW2/file/testfixture.sv(162)
#      Time: 2355 ns   Iteration: 0   Instance: /testfixture
#

```

LCD_CTRL Finite-State Machine Design:



我把有限狀態機拆成多個 submachine, 第一部分是 load 區域, 只有處理完 load(state = LOADDONE)才能讀取/執行指令。rst 進來之後, state 設為 LOAD 先維持住 Load 狀態, 每次 clk 遞增 IROM_A, 讀取 IROM_Q, 存到 LCD controller 的 memory。直到 IROM_A==63, 也會讀取 IROM_Q, 再下一個 clk 之後才會改變狀態為 LOADDONE。說明 Load 結束

```

always @(posedge clk) begin
    if (state == LOAD) begin
        memory[IROM_A] <= IROM_Q;
    end
end

always @(posedge clk or posedge rst) begin
    if (rst) begin
        IROM_rd    <= 1'b1; // 复位后立刻开始读 IROM
        IROM_A     <= 6'd0; // 地址从 0 开始
        state      <= LOAD;
        busy       <= 1'b1;
        done       <= 1'b0;
        index_x    <= 4;
        index_y    <= 4;
    end else begin
        case (state)
            LOAD: begin
                // 存入 memory
                if (IROM_A == 6'd63) begin
                    IROM_rd <= 1'b0; // 读取完成
                    state  <= LOADDONE; // 进入下一个阶段
                    busy   <= 1'b0;
                end else begin
                    IROM_A <= IROM_A + 1; // 地址递增
                end
            end
            LOADDONE: begin
                IROM_A <= 6'd0;
            end
        endcase
    end
end
end

```

=====

cmd 和 cmd_valid 進來時，有一個 command classification 分類 cmd 當遇到 Write/Max/Min/Avg 會 hold cmd 與初始化 4 * 4 的 table。遇到 shift 就簡單的變化 index_x 與 index_y。

```

always @(posedge clk) begin
    if (cmd_valid && state == LOADDONE && !busy) begin
        busy    <= 1'b1;
        done    <= 1'b0;
        count   <= 6'd0;

        case (cmd)
            WRITE: begin
                IRAM_ceb <= 1'b1;
                IRAM_web <= 1'b0;
                IRAM_A   <= 6'd0;
                IRAM_D   <= 8'd0;
                process  <= WRITE;
            end

            SHIFT_UP: begin
                if (index_y > 2) begin
                    index_y <= index_y - 1;
                end
                process = SHIFT_UP;
            end

            SHIFT_DOWN: begin
                if (index_y < 6) begin
                    index_y <= index_y + 1;
                end
                process = SHIFT_DOWN;
            end

            SHIFT_LEFT: begin
                if (index_x > 2) begin
                    index_x <= index_x - 1;
                end
                process = SHIFT_LEFT;
            end

            SHIFT_RIGHT: begin
                if (index_x < 6) begin
                    index_x <= index_x + 1;
                end
                process = SHIFT_RIGHT;
            end

            MAX, MIN, AVG: begin
                region_memory = {
                    memory[8 * index_y + index_x - 18], memory[8 * index_y + index_x - 17],
                    memory[8 * index_y + index_x - 16], memory[8 * index_y + index_x - 15],
                    memory[8 * index_y + index_x - 10], memory[8 * index_y + index_x - 9],
                    memory[8 * index_y + index_x - 8 ], memory[8 * index_y + index_x - 7],
                    memory[8 * index_y + index_x - 2 ], memory[8 * index_y + index_x - 1],
                    memory[8 * index_y + index_x ], memory[8 * index_y + index_x + 1],
                    memory[8 * index_y + index_x + 6 ], memory[8 * index_y + index_x + 7],
                    memory[8 * index_y + index_x + 8 ], memory[8 * index_y + index_x + 9]
                };
                cmd_count <= 4'd0;
                process = cmd;
                calculate_done = 1'b0;
            end
            default: begin
            end
        endcase
    end
end

```

接著就是執行長的指令(Write/Min/Max/Avg)， “process” 變數會儲存 hold cmd。Write 時，每個 clk，一步一步的將 LCD memory 的值傳給 IRAM，Write 完即回傳 Done。Min/Max/Avg 時，針對 4*4 的 table，一步一步運算，最後再寫回去 LCD memory，完成後 busy = 0。Shift 已經處理完了，所以就 busy = 0

```
// process
always @(posedge clk) begin
    if (busy) begin
        case (process)
            WRITE: begin
                IRAM_A = IRAM_A + 1;
                IRAM_D <= memory[IRAM_A];
                if (IRAM_A == 6'd63) begin
                    process <= DONE;
                end
            end
            SHIFT_UP, SHIFT_DOWN, SHIFT_LEFT, SHIFT_RIGHT: begin
                process = NONE;
            end
            MAX, MIN, AVG: begin
                if (!calculate_done) begin
                    if (cmd_count == 4'd0) begin
                        max_value <= region_memory[cmd_count*8 +: 8];
                        min_value <= region_memory[cmd_count*8 +: 8];
                        sum <= region_memory[cmd_count*8 +: 8];
                    end
                    else begin
                        if (region_memory[cmd_count*8 +: 8] > max_value)
                            max_value <= region_memory[cmd_count*8 +: 8];
                        if (region_memory[cmd_count*8 +: 8] < min_value)
                            min_value <= region_memory[cmd_count*8 +: 8];
                        sum <= sum + region_memory[cmd_count*8 +: 8];
                    end

                    cmd_count <= cmd_count + 1;
                    if (cmd_count == 4'd15) begin
                        calculate_done = 1'b1;
                    end
                end
                else begin
                    modified_value = (process == MAX) ? max_value : (process == MIN) ? min_value : sum >> 4;
                    {
                        memory[8 * index_y + index_x - 18], memory[8 * index_y + index_x - 17],
                        memory[8 * index_y + index_x - 16], memory[8 * index_y + index_x - 15],
                        memory[8 * index_y + index_x - 10], memory[8 * index_y + index_x - 9],
                        memory[8 * index_y + index_x - 8 ], memory[8 * index_y + index_x - 7],
                        memory[8 * index_y + index_x - 2 ], memory[8 * index_y + index_x - 1],
                        memory[8 * index_y + index_x ], memory[8 * index_y + index_x + 1],
                        memory[8 * index_y + index_x + 6 ], memory[8 * index_y + index_x + 7],
                        memory[8 * index_y + index_x + 8 ], memory[8 * index_y + index_x + 9]
                    } = {16{modified_value}};
                    process = NONE;
                end
            end
        endcase
    end
    DONE: begin
        done <= 1'b1;
        busy <= 1'b0;
        IRAM_web <= 1'b1;
        IRAM_ceb <= 1'b0;
    end
    NONE: begin
        done <= 1'b0;
        busy <= 1'b0;
        IRAM_web <= 1'b1;
        IRAM_ceb <= 1'b0;
        process = NONE;
    end
end
```

