Digital System Design and Implementation

Lab #3

姓名:黃鉦淳 學號:108303013

a. Verilog codes

```
`timescale 1ns / 1ps

module Main(clk_100MHz, Reset, Button_PIN, PS2_CLK, PS2_DATA, Seg_G0_En,
Seg_G1_En, Seg_G0, Seg_G1, LED_PORT);

// ============ [Parameter] ============ //
parameter   State_Idle    = 2'h0,
           State_Capture = 2'h1,
           State_Result  = 2'h2;
parameter   Grade_Ball_R      = 2'h0,
           Grade_Ball_B      = 2'h1,
           Grade_Ball_K      = 2'h2,
           Grade_Ball_Unknown = 2'h3;
parameter   Unknown = "-";
parameter   Name_Pokemon_Unknown  = {4{Unknown}},
           Name_Pokemon_Pikachu  = "PICA",
           Name_Pokemon_Rattata  = "RATT",
           Name_Pokemon_Caterpie = "CATE";
parameter   Grade_Pokemon_Unknown  = 2'h3,
           Grade_Pokemon_Pikachu  = 2'h2,
           Grade_Pokemon_Rattata  = 2'h1,
           Grade_Pokemon_Caterpie = 2'h0;
// 0 <= I0 < 4 < I1 < 9 < I2 <= 15
parameter   I0 = 4'd01,
           I1 = 4'd05,
           I2 = 4'd11;

// ============== [I/O] ============== //
input clk_100MHz, Reset;
```

```
input [2:0] Button_PIN; // [2] = S4, [1] = S1, [0] = S0
input PS2_CLK, PS2_DATA;
output [3:0] Seg_G0_En, Seg_G1_En;
output [7:0] Seg_G0, Seg_G1;
output reg [15:0] LED_PORT;

// ========== [Freq. Divider] ========== //
wire clk_LED, clk_Seg7, clk_Button;
wire isThrownBall;
CLK_DIV CLK_Div(clk_LED, clk_Seg7, clk_Button, clk_100MHz, Reset,
isThrownBall);

// ========== [BCD Converter] ========== //
reg [3:0] BCD_num;
wire [3:0] digit_Units;
wire digit_Tens;
BCD_Converter BCD_Converter(digit_Tens, digit_Units, BCD_num);

// ========== [Seven Segment] ========== //
reg [2:0] Seg_idx = 0;
reg [7:0] Seg_num;
reg [4*8-1:0] Name_Pokemon;
reg [1:0] Grade_Pokemon;
SevenSegment SevenSegment(Seg_G0_En, Seg_G1_En, Seg_G0, Seg_G1, Seg_idx,
Seg_num);

// ============== [LED] ============== //
reg [3:0] LED_idx = 0, LED_idx_next = 0;
reg isThrownBallDone, isThrownBallDone_next = 0;

// ============= [Button] ============= //
wire [3:0] Grade_Incense;
Button Button(isThrownBall, Grade_Incense, Button_PIN, clk_Button, Reset);

// ============= [Keyboard] ============= //
reg [1:0] Grade_Ball, Grade_Ball_next = Grade_Ball_Unknown;
wire [7:0] keyPressed;
Keyboard Keyboard(keyPressed, PS2_CLK, PS2_DATA, clk_100MHz, Reset);
```

```
// =========== [State Machine] =========== //
reg [1:0] State, State_next = State_Idle;

// ============= [Simulation] ============ //
wire [3:0] right_segment_value;
wire [7:0] left_segment;
wire [7:0] PS2_DATA_value;
wire [2:0] button;
wire [15:0] LED;
Simulation Simulation(right_segment_value, left_segment, button, LED,
PS2_DATA_value, Grade_Incense, Grade_Ball, Name_Pokemon, Button_PIN,
keyPressed, LED_PORT, clk_100MHz, Reset);

// ====================================== //
//   Current State Register (sequential)   //
// ====================================== //
// State
always @(posedge clk_100MHz or negedge Reset)
begin
    if(!Reset)
        State <= State_Idle;
    else
        State <= State_next;
end

// Keyboard
always @(posedge clk_100MHz or negedge Reset)
begin
    if(!Reset)
        Grade_Ball <= Grade_Ball_Unknown;
    else
        Grade_Ball <= Grade_Ball_next;
end

// ====================================== //
//     Next State Logic (combinational)    //
// ====================================== //
```

```
// State
always @(*)
begin
    case (State)
        State_Idle:
            State_next = isThrownBall ? State_Capture : State_Idle;
        State_Capture:
            State_next = isThrownBallDone ? State_Result : State_Capture;
        State_Result:
            State_next = State_Result;
        default :
            State_next = State_Idle;
    endcase
end

// Seven Segment
always @(*)
begin
    if(I0 <= Grade_Incense && Grade_Incense < I1)
    begin
        Name_Pokemon = Name_Pokemon_Caterpie;
        Grade_Pokemon = Grade_Pokemon_Caterpie;
    end
    else if(I1 <= Grade_Incense && Grade_Incense < I2)
    begin
        Name_Pokemon = Name_Pokemon_Rattata;
        Grade_Pokemon = Grade_Pokemon_Rattata;
    end
    else if(I2 <= Grade_Incense)
    begin
        Name_Pokemon = Name_Pokemon_Pikachu;
        Grade_Pokemon = Grade_Pokemon_Pikachu;
    end
    else
    begin
        Name_Pokemon = Name_Pokemon_Unknown;
        Grade_Pokemon = Grade_Pokemon_Unknown;
    end
```

```
    // [0-3] Pokemon Characters
    // [4-5] Ball Grade
    // [6-7] Incense Grade
    BCD_num = 0;
    case (Seg_idx)
        3'h0, 3'h1, 3'h2, 3'h3:
            Seg_num = Name_Pokemon[8 * (3 - Seg_idx) +: 8];
        3'h4:
            Seg_num = Unknown;
        3'h5:
        begin
            if(Grade_Ball == 3)
                Seg_num = Unknown;
            else
                Seg_num = Grade_Ball;
        end
        3'h6:
        begin
            BCD_num = Grade_Incense;
            Seg_num = digit_Tens;
        end
        3'h7:
        begin
            BCD_num = Grade_Incense;
            Seg_num = digit_Units;
        end
    endcase
end

// Keyboard
always @(*)
begin
    case (keyPressed)
        8'h52   : Grade_Ball_next = Grade_Ball_R; // R
        8'h42   : Grade_Ball_next = Grade_Ball_B; // B
        8'h4B   : Grade_Ball_next = Grade_Ball_K; // K
        default : Grade_Ball_next = Grade_Ball;
```

```
    endcase
end

// LED
always @(*)
begin
    case (State)
    State_Idle:
    begin
        LED_idx_next = 0;
        isThrownBallDone_next = 0;
        LED_PORT = 0;
    end
    State_Capture:
    begin
        if(!isThrownBallDone)
        begin
            LED_PORT = (1 << (15 - LED_idx));
            LED_idx_next = LED_idx + 1;
            isThrownBallDone_next = (LED_idx == 15);
        end
        else
        begin
            // BUGS : State Transition will have wait one cycle
            LED_PORT = 0;
            LED_idx_next = 0;
            isThrownBallDone_next = 0;
        end
    end
    State_Result:
    begin
        isThrownBallDone_next = 0;
        if(Grade_Ball >= Grade_Pokemon) // Capture Successfully
        begin
            LED_idx_next = (LED_idx == 1) ? 0 : 1;
            LED_PORT = (8'hFF << (8 * LED_idx));
        end
        else // Capture Unsuccessfully
```

```
        begin
            LED_idx_next = (LED_idx < 7) ? (LED_idx + 1) : (LED_idx - 7);
            LED_PORT = (1 << (7 - LED_idx)) | (1 << (8 + LED_idx));
        end
    end
    default:
    begin
        LED_idx_next = 0;
        isThrownBallDone_next = 0;
        LED_PORT = 0;
    end
    endcase
end


// ====================================== //
//       Output Logic (sequential)       //
// ====================================== //
// LED
always @(posedge clk_LED or negedge Reset)
begin
    if(!Reset)
    begin
        LED_idx <= 0;
        isThrownBallDone <= 0;
    end
    else
    begin
        LED_idx <= LED_idx_next;
        isThrownBallDone <= isThrownBallDone_next;
    end
end

// Seven Segment
always @(posedge clk_Seg7 or negedge Reset)
begin
    if(!Reset)
        Seg_idx <= 0;
```

```
    else
        Seg_idx <= Seg_idx + 1;
end
endmodule


module BCD_Converter(digit_Tens, digit_Units, BCD_num);
input [3:0] BCD_num;
output reg [3:0] digit_Units;
output reg digit_Tens;
reg Cout;

always @(BCD_num)
begin
    if(BCD_num > 9)
    begin
        digit_Units = BCD_num + 4'd6;
        digit_Tens = 1;
    end
    else
    begin
        digit_Units = BCD_num;
        digit_Tens = 0;
    end
end
endmodule

module SevenSegment(Seg_G0_En, Seg_G1_En, Seg_G0, Seg_G1, Seg_idx,
Seg_num);
input [2:0]Seg_idx;
input [7:0]Seg_num;
output reg [3:0]Seg_G0_En, Seg_G1_En;
output reg [7:0]Seg_G0, Seg_G1;
reg [7:0]SevenSeg;

always @(*)
begin
    case(Seg_num)
```

```
        8'h00 : SevenSeg = 8'b0011_1111;
        8'h01 : SevenSeg = 8'b0000_0110;
        8'h02 : SevenSeg = 8'b0101_1011;
        8'h03 : SevenSeg = 8'b0100_1111;
        8'h04 : SevenSeg = 8'b0110_0110;
        8'h05 : SevenSeg = 8'b0110_1101;
        8'h06 : SevenSeg = 8'b0111_1101;
        8'h07 : SevenSeg = 8'b0000_0111;
        8'h08 : SevenSeg = 8'b0111_1111;
        8'h09 : SevenSeg = 8'b0110_1111;
        8'h41 : SevenSeg = 8'b0111_0111; // A
        8'h43 : SevenSeg = 8'b0011_1001; // C
        8'h45 : SevenSeg = 8'b0111_1001; // E
        8'h49 : SevenSeg = 8'b0000_0110; // I
        8'h50 : SevenSeg = 8'b0111_0011; // P
        8'h52 : SevenSeg = 8'b0101_0000; // R
        8'h54 : SevenSeg = 8'b0111_1000; // T
        default : SevenSeg = 8'b0000_0000;
    endcase

    if(Seg_idx < 4)
    begin
        Seg_G0_En = (1 << (Seg_idx - 0));
        Seg_G1_En = 0;
        Seg_G0 = SevenSeg;
        Seg_G1 = 0;
    end
    else
    begin
        Seg_G0_En = 0;
        Seg_G1_En = (1 << (Seg_idx - 4));
        Seg_G0 = 0;
        Seg_G1 = SevenSeg;
    end
end
endmodule

module Button(isThrownBall, Grade_Incense, Button_PIN, clk_Button, Reset);
```

```
input clk_Button, Reset;
input [2:0] Button_PIN;
output reg isThrownBall;
output reg [3:0] Grade_Incense;

reg [2:0] Button;
reg isThrownBall_next = 0;
reg [3:0] Grade_Incense_next = 0;

always @(posedge clk_Button or negedge Reset)
begin
    if(!Reset)
    begin
        Button <= 0;
        isThrownBall <= 0;
        Grade_Incense <= 0;
    end
    else
    begin
        Button <= Button_PIN;
        isThrownBall <= isThrownBall_next;
        Grade_Incense <= Grade_Incense_next;
    end
end

always @(*)
begin
    isThrownBall_next = isThrownBall;
    Grade_Incense_next = Grade_Incense;

    // Current State : Released, Last State : Pressed
    if(!(|Button_PIN) && (|Button))
    begin
        isThrownBall_next = Button[0];

        if(0 < Grade_Incense && Button[1])
            Grade_Incense_next = Grade_Incense - 1;
        else if(Grade_Incense < 15 && Button[2])
```

```
            Grade_Incense_next = Grade_Incense + 1;
    end
end
endmodule

module Keyboard(keyPressed, PS2_CLK, PS2_DATA, clk_100MHz, Reset);

parameter   State_UART_Start  = 2'h0,
            State_UART_Data   = 2'h1,
            State_UART_Parity = 2'h2,
            State_UART_Stop   = 2'h3;

parameter   BreakCode = 8'hF0;

parameter   MakeCode_R = 8'h2D,
            MakeCode_B = 8'h32,
            MakeCode_K = 8'h42;

input PS2_CLK, PS2_DATA;
input clk_100MHz, Reset;
output reg [7:0] keyPressed;
reg [7:0] keyPressed_next = 0;

reg isUARTTransmitComplete, isUARTTransmitComplete_next = 0;
reg isKeyReleased, isKeyReleased_next = 0;
reg [2:0] State_UART, State_UART_next = State_UART_Start;
reg [2:0] UART_Data_num, UART_Data_num_next = 0;
reg [7:0] UART_Data, UART_Data_next = 0;

always @(posedge clk_100MHz or negedge Reset)
begin
    if(!Reset)
    begin
        keyPressed <= 0;
        isUARTTransmitComplete <= 0;
        isKeyReleased <= 0;
    end
    else
```

```
    begin
        keyPressed <= keyPressed_next;
        isUARTTransmitComplete <= isUARTTransmitComplete_next;
        isKeyReleased <= isKeyReleased_next;
    end
end

always @(posedge PS2_CLK or negedge Reset)
begin
    if(!Reset)
    begin
        State_UART <= State_UART_Start;
        UART_Data_num <= 0;
        UART_Data <= 0;
    end
    else
    begin
        State_UART <= State_UART_next;
        UART_Data_num <= UART_Data_num_next;
        UART_Data <= UART_Data_next;
    end
end

always @(negedge PS2_CLK or negedge Reset)
begin
    if(!Reset)
    begin
        State_UART_next = State_UART_Start;
        UART_Data_next = 0;
        UART_Data_num_next = 0;
    end
    else
    begin
        State_UART_next = State_UART;
        isUARTTransmitComplete_next = 0;
        case (State_UART)
            State_UART_Start:
            begin
```

```
                if(PS2_DATA == 0)
                    State_UART_next = State_UART_Data;

                UART_Data_next = 0;
                UART_Data_num_next = 0;
            end
            State_UART_Data:
            begin
                if(UART_Data_num == 7)
                    State_UART_next = State_UART_Parity;
                UART_Data_next = UART_Data | (PS2_DATA << UART_Data_num);
                UART_Data_num_next = UART_Data_num + 1;
            end
            State_UART_Parity:
            begin
                // Odd Parity
                if(PS2_DATA == ~^UART_Data)
                    State_UART_next = State_UART_Stop;
                else
                    State_UART_next = State_UART_Start;
            end
            State_UART_Stop:
            begin
                State_UART_next = State_UART_Start;
                if(PS2_DATA == 1)
                    isUARTTransmitComplete_next = 1;
            end
        endcase
    end
end

always @(*)
begin
    keyPressed_next = keyPressed;
    isKeyReleased_next = isUARTTransmitComplete ? (UART_Data ==
BreakCode) : isKeyReleased;

    // Current State : Released, Last State : Pressed
```

```
    if(isKeyReleased && !isKeyReleased_next)
    begin
        case (UART_Data)
            MakeCode_R: keyPressed_next = 8'h52; // R
            MakeCode_B: keyPressed_next = 8'h42; // B
            MakeCode_K: keyPressed_next = 8'h4B; // K
            default:    keyPressed_next = 8'h2D; // -
        endcase
    end
end
endmodule

module CLK_DIV(clk_LED, clk_Seg7, clk_Button, clk_100MHz, Reset,
isThrownBall);
input isThrownBall;
input clk_100MHz, Reset;
output clk_LED, clk_Seg7, clk_Button;

// LED            2Hz => 2^25
// Seven Segment  3KHz => 2^15
// Button        100Hz => 2^20

// Hardware Program
reg clk_2Hz, clk_2Hz_next = 0;
reg clk_100Hz, clk_100Hz_next = 0;
reg clk_3KHz, clk_3KHz_next = 0;

reg [25:0] counter_LED, counter_LED_next = 0;
reg [15:0] counter_Seg7, counter_Seg7_next = 0;
reg [20:0] counter_Button, counter_Button_next = 0;

reg isCounter_LEDFirstReset, isCounter_LEDFirstReset_next = 1;

always @(posedge clk_100MHz or negedge Reset)
begin
    if(!Reset)
    begin
        counter_LED <= 0;
```

```
        counter_Seg7 <= 0;
        counter_Button <= 0;
        clk_2Hz <= 0;
        clk_3KHz <= 0;
        clk_100Hz <= 0;
        isCounter_LEDFirstReset <= 1;
    end
    else
    begin
        counter_LED <= counter_LED_next;
        counter_Seg7 <= counter_Seg7_next;
        counter_Button <= counter_Button_next;
        clk_2Hz <= clk_2Hz_next;
        clk_3KHz <= clk_3KHz_next;
        clk_100Hz <= clk_100Hz_next;
        isCounter_LEDFirstReset <= isCounter_LEDFirstReset_next;
    end
end

always @(*)
begin
    counter_LED_next = counter_LED + 1;
    counter_Seg7_next = counter_Seg7 + 1;
    counter_Button_next = counter_Button + 1;
    clk_2Hz_next = clk_2Hz;
    clk_3KHz_next = clk_3KHz;
    clk_100Hz_next = clk_100Hz;
    isCounter_LEDFirstReset_next = isCounter_LEDFirstReset;

    if(isThrownBall && isCounter_LEDFirstReset)
    begin
        counter_LED_next = 0;
        isCounter_LEDFirstReset_next = 0;
    end

    if(counter_LED[25])
    begin
        counter_LED_next = 0;
```

```
        clk_2Hz_next = ~clk_2Hz;
    end

    if(counter_Seg7[15])
    begin
        counter_Seg7_next = 0;
        clk_3KHz_next = ~clk_3KHz;
    end

    if(counter_Button[20])
    begin
        counter_Button_next = 0;
        clk_100Hz_next = ~clk_100Hz;
    end
end

assign clk_LED = clk_2Hz;
assign clk_Seg7 = clk_3KHz;
assign clk_Button = clk_100Hz;

// Software Simulation
// assign clk_Seg7 = clk_100MHz;
// assign clk_Button = clk_100MHz;
// assign clk_LED = clk_100MHz;
endmodule

module Simulation(right_segment_value, left_segment, button, LED,
PS2_DATA_value, Grade_Incense, Grade_Ball, Name_Pokemon, Button_PIN,
keyPressed, LED_PORT, clk_100MHz, Reset);
input [3:0] Grade_Incense;
input [1:0] Grade_Ball;
input [4*8-1:0] Name_Pokemon;
input [2:0] Button_PIN;
input [7:0] keyPressed;
input [15:0] LED_PORT;
input clk_100MHz, Reset;

output [3:0] right_segment_value;
```

```
output reg [7:0] left_segment;
output reg [7:0] PS2_DATA_value;
output [2:0] button;
output [15:0] LED;

reg counter = 0;

parameter   MakeCode_R = 8'h2D,
           MakeCode_B = 8'h32,
           MakeCode_K = 8'h42;

always @(*)
begin
    case (keyPressed)
        8'h52 : PS2_DATA_value = MakeCode_R; // R
        8'h42 : PS2_DATA_value = MakeCode_B; // B
        8'h4B : PS2_DATA_value = MakeCode_K; // K
        default : PS2_DATA_value = 0;
    endcase
end

always @(*)
begin
    case(Name_Pokemon[8*4-1:8*3])
        8'h41 : left_segment = 8'b1110_1110; // A
        8'h43 : left_segment = 8'b1001_1100; // C
        8'h45 : left_segment = 8'b1001_1110; // E
        8'h49 : left_segment = 8'b0110_0000; // I
        8'h50 : left_segment = 8'b1100_1110; // P
        8'h52 : left_segment = 8'b0000_1010; // R
        8'h54 : left_segment = 8'b0001_1110; // T
        default : left_segment = 8'b0000_0000;
    endcase
end

always @(posedge clk_100MHz or negedge Reset)
begin
    if(!Reset)
```

```
        counter <= 0;
    else
        counter <= counter + 1;
end

assign LED = LED_PORT;
assign right_segment_value = counter ? Grade_Ball : Grade_Incense;
assign button = {Button_PIN[2], Button_PIN[0], Button_PIN[1]};
endmodule
```

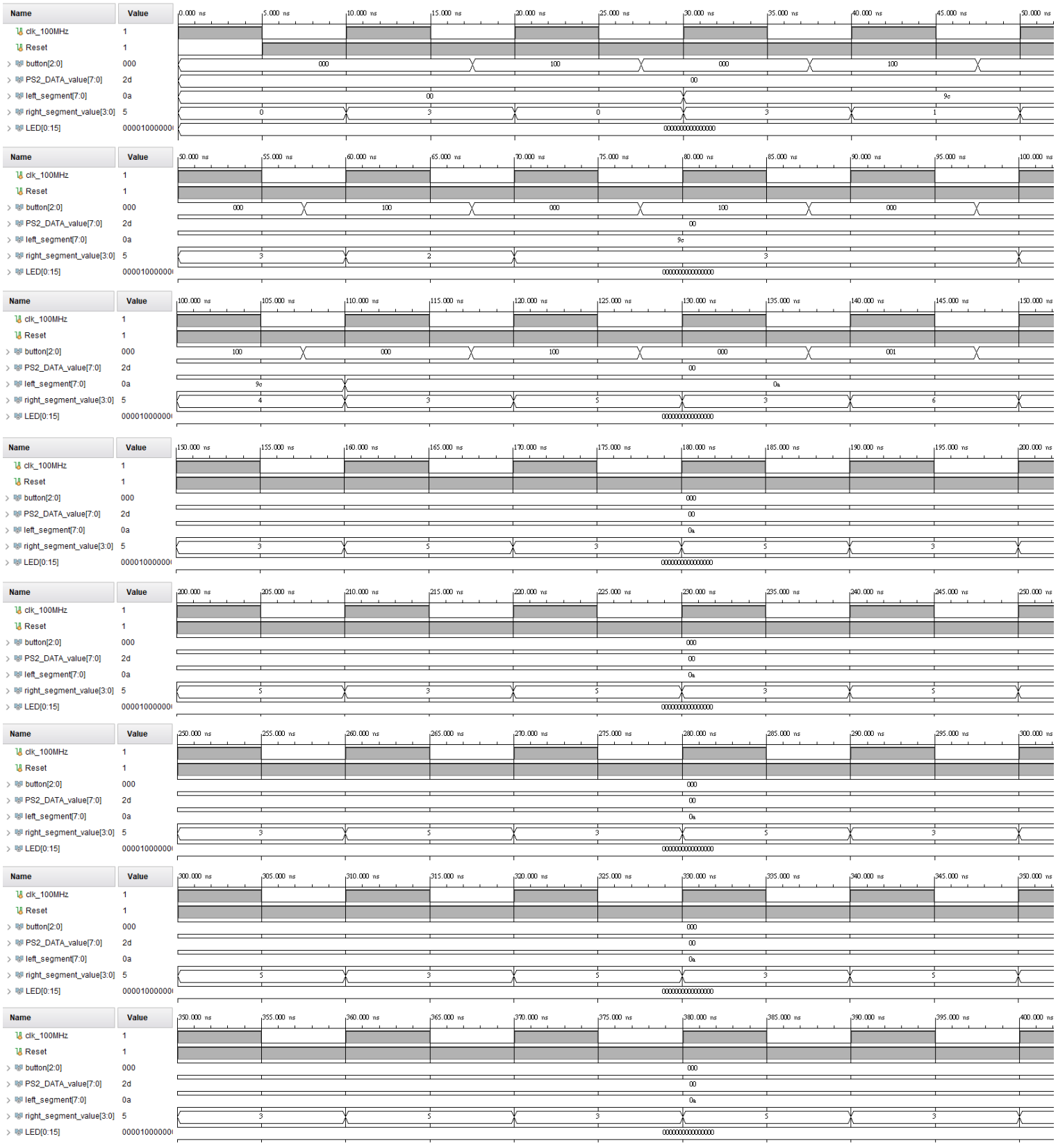
b. Test bench

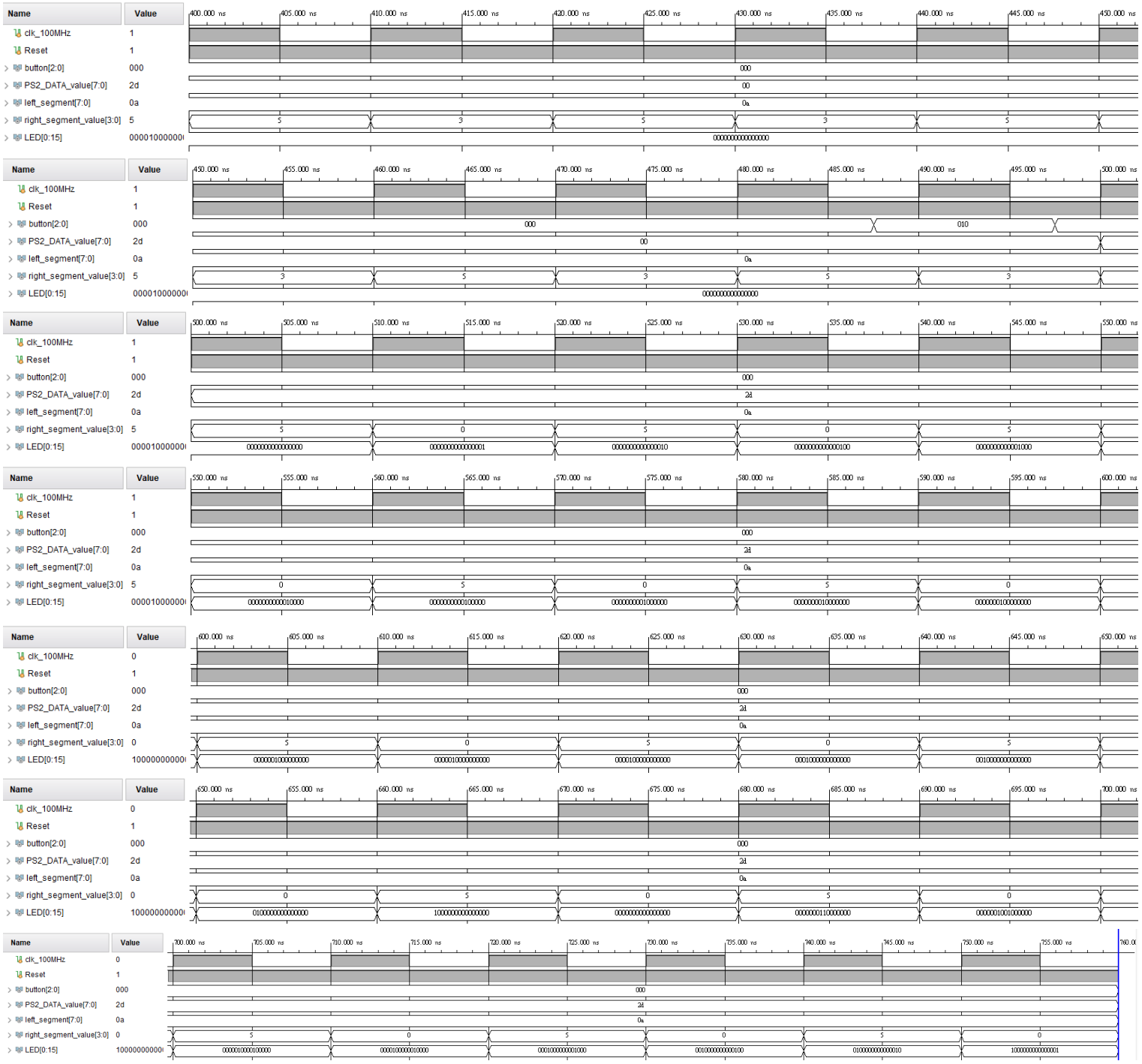
```
`timescale 1ns / 1ps

module Main_tb();
reg clk_100MHz, Reset;
reg PS2_CLK, PS2_DATA;
reg [2:0]Button_PIN;

wire [3:0]Seg_G0_En, Seg_G1_En;
wire [7:0]Seg_G0, Seg_G1;
wire [15:0]LED_PORT;

Main main(clk_100MHz, Reset, Button_PIN, PS2_CLK, PS2_DATA, Seg_G0_En,
Seg_G1_En, Seg_G0, Seg_G1, LED_PORT);

reg [7:0] MakeCode_R = 8'h2D;
reg [7:0] BreakCode = 8'hF0;
reg [3:0] i;

initial begin
    clk_100MHz = 1; Reset = 0;
    PS2_CLK = 1; PS2_DATA = 1;
    Button_PIN = 0;
    #5; Reset = 1;
    #2.5;

    // ================================== //
    //      Increase Incense Grade      //
    // ================================== //
    // Incense Grade = 0
    for (i = 0; i < 6; i = i + 1)
    begin
        #10; Button_PIN[2] = 1; // S4 = 1
        #10; Button_PIN[2] = 0; // S4 = 0
    end
    // Incense Grade = 6
    #10; Button_PIN[1] = 1; // S1 = 1
    #10; Button_PIN[1] = 0; // S1 = 0
```

```
// Incense Grade = 5

// ================================== //
//          Set Ball Grade          //
// ================================== //
// UART send 0x2D (Make Code of 'R')
// State_UART = State_UART_Start
#10; PS2_DATA = 0;

// State_UART = State_UART_Data
for (i = 0; i < 8; i = i + 1) begin
    #10; PS2_DATA = MakeCode_R[i]; // LSB
end

// State_UART = State_UART_Parity
#10; PS2_DATA = ~^MakeCode_R;

// State_UART = State_UART_Stop
#10; PS2_DATA = 1;
// ---------------------------------- //
// UART send 0xF0 (Break Code of normal key)
// State_UART = State_UART_Start
#10; PS2_DATA = 0;

// State_UART = State_UART_Data
for (i = 0; i < 8; i = i + 1) begin
    #10; PS2_DATA = BreakCode[i]; // LSB
end

// State_UART = State_UART_Parity
#10; PS2_DATA = ~^BreakCode;

// State_UART = State_UART_Stop
#10; PS2_DATA = 1;
// ---------------------------------- //
// UART send 0x2D (Make Code of 'R')
// State_UART = State_UART_Start
#10; PS2_DATA = 0;
```

```
    // State_UART = State_UART_Data
    for (i = 0; i < 8; i = i + 1) begin
        #10; PS2_DATA = MakeCode_R[i]; // LSB
    end

    // State_UART = State_UART_Parity
    #10; PS2_DATA = ~^MakeCode_R;

    // State_UART = State_UART_Stop
    #10; PS2_DATA = 1;
    // ---------------------------------- //
    // Throw Ball
    #10; Button_PIN[0] = 1; // S0 = 1
    #10; Button_PIN[0] = 0; // S0 = 0
    #12.5;
    // ================================== //
    //           LED Blinking           //
    // ================================== //
    #(10*16); // trajectory of the ball
    #10;
    #(10*8); // Pokemon is escaped
    $finish;
end

always begin
    #5; clk_100MHz = ~clk_100MHz;
end
always begin
    #5; PS2_CLK = ~PS2_CLK;
end
endmodule
```

c. Simulation results





Explanation

I0 = 4'd01, I1 = 4'd05, I2 = 4'd11;

0 [ns] => // Incense Grade=0

20 [ns] => S2=1 // Incense Grade=1, left_segment = 0b10011100 (‘C’), Seg_left = “CATE”

40 [ns] => S2=1 // Incense Grade=2

60 [ns] => S2=1 // Incense Grade=3
80 [ns] => S2=1 // Incense Grade=4
100 [ns] => S2=1 // Incense Grade=5, left_segment = 0b00001010 (‘R’), Seg_left = “RATT”
120 [ns] => S2=1 // Incense Grade=6
140 [ns] => S1=1 // Incense Grade=5
490 [ns] => S0=1 // Throw Ball
500 [ns] => PS2_DATA_value=MakeCode_R(0x2D)
510 ~ 670 [ns] => Trajectory of the ball
680 [ns] => LED blinks when Pokemon is escaped