Digital System Design and Implementation

Lab #5

姓名:黃鉦淳 學號:108303013

a. Verilog codes

```
`timescale 1ns / 1ps



module Main(clk_100MHz, Reset, Button_PIN, hsync, vsync, vga_r, vga_g, vga_b, Seg_G0_En, Seg_G1_En,
Seg_G0, Seg_G1, LED_PORT);

// ============================================================ //
//                        Input/Output                          //
// ============================================================ //
input clk_100MHz, Reset;
input [2:0] Button_PIN; // [2] = S4, [1] = S3, [0] = S0

output hsync, vsync;
output [3:0] vga_r, vga_g, vga_b;
output [3:0] Seg_G0_En, Seg_G1_En;
output [7:0] Seg_G0, Seg_G1;
output reg [15:0] LED_PORT;

// ============================================================ //
//                          Variable                            //
// ============================================================ //
wire clk_25MHz;
wire [9:0] h_cnt,v_cnt;
wire isInDisplay_region;

// ========== [VGA Color] ========== //
reg [11:0] vga_data;
parameter color_white = {4'hf, 4'hf, 4'hf};
parameter color_black = {4'h0, 4'h0, 4'h0};
parameter color_orange = {4'hf, 4'h8, 4'h0};
parameter color_blue = {4'h0, 4'h0, 4'hf};

parameter width_icon = 60;
parameter height_icon = 60;
```

```
parameter posX_max = 8;
parameter posY_max = 6;

parameter posX_animal_init = 3;
parameter posX_car_init = 0;
parameter posX_truck_init = 0;
parameter posX_longTruck_init = 7;

// =============== [ROM] =============== //
reg [11:0] rom_addr_brick;  // Depth = 2^12 > 60*60
wire [11:0] rom_data_brick; // Width = 2^12
reg [11:0] rom_addr_brick_all [4:0];
reg [2:0] posX_brick [4:0]; // square
parameter posY_brick = 5;   // square
parameter num_inRow_max_brick = 5;



reg [11:0] rom_addr_longTruck[2:0];  // Depth = 2^12 > 60*60
wire [11:0] rom_data_longTruck[2:0]; // Width = 2^12
reg [11:0] rom_addr_longTruck_all[1:0][2:0];
reg [3:0] posX_longTruck[1:0][2:0], posX_longTruck_next[1:0][2:0]; // square
parameter posY_longTruck = 4;                                      // square
parameter length_longTruck = 3;
parameter spacing_longTruck = 2;
parameter num_inRow_max_longTruck = 2;

reg [11:0] rom_addr_truck[1:0];  // Depth = 2^12 > 60*60
wire [11:0] rom_data_truck[1:0]; // Width = 2^12
reg [11:0] rom_addr_truck_all[2:0][1:0];
reg [3:0] posX_truck[2:0][1:0], posX_truck_next[2:0][1:0]; // square
parameter posY_truck = 2;                                  // square
parameter length_truck = 2;
parameter spacing_truck = 1;
parameter num_inRow_max_truck = 3;

reg [11:0] rom_addr_car;  // Depth = 2^12 > 60*60
wire [11:0] rom_data_car; // Width = 2^12
reg [11:0] rom_addr_car_all[2:0];
reg [3:0] posX_car[2:0], posX_car_next[2:0]; // square
parameter posY_car = 1;                      // square
parameter length_car = 1;
parameter spacing_car = 2;
```

```
parameter num_inRow_max_car = 3;

reg [11:0] rom_addr_animal;  // Depth = 2^12 > 60*60
wire [11:0] rom_data_animal; // Width = 2^12
reg [2:0] posX_animal, posX_animal_next; // square
reg [2:0] posY_animal, posY_animal_next; // square

parameter width_screen = 640;  // pixel
parameter height_screen = 480; // pixel

parameter width_square = 80;  // pixel
parameter height_square = 80; // pixel

parameter width_offset  = (width_square-width_icon)>>1;
parameter height_offset = (height_square-height_icon)>>1;

// ============== [Line] ============== //
parameter width_realLine = 5;
parameter width_dotLine = 3;
parameter spacing_dotLine = 10;

// ========== [Seven Segment] ========== //
reg [2:0] Seg_idx = 0;
reg [3:0] Seg_num;

// ============== [Button] ============== //
reg [2:0] Button;

// ============= [Freq Div] ============= //
wire clk_LED, clk_Seg7, clk_Button, clk_car, clk_truck;

// =========== [Seven Segment] ========== //
parameter Seg7_Unknown = 8'h0f;

// =============== [Animal] =============== //
reg [6:0] step_animal = 0, step_animal_next = 0;

reg isPreparingData_car = 0, isLockingData_car = 0;
reg isPreparingData_truck = 0, isLockingData_truck = 0;
reg isPreparingData_longTruck = 0, isLockingData_longTruck = 0;

// ============ [BCD Converter] ============ //
```

```
wire [3:0] digit_Tens;
wire [3:0] digit_Units;

// ================ [State] ================ //
reg state, state_next;
parameter   state_stop = 0,
            state_idle = 1;

// ================= [LED] ================= //
reg [2:0] idx_LED = 0;
reg isidx_LEDReset = 0;

// ============================================================ //
//                          Function                          //
// ============================================================ //
SyncGeneration u1 (
            .pclk(clk_25MHz),
            .reset(Reset),
            .hSync(hsync),
            .vSync(vsync),
            .dataValid(isInDisplay_region),
            .hDataCnt(h_cnt),
            .vDataCnt(v_cnt)
);

clk_25MHz u2 (.clk_in1(clk_100MHz), .clk_out1(clk_25MHz), .reset(!Reset));
icon_animal u3 (.clka(clk_25MHz), .addra(rom_addr_animal), .douta(rom_data_animal));
icon_brick u4 (.clka(clk_25MHz), .addra(rom_addr_brick), .douta(rom_data_brick));
icon_car u5 (.clka(clk_25MHz), .addra(rom_addr_car), .douta(rom_data_car));
icon_truck_1 u6 (.clka(clk_25MHz), .addra(rom_addr_truck[0]), .douta(rom_data_truck[0]));
icon_truck_2 u7 (.clka(clk_25MHz), .addra(rom_addr_truck[1]), .douta(rom_data_truck[1]));
icon_longTruck_1 u8 (.clka(clk_25MHz), .addra(rom_addr_longTruck[0]), .douta(rom_data_longTruck[0]));
icon_longTruck_2 u9 (.clka(clk_25MHz), .addra(rom_addr_longTruck[1]), .douta(rom_data_longTruck[1]));
icon_longTruck_3 u10 (.clka(clk_25MHz), .addra(rom_addr_longTruck[2]), .douta(rom_data_longTruck[2]));
SevenSegment u12 (Seg_G0_En, Seg_G1_En, Seg_G0, Seg_G1, Seg_idx, Seg_num);
CLK_DIV u13 (clk_100MHz, Reset, clk_LED, clk_Seg7, clk_Button, clk_car, clk_truck);
BCD_Converter u14 (digit_Tens, digit_Units, step_animal);
// ============================================================ //
//                   Conbinational Logic                    //
// ============================================================ //
assign {vga_r, vga_g, vga_b} = vga_data;
assign isInRealLine_region =(h_cnt<=width_realLine) |                                        // left
```

```
                        (((width_screen-width_realLine)<h_cnt) & (h_cnt<=width_screen)) |  // right
                        (v_cnt<=width_realLine) |                                         // top
                        (((height_screen-width_realLine)<v_cnt) & (v_cnt<=height_screen)); // bottom
assign isInDotLine_region = (((width_square - (width_dotLine>>1) <= h_cnt%width_square) |
(h_cnt%width_square<=(width_dotLine>>1))) & (v_cnt%spacing_dotLine==0)) |
                        (((height_square - (width_dotLine>>1) <= v_cnt%height_square) |
(v_cnt%height_square<=(width_dotLine>>1))) & (h_cnt%spacing_dotLine==0));
assign isInFloorOrange_region = (height_square*(posY_max-posY_car)>=v_cnt) &
(v_cnt>=height_square*(posY_max-1-posY_truck));
assign isInFloorBlue_region = (height_square*(posY_max-posY_longTruck)>=v_cnt) &
(v_cnt>=height_square*(posY_max-posY_brick));


reg [10:0] posX_icon_start, posX_icon_end;
reg [10:0] posY_icon_start, posY_icon_end;
reg isInAnimal_region;
always @(*)
begin
    posX_icon_start = width_square*posX_animal+width_offset;
    posX_icon_end = posX_icon_start+width_icon;
    posY_icon_start = height_square*(posY_max-1-posY_animal)+height_offset;
    posY_icon_end = posY_icon_start+height_icon;
    isInAnimal_region = (posX_icon_end>h_cnt) & (h_cnt>=posX_icon_start) & (posY_icon_end>v_cnt) &
(v_cnt>=posY_icon_start);
end

reg [2:0] i, j;
reg [2:0] idx_brick, idx_car, idx_truck, idx_longTruck;
reg [2:0] idx_img_truck, idx_img_longTruck;

// brick
always @(*)
begin
    idx_brick = 3'h7;
    posY_icon_start = height_square*(posY_max-1-posY_brick)+height_offset;
    posY_icon_end = posY_icon_start+height_icon;
    for (i = 0; i < num_inRow_max_brick; i = i + 1)
    begin
        posX_icon_start = width_square*posX_brick[i]+width_offset;
        posX_icon_end = posX_icon_start+width_icon;
        if ((posX_icon_end>h_cnt) & (h_cnt>=posX_icon_start) & (posY_icon_end>v_cnt) &
(v_cnt>=posY_icon_start))
```

```
            idx_brick = i;
    end
end

// car
always @(*)
begin
    idx_car = 3'h7;
    isPreparingData_car = 0;
    isLockingData_car = 0;
    for (i = 0; i < num_inRow_max_car; i = i + 1)
    begin
        posY_icon_start = height_square * (posY_max-1-posY_car)+height_offset;
        posY_icon_end = posY_icon_start + height_icon - 1;
        posX_icon_start = width_square * posX_car[i] + width_offset;
        posX_icon_end = posX_icon_start + width_icon - 1;
        if ((posX_icon_end+5>=h_cnt) & (h_cnt>=posX_icon_start-5) & (posY_icon_end>=v_cnt) &
(v_cnt>=posY_icon_start))
        begin
            idx_car = i;

            if((posX_icon_start>h_cnt) & (h_cnt>=posX_icon_start-5))
                isPreparingData_car = 1;

            if((posX_icon_end+5>=h_cnt) & (h_cnt>posX_icon_end))
                isLockingData_car = 1;
        end
    end
end

always @(*)
begin
    for (i = 0; i < num_inRow_max_car; i = i + 1)
    begin
        if(state==state_idle)
            posX_car_next[i] = posX_car[i] + 1;
        else
            posX_car_next[i] = posX_car[i];
    end

    for (i = 0; i < num_inRow_max_car; i = i + 1)
        if(posX_car[i] == (posX_car_init + spacing_car))
```

```
            posX_car_next[(i == 0)?i-1+num_inRow_max_car:i-1] = 0;
end

// truck
always @(*)
begin
    idx_truck = 3'h7;
    idx_img_truck = 3'h7;
    isPreparingData_truck = 0;
    isLockingData_truck = 0;
    for (i = 0; i < num_inRow_max_truck; i = i + 1)
    begin
        posY_icon_start = height_square*(posY_max-1-posY_truck)+height_offset;
        posY_icon_end = posY_icon_start+height_icon - 1;

        for (j = 0; j < length_truck; j = j + 1)
        begin
            posX_icon_start = width_square * posX_truck[i][j] + width_offset;
            posX_icon_end = posX_icon_start + width_icon - 1;

            if ((posX_icon_end+5>=h_cnt) & (h_cnt>=posX_icon_start-5) & (posY_icon_end>=v_cnt) &
(v_cnt>=posY_icon_start))
            begin
                idx_truck = i;
                idx_img_truck = j;

                if((posX_icon_start>h_cnt) & (h_cnt>=posX_icon_start-5))
                    isPreparingData_truck = 1;

                if((posX_icon_end+5>=h_cnt) & (h_cnt>posX_icon_end))
                    isLockingData_truck = 1;
            end
        end
    end
end

always @(*)
begin
    for (i = 0; i < num_inRow_max_truck; i = i + 1)
    begin
        for (j = 0; j < length_truck; j = j + 1)
        begin
```

```
            if(posX_truck[i][j] < posX_max & state==state_idle)
                posX_truck_next[i][j] = posX_truck[i][j] + 1;
            else
                posX_truck_next[i][j] = posX_truck[i][j];
        end
    end

    for (i = 0; i < num_inRow_max_truck; i = i + 1)
        for (j = 0; j < length_truck; j = j + 1)
            if(posX_truck_next[i][j] == (posX_truck_init + spacing_truck + length_truck))
                posX_truck_next[(i == 0)?num_inRow_max_truck-1:i-1][j] = posX_truck_init;
end



// longTruck
always @(*)
begin
    idx_longTruck = 3'h7;
    idx_img_longTruck = 3'h7;
    isPreparingData_longTruck = 0;
    isLockingData_longTruck = 0;

    for (i = 0; i < num_inRow_max_longTruck; i = i + 1)
    begin
        posY_icon_start = height_square*(posY_max-1-posY_longTruck)+height_offset;
        posY_icon_end = posY_icon_start+height_icon - 1;
        for (j = 0; j < length_longTruck; j = j + 1)
        begin
            posX_icon_start = width_square * posX_longTruck[i][j] + width_offset;
            posX_icon_end = posX_icon_start + width_icon - 1;
            if ((posX_icon_end+5>=h_cnt) & (h_cnt>=posX_icon_start-5) & (posY_icon_end>v_cnt) &
(v_cnt>=posY_icon_start))
            begin
                idx_longTruck = i;
                idx_img_longTruck = j;

                if((posX_icon_start>h_cnt) & (h_cnt>=posX_icon_start-5))
                    isPreparingData_longTruck = 1;

                if((posX_icon_end+5>=h_cnt) & (h_cnt>posX_icon_end))
                    isLockingData_longTruck = 1;
            end
```

```
        end
    end
end

always @(*)
begin
    for (i = 0; i < num_inRow_max_longTruck; i = i + 1)
    begin
        for (j = 0; j < length_longTruck; j = j + 1)
        begin
            if(posX_longTruck[i][j] < posX_max & state==state_idle)
                posX_longTruck_next[i][j] = posX_longTruck[i][j] - 1;
            else
                posX_longTruck_next[i][j] = posX_longTruck[i][j];
        end
    end

    for (i = 0; i < num_inRow_max_longTruck; i = i + 1)
        for (j = 0; j < length_longTruck; j = j + 1)
            if(posX_longTruck_next[i][j] == posX_longTruck_init - (spacing_longTruck + length_longTruck))
                posX_longTruck_next[(i == 0)?num_inRow_max_longTruck-1:i-1][j] = posX_longTruck_init;
end

// animal
always @(*)
begin
    posX_animal_next = posX_animal;
    posY_animal_next = posY_animal;
    step_animal_next = step_animal;

    // Current State : Released, Last State : Pressed
    if(!Button_PIN && Button && state==state_idle)
    begin
        step_animal_next = step_animal + 1;
        if(Button[2] & (posY_animal<posY_max - 1)) // S4 : Move Up
            posY_animal_next = posY_animal + 1;

        if(Button[1] & (posX_animal>0)) // S3 : Move Left
            posX_animal_next = posX_animal - 1;
        else if(Button[0] & (posX_animal<posX_max - 1)) // S0 : Move Right
            posX_animal_next = posX_animal + 1;
    end
```

```
end

// Seven Segment
reg isHitBrick = 0;
always @(*)
begin
    isHitBrick = 0;
    case (Seg_idx)
        3'h0, 3'h1:
        begin
            for (i = 0; i < num_inRow_max_brick; i = i + 1)
                if(posY_animal==posY_brick & posX_animal==posX_brick[i])
                    isHitBrick = 1;
            Seg_num = (posY_animal<posY_max-1 | isHitBrick) ? Seg7_Unknown : 4'h9;
        end
        3'h2:
            Seg_num = digit_Tens;
        3'h3:
            Seg_num = digit_Units;
        default:
            Seg_num = Seg7_Unknown;
    endcase
end

// State Machine
always @(*)
begin
    state_next = state;

    case(posY_animal)
        posY_car : begin
            for (i = 0; i < num_inRow_max_car; i = i + 1)
                if(posX_animal == posX_car[i])
                    state_next = state_stop;
        end
        posY_truck : begin
            for (i = 0; i < num_inRow_max_truck; i = i + 1)
                for (j = 0; j < length_truck; j = j + 1)
                    if(posX_animal == posX_truck[i][j])
                        state_next = state_stop;
        end
        posY_longTruck : begin
```

```
            for (i = 0; i < num_inRow_max_longTruck; i = i + 1)
                for (j = 0; j < length_longTruck; j = j + 1)
                    if(posX_animal == posX_longTruck[i][j])
                        state_next = state_stop;
        end
        posY_brick : begin
            for (i = 0; i < num_inRow_max_brick; i = i + 1)
                if(posX_animal == posX_brick[i])
                    state_next = state_stop;
        end
    endcase
end

// LED
always @(*)
begin
    if(state==state_stop)
        LED_PORT = 1'b1 << (7 - idx_LED) | 1'b1 << (8 + idx_LED);
    else
        LED_PORT = 0;
end

// ============================================================ //
//                    Sequential Logic                       //
// ============================================================ //
// VGA Display
always @(posedge clk_25MHz or negedge Reset)
begin
    if(!Reset)
        vga_data <= color_black;
    else
    begin
        if(isInDisplay_region)
        begin
            if(isInRealLine_region | isInDotLine_region)
                vga_data <= color_white;
            else if(isInFloorOrange_region)
                vga_data <= color_orange;
            else if(isInFloorBlue_region)
                vga_data <= color_blue;
            else
                vga_data <= color_black;
```

```
            if(idx_brick!=3'h7)
            begin
                rom_addr_brick_all[idx_brick] <= rom_addr_brick_all[idx_brick] + 12'd1;
                rom_addr_brick <= rom_addr_brick_all[idx_brick] + 12'd1;
                vga_data <= rom_data_brick;
            end

            if(idx_car!=3'h7)
            begin
                if(isPreparingData_car)
                    rom_addr_car <= rom_addr_car_all[idx_car];
                else if(isLockingData_car)
                    rom_addr_car_all[idx_car] <= rom_addr_car;
                else
                begin
                    rom_addr_car <= rom_addr_car + 12'd1;
                    vga_data <= rom_data_car;
                end
            end

            if(idx_truck!=3'h7)
            begin
                if(isPreparingData_truck)
                    rom_addr_truck[idx_img_truck] <= rom_addr_truck_all[idx_truck][idx_img_truck];
                else if(isLockingData_truck)
                    rom_addr_truck_all[idx_truck][idx_img_truck] <= rom_addr_truck[idx_img_truck];
                else
                begin
                    rom_addr_truck[idx_img_truck] <= rom_addr_truck[idx_img_truck] + 12'd1;
                    vga_data <= rom_data_truck[idx_img_truck];
                end
            end

            if(idx_longTruck!=3'h7 & idx_img_longTruck!=3'h7)
            begin
                if(isPreparingData_longTruck)
                    rom_addr_longTruck[idx_img_longTruck] <=
rom_addr_longTruck_all[idx_longTruck][idx_img_longTruck];
                else if(isLockingData_longTruck)
                    rom_addr_longTruck_all[idx_longTruck][idx_img_longTruck] <=
rom_addr_longTruck[idx_img_longTruck];
```

```
        else
        begin
            rom_addr_longTruck[idx_img_longTruck] <= rom_addr_longTruck[idx_img_longTruck] + 12'd1;
            vga_data <= rom_data_longTruck[idx_img_longTruck];
        end
    end

    if(isInAnimal_region)
    begin
        rom_addr_animal <= rom_addr_animal + 12'd1;
        vga_data <= rom_data_animal;
    end
end
else
begin
    vga_data <= color_black;

    if(v_cnt==0)
    begin
        rom_addr_animal <= 12'b0;
        rom_addr_brick <= 12'b0;
        rom_addr_car <= 12'b0;

        for (i = 0; i < num_inRow_max_brick; i = i + 1)
            rom_addr_brick_all[i] <= 12'b0;

        for (i = 0; i < num_inRow_max_car; i = i + 1)
            rom_addr_car_all[i] <= 12'b0;

        for (i = 0; i < num_inRow_max_truck; i = i + 1)
            for (j = 0; j < length_truck; j = j + 1)
                rom_addr_truck_all[i][j] <= 12'b0;

        for (i = 0; i < length_truck; i = i + 1)
            rom_addr_truck[i] <= 12'b0;

        for (i = 0; i < num_inRow_max_longTruck; i = i + 1)
            for (j = 0; j < length_longTruck; j = j + 1)
                rom_addr_longTruck_all[i][j] <= 12'b0;

        for (i = 0; i < length_longTruck; i = i + 1)
            rom_addr_longTruck[i] <= 12'b0;
```

```
            end
        end
    end
end

// car
always @(posedge clk_car or negedge Reset)
begin
    if(!Reset)
    begin
        for (i = 0; i < num_inRow_max_car; i = i + 1)
            posX_car[i] <= posX_car_init + (length_car + spacing_car) * i;
    end
    else
    begin
        for (i = 0; i < num_inRow_max_car; i = i + 1)
            posX_car[i] <= posX_car_next[i];
    end
end

// truck
always @(posedge clk_truck or negedge Reset)
begin
    if(!Reset)
    begin
        for (i = 0; i < num_inRow_max_truck; i = i + 1)
            for (j = 0; j < length_truck; j = j + 1)
                posX_truck[i][j] <= posX_truck_init + (length_truck + spacing_truck) * i + j;
    end
    else
    begin
        for (i = 0; i < num_inRow_max_truck; i = i + 1)
            for (j = 0; j < length_truck; j = j + 1)
                posX_truck[i][j] <= posX_truck_next[i][j];
    end
end

// longTruck
always @(posedge clk_truck or negedge Reset)
begin
    if(!Reset)
    begin
```

```
        for (i = 0; i < num_inRow_max_longTruck; i = i + 1)
            for (j = 0; j < length_longTruck; j = j + 1)
                posX_longTruck[i][j] <= posX_longTruck_init - (length_longTruck + spacing_longTruck) * i -
j;
    end
    else
    begin
        for (i = 0; i < num_inRow_max_longTruck; i = i + 1)
            for (j = 0; j < length_longTruck; j = j + 1)
                posX_longTruck[i][j] <= posX_longTruck_next[i][j];
    end
end

// brick
always @(posedge clk_100MHz or negedge Reset)
begin
    if(!Reset)
    begin
        posX_brick[0] = 3'd0;
        posX_brick[1] = 3'd2;
        posX_brick[2] = 3'd3;
        posX_brick[3] = 3'd5;
        posX_brick[4] = 3'd6;
    end
end

// animal
always @(posedge clk_Button or negedge Reset)
begin
    if(!Reset)
    begin
        posX_animal <= posX_animal_init;
        posY_animal <= 0;
        Button <= 0;
        step_animal <= 0;
    end
    else
    begin
        posX_animal <= posX_animal_next;
        posY_animal <= posY_animal_next;
        Button <= Button_PIN;
        step_animal <= step_animal_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

// State Machine
always @(posedge clk_100MHz or negedge Reset)
begin
    if(!Reset)
        state <= state_idle;
    else
        state <= state_next;
end

// LED
always @(posedge clk_LED or negedge Reset)
begin
    if(!Reset)
    begin
        isidx_LEDReset <= 0;
        idx_LED <= 0;
    end
    else
    begin
        if(state==state_stop)
        begin
            if(isidx_LEDReset)
                idx_LED <= idx_LED + 1;
            else
            begin
                idx_LED <= 0;
                isidx_LEDReset <= 1;
            end
        end
    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 [3:0]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)
        4'h0 : SevenSeg = 8'b0011_1111;
        4'h1 : SevenSeg = 8'b0000_0110;
        4'h2 : SevenSeg = 8'b0101_1011;
        4'h3 : SevenSeg = 8'b0100_1111;
        4'h4 : SevenSeg = 8'b0110_0110;
        4'h5 : SevenSeg = 8'b0110_1101;
        4'h6 : SevenSeg = 8'b0111_1101;
        4'h7 : SevenSeg = 8'b0000_0111;
        4'h8 : SevenSeg = 8'b0111_1111;
        4'h9 : SevenSeg = 8'b0110_1111;
        4'hf : SevenSeg = 8'b0000_0000;
        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 BCD_Converter(digit_Tens, digit_Units, BCD_num);
input [6:0] BCD_num;
output reg [3:0] digit_Tens;
output [3:0] digit_Units;

assign digit_Units = BCD_num % 10;

reg [6:0] i;
always @(*)
begin
    digit_Tens = 0;
    for (i = 0 ; i < 10; i = i + 1)
    begin
        if((10*i<=BCD_num) & (BCD_num<10*(i+1)))
            digit_Tens = i;
    end
end
endmodule

module CLK_DIV(clk_100MHz, Reset, clk_LED, clk_Seg7, clk_Button, clk_car, clk_truck);
input clk_100MHz, Reset;
output reg clk_LED, clk_Seg7, clk_Button, clk_car, clk_truck;

// LED            2Hz => 2^25
// Seven Segment  3KHz => 2^15
// Button       100Hz => 2^20
// Orange         1Hz => 2^27
// Blue           2Hz => 2^26

reg [25:0] counter_LED = 0;
reg [15:0] counter_Seg7 = 0;
reg [20:0] counter_Button = 0;
reg [26:0] counter_truck = 0;
reg [25:0] counter_car = 0;


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

```
        counter_LED <= 0;
        counter_Seg7 <= 0;
        counter_Button <= 0;
        counter_truck <= 0;
    end
    else
    begin
        counter_LED <= (counter_LED[25]) ? 0 : (counter_LED + 1);
        counter_Seg7 <= (counter_Seg7[15]) ? 0 : (counter_Seg7 + 1);
        counter_Button <= (counter_Button[20]) ? 0 : (counter_Button + 1);
        counter_truck <= (counter_truck[26] & counter_truck[25]) ? 0 : (counter_truck + 1);
        counter_car <= (counter_car[25] & counter_car[24]) ? 0 : (counter_car + 1);
    end
end

always @(posedge clk_100MHz or negedge Reset)
begin
    if(!Reset)
    begin
        clk_LED <= 0;
        clk_Seg7 <= 0;
        clk_Button <= 0;
        clk_car <= 0;
        clk_truck <= 0;
    end
    else
    begin
        if(counter_LED[25])
            clk_LED <= ~clk_LED;

        if(counter_Seg7[15])
            clk_Seg7 <= ~clk_Seg7;

        if(counter_Button[20])
            clk_Button <= ~clk_Button;

        if(counter_truck[26] & counter_truck[25])
            clk_truck <= ~clk_truck;

        if(counter_car[25] & counter_car[24])
            clk_car <= ~clk_car;
    end
```

```
end
endmodule

module SyncGeneration(pclk, reset, hSync, vSync, dataValid, hDataCnt, vDataCnt);
input  pclk;
input  reset;
output hSync;
output vSync;
output dataValid;
output [9:0] hDataCnt;
output [9:0] vDataCnt;
parameter H_SP_END = 96;
parameter H_BP_END = 144;
parameter H_FP_START = 785;
parameter H_TOTAL = 800;
parameter V_SP_END = 2;
parameter V_BP_END = 35;
parameter V_FP_START = 516;
parameter V_TOTAL= 525;
reg [9:0] x_cnt, y_cnt;
wire        h_valid, y_valid;

always @(posedge pclk or negedge reset) begin
    if (!reset)
        x_cnt <= 10'd1;
    else begin
        if (x_cnt == H_TOTAL) // horizontal
            x_cnt <= 10'd1; // retracing
        else
            x_cnt <= x_cnt + 1'b1;
    end
end

always @(posedge pclk or negedge reset) begin
    if (!reset)
        y_cnt <= 10'd1;
    else begin
        if (y_cnt == V_TOTAL & x_cnt == H_TOTAL)
            y_cnt <= 1; // vertical retracing
        else if (x_cnt == H_TOTAL)
            y_cnt <= y_cnt + 1;
        else
```

```
            y_cnt<=y_cnt;
    end
end

assign hSync = ((x_cnt > H_SP_END)) ? 1'b1 : 1'b0;
assign vSync = ((y_cnt > V_SP_END)) ? 1'b1 : 1'b0;
// Check P7 for horizontal timing
assign h_valid = ((x_cnt > H_BP_END) & (x_cnt < H_FP_START)) ? 1'b1 : 1'b0;
// Check P9 for vertical timing
assign v_valid = ((y_cnt > V_BP_END) & (y_cnt < V_FP_START)) ? 1'b1 : 1'b0;
assign dataValid = ((h_valid == 1'b1) & (v_valid == 1'b1)) ? 1'b1 : 1'b0;
// hDataCnt from 1 if h_valid==1
assign hDataCnt = ((h_valid == 1'b1)) ? x_cnt - H_BP_END : 10'b0;
// vDataCnt from 1 if v_valid==1
assign vDataCnt = ((v_valid == 1'b1)) ? y_cnt - V_BP_END : 10'b0;
endmodule
```