Digital System Design and Implementation

Final Project #D

姓名:黃鈺淳 學號:108303013

a. Verilog codes

```
`timescale 1ns / 1ps



module Main(clk_100MHz, Reset, PS2_CLK, PS2_DATA, Button_PIN, Switch_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 [4:0] Button_PIN; // [4] = S4, [3] = S3, [2] = S2, [1] = S1, [0] = S0
input [7:0] Switch_PIN;
input PS2_CLK, PS2_DATA;

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                           //
// ============================================================ //
// =============== [VGA] =============== //
wire clk_25MHz;
wire [9:0] h_cnt, v_cnt;
wire isInDisplay_region;
parameter width_screen = 640;  // pixel
parameter height_screen = 480; // pixel

parameter width_square = 80;   // pixel
parameter height_square = 80;  // pixel
parameter width_square_half = 40;   // pixel
parameter height_square_half = 40;  // pixel

parameter width_icon_05 = 40;     // pixel
```

```
parameter height_icon_05 = 40;    // pixel
parameter width_icon_1 = 80;     // pixel
parameter height_icon_1 = 80;    // pixel
parameter width_icon_2 = width_icon_1*2;   // pixel
parameter height_icon_2 = height_icon_1*2; // pixel

// ============ [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_green = {4'h8, 4'hc, 4'h8};
parameter color_deepGreen = {4'h3, 4'h7, 4'h3};
parameter color_brown = {4'hc, 4'ha, 4'h6};
parameter color_deepBrown = {4'h7, 4'h4, 4'h2};
parameter color_yellow = {4'hc, 4'hc, 4'h2};
parameter color_red = {4'hf, 4'h0, 4'h0};

// ============== [ROM] ============== //
reg [10:0] rom_addr_pea;  // Depth = 2^11 > 40*40
wire [11:0] rom_data_pea; // Width = 2^12

reg [12:0] rom_addr_peashooter;  // Depth = 2^13 > 75*75
wire [11:0] rom_data_peashooter; // Width = 2^12

reg [14:0] rom_addr_peashooter_large;  // Depth = 2^15 > 150*150
wire [11:0] rom_data_peashooter_large; // Width = 2^12

reg [12:0] rom_addr_wallnut;  // Depth = 2^13 > 75*75
wire [11:0] rom_data_wallnut; // Width = 2^12

reg [14:0] rom_addr_wallnut_large;  // Depth = 2^15 > 150*150
wire [11:0] rom_data_wallnut_large; // Width = 2^12

reg [12:0] rom_addr_sunshine;  // Depth = 2^13 > 75*75
wire [11:0] rom_data_sunshine; // Width = 2^12

reg [11:0] rom_addr_zombie;  // Depth = 2^13 > 75*75
wire [11:0] rom_data_zombie; // Width = 2^12

reg [11:0] rom_addr_zombie_buckethead;  // Depth = 2^13 > 75*75
wire [11:0] rom_data_zombie_buckethead; // Width = 2^12
```

```
reg [12:0] rom_addr_mower;  // Depth = 2^13 > 75*75
wire [11:0] rom_data_mower; // Width = 2^12

parameter posX_peashooter_large = 0;
parameter posX_wallnut_large = 2;
parameter posY_start_large = 0;

// ============ [Freq.DIV] ============= //
wire clk_3KHz, clk_100Hz, clk_2Hz, clk_1Hz, clk_05Hz, clk_025Hz;

// ========== [Seven Segment] ========== //
reg [2:0] Seg_idx = 0;
reg [3:0] Seg_num;
parameter Seg_Unknown = 4'hf;

// ============== [Switch] ============== //
wire start;
wire [1:0] level;
wire [2:0] sunshine_max;
wire [2:0] zombie_normal_max;
wire [1:0] zombie_buckethead_max;

// ============= [Keyboard] ============= //
wire [7:0] keyPressed;

// =============== [Button] =============== //
reg [4:0] Button;

// =============== [Button] =============== //
parameter width_realLine = 1;

// ============= [Select Box] ============= //
reg [2:0] posX_selectBox_button, posX_selectBox_button_next;
reg [2:0] posY_selectBox_button, posY_selectBox_button_next;
reg isSelect, isSelect_next;
parameter posX_selectBox_button_init = 3;
parameter posY_selectBox_button_init = 3;

reg [2:0] posX_selectBox_keyboard;
parameter posY_selectBox_keyboard = 0;
parameter posX_max = 8;
parameter posY_max = 6;
```

```
parameter width_SelectBox_button = 2;
parameter width_SelectBox_keyboard = 4;

// =============== [Other] =============== //
reg [10:0] posX_start_selectBox_button, posX_end_selectBox_button;
reg [10:0] posY_start_selectBox_button, posY_end_selectBox_button;

reg [10:0] posX_start_selectBox_keyboard, posX_end_selectBox_keyboard;
reg [10:0] posY_start_selectBox_keyboard, posY_end_selectBox_keyboard;

reg [10:0] posX_start_peashooter_large, posX_end_peashooter_large;
reg [10:0] posY_start_peashooter_large, posY_end_peashooter_large;

reg [10:0] posX_start_wallnut_large, posX_end_wallnut_large;
reg [10:0] posY_start_wallnut_large, posY_end_wallnut_large;

reg [10:0] posX_start_mower, posX_end_mower;
reg [10:0] posY_start_mower, posY_end_mower;

reg [10:0] posX_start_sunshine, posX_end_sunshine;
reg [10:0] posY_start_sunshine, posY_end_sunshine;

reg [10:0] posX_start_wallnut;
reg [10:0] posY_start_wallnut;

reg [10:0] posX_start_peashooter;
reg [10:0] posY_start_peashooter;

reg [10:0] posX_start_pea;
reg [10:0] posY_start_pea;

reg [10:0] posX_start_zombie_normal;
reg [10:0] posY_start_zombie_normal;

reg [10:0] posX_start_zombie_buckethead;
reg [10:0] posY_start_zombie_buckethead;

reg [10:0] posX_start_tmp, posX_end_tmp;
reg [10:0] posY_start_tmp, posY_end_tmp;

parameter posY_ROM_offset = 1;
```

```
parameter posY_map_start = 2;
parameter posY_map_end = posY_max - posY_map_start;

parameter mower_max = 3;
parameter posX_mower = 0;
parameter posY_mower = posY_map_start;

parameter posX_sunshine = 1;
parameter posY_sunshine = posY_map_start + 3;

// map_plants
// zone     - numerical meaning
// mower    - 1/0 : exsist or not
// sunshine - 1/0 : exsist or not
// zombie   - 0 : none, 1 : mower, 2 : sunshine, 3 : wallnut, 4 : peashooter
reg [2:0] map_plants [7:0][3:0];
reg map_bullets [13:0][2:0], map_bullets_next [13:0][2:0];
reg [1:0] map_zombies [15:0][2:0], map_zombies_next [15:0][2:0];
reg [1:0] map_zombies_life [15:0][2:0], map_zombies_life_next [15:0][2:0];
reg [4:0] x;
reg [1:0] y;
reg [1:0] z;
reg isSelected;
// ================= [] ================= //
reg [3:0] num_sunshine;
reg [3:0] zombie_normal_left;
reg [3:0] zombie_normal_created, zombie_normal_created_next;
reg [3:0] zombie_buckethead_left;
reg [3:0] zombie_buckethead_created, zombie_buckethead_created_next;



parameter zombie_none = 0;
parameter zombie_normal = 1;
parameter zombie_buckethead = 2;

parameter life_zombie_none = 0;
parameter life_zombie_normal = 1;
parameter life_zombie_buckethead = 2;

parameter plants_none = 0;
parameter plants_mower = 1;
parameter plants_sunshine = 2;
```

```
parameter plants_wallnut = 3;
parameter plants_peashooter = 4;

parameter sunshine_plants_peashooter = 1;
parameter sunshine_plants_wallnut = 2;
reg isMowerActive, isMowerActived;
// ================ [Random] ================ //
parameter seed_init = 3'd2;
reg [2:0] seed = seed_init;
reg [2:0] num_rand;
wire [2:0] num_rand_next;
reg [1:0] posY_rand_z1, posY_rand_z2;
reg Set = 1;
// ================ [LED] ================ //
reg [4:0] LED_idx;

// ================ [State] ================ //
reg [2:0] state;
parameter   state_stop = 0,
            state_stop_idle = 1,
            state_idle = 2,
            state_clear = 3,
            state_die = 4;
reg isState_stop_idle;
// ============================================================ //
//                         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));
CLK_DIV u3 (clk_100MHz, Reset, clk_3KHz, clk_100Hz, clk_2Hz, clk_1Hz, clk_05Hz, clk_025Hz);
Switch u4 (clk_100MHz, Reset, Switch_PIN, level, start, sunshine_max, zombie_normal_max,
zombie_buckethead_max);
SevenSegment u5 (Seg_G0_En, Seg_G1_En, Seg_G0, Seg_G1, Seg_idx, Seg_num);
```

```
Keyboard u6 (clk_100MHz, Reset, PS2_CLK, PS2_DATA, keyPressed);
pea u7 (.clka(clk_100MHz), .addra(rom_addr_pea), .douta(rom_data_pea));
peashooter u8 (.clka(clk_100MHz), .addra(rom_addr_peashooter), .douta(rom_data_peashooter));
peashooter_large u9
(.clka(clk_100MHz), .addra(rom_addr_peashooter_large), .douta(rom_data_peashooter_large));
wallnut u10 (.clka(clk_100MHz), .addra(rom_addr_wallnut), .douta(rom_data_wallnut));
wallnut_large u11 (.clka(clk_100MHz), .addra(rom_addr_wallnut_large), .douta(rom_data_wallnut_large));
sunshine u12 (.clka(clk_100MHz), .addra(rom_addr_sunshine), .douta(rom_data_sunshine));
zombie u13 (.clka(clk_100MHz), .addra(rom_addr_zombie), .douta(rom_data_zombie));
buckethead_zombie u14
(.clka(clk_100MHz), .addra(rom_addr_zombie_buckethead), .douta(rom_data_zombie_buckethead));
mower u15 (.clka(clk_100MHz), .addra(rom_addr_mower), .douta(rom_data_mower));
RandIntGenerator u16 (clk_2Hz, Reset, Set, seed, num_rand_next);

// ============================================================ //
//                   Conbinational Logic                    //
// ============================================================ //
// VGA Display
assign {vga_r, vga_g, vga_b} = vga_data;
assign isInRealLine_region =    (
                                !((width_realLine<v_cnt) & (v_cnt<height_square*posY_map_start) &
(width_square*4+width_realLine<h_cnt) & (h_cnt<=width_screen-width_realLine)) &
                                !((height_square*0+width_realLine<v_cnt) & (v_cnt<height_square*2) &
(width_square*0+width_realLine<h_cnt) & (h_cnt<=width_square*2-width_realLine)) &
                                !((height_square*0+width_realLine<v_cnt) & (v_cnt<height_square*2) &
(width_square*2+width_realLine<h_cnt) & (h_cnt<=width_square*4-width_realLine))
                            ) &
                            ((((height_square-width_realLine)<=(v_cnt-1)%height_square) | ((v_cnt-
1)%height_square<width_realLine)) |
                            (((width_square-width_realLine)<=(h_cnt-1)%width_square) | ((h_cnt-
1)%width_square<width_realLine)));
assign isInBrown_region =   (width_square*0 < h_cnt) & (h_cnt <= width_square*4) &
                          (height_square*0 < v_cnt) & (v_cnt <= height_square*2);
assign isInDeepGreen_region =   (width_square*0 < h_cnt) & (h_cnt <= width_square*1) &
                            (height_square*2 < v_cnt) & (v_cnt <= height_square*posY_max);
assign isInGreen_region =   (width_square*1 < h_cnt) & (h_cnt <= width_square*posX_max) &
                          (height_square*2 < v_cnt) & (v_cnt <= height_square*(posY_max-1));
assign isInYellow_region =  (width_square*1 < h_cnt) & (h_cnt <= width_square*posX_max) &
                          (height_square*5 < v_cnt) & (v_cnt <= height_square*posY_max);

reg isInSelectBox_button_region;
always @(*)
```

```
begin
    posX_start_selectBox_button = width_square*posX_selectBox_button;
    posX_end_selectBox_button = posX_start_selectBox_button + width_square;
    posY_start_selectBox_button = height_square*posY_selectBox_button;
    posY_end_selectBox_button = posY_start_selectBox_button + height_square;
    isInSelectBox_button_region =   ((posX_start_selectBox_button < h_cnt) & (h_cnt <=
posX_start_selectBox_button+width_SelectBox_button) & (posY_start_selectBox_button < v_cnt) & (v_cnt <=
posY_end_selectBox_button)) | // left
                                    ((posX_end_selectBox_button-width_SelectBox_button < h_cnt) & (h_cnt <=
posX_end_selectBox_button) & (posY_start_selectBox_button < v_cnt) & (v_cnt <=
posY_end_selectBox_button)) |     // right
                                    ((posX_start_selectBox_button < h_cnt) & (h_cnt <=
posX_end_selectBox_button) & (posY_start_selectBox_button < v_cnt) & (v_cnt <=
posY_start_selectBox_button+width_SelectBox_button)) | // top
                                    ((posX_start_selectBox_button < h_cnt) & (h_cnt <=
posX_end_selectBox_button) & (posY_end_selectBox_button-width_SelectBox_button < v_cnt) & (v_cnt <=
posY_end_selectBox_button));     // bottom
end

reg isInSelectBox_keyboard_region;
always @(*)
begin
    posX_start_selectBox_keyboard = width_square*posX_selectBox_keyboard;
    posX_end_selectBox_keyboard = posX_start_selectBox_keyboard + width_square*2;
    posY_start_selectBox_keyboard = height_square*posY_selectBox_keyboard;
    posY_end_selectBox_keyboard = posY_start_selectBox_keyboard + height_square*2;
    isInSelectBox_keyboard_region =    ((posX_start_selectBox_keyboard < h_cnt) & (h_cnt <=
posX_start_selectBox_keyboard+width_SelectBox_keyboard) & (posY_start_selectBox_keyboard < v_cnt) &
(v_cnt <= posY_end_selectBox_keyboard)) | // left
                                        ((posX_end_selectBox_keyboard-width_SelectBox_keyboard < h_cnt) &
(h_cnt <= posX_end_selectBox_keyboard) & (posY_start_selectBox_keyboard < v_cnt) & (v_cnt <=
posY_end_selectBox_keyboard)) |     // right
                                        ((posX_start_selectBox_keyboard < h_cnt) & (h_cnt <=
posX_end_selectBox_keyboard) & (posY_start_selectBox_keyboard < v_cnt) & (v_cnt <=
posY_start_selectBox_keyboard+width_SelectBox_keyboard)) | // top
                                        ((posX_start_selectBox_keyboard < h_cnt) & (h_cnt <=
posX_end_selectBox_keyboard) & (posY_end_selectBox_keyboard-width_SelectBox_keyboard < v_cnt) & (v_cnt <=
posY_end_selectBox_keyboard));     // bottom
end

reg isInPeashooter_large_region;
always @(*)
```

```
begin
    posX_start_peashooter_large = width_square*posX_peashooter_large;
    posX_end_peashooter_large = posX_start_peashooter_large + width_icon_2;
    posY_start_peashooter_large = height_square*posY_start_large;
    posY_end_peashooter_large = posY_start_peashooter_large + height_icon_2;
    isInPeashooter_large_region = (posX_start_peashooter_large<h_cnt) & (h_cnt<=posX_end_peashooter_large)
& (posY_start_peashooter_large<v_cnt) & (v_cnt<=posY_end_peashooter_large);
end

reg isInWallnut_large_region;
always @(*)
begin
    posX_start_wallnut_large = width_square*posX_wallnut_large;
    posX_end_wallnut_large = posX_start_wallnut_large + width_icon_2;
    posY_start_wallnut_large = height_square*posY_start_large;
    posY_end_wallnut_large = posY_start_wallnut_large + height_icon_2;
    isInWallnut_large_region = (posX_start_wallnut_large<h_cnt) & (h_cnt<=posX_end_wallnut_large) &
(posY_start_wallnut_large<v_cnt) & (v_cnt<=posY_end_wallnut_large);
end

reg isInMower_region;
always @(*)
begin
    isInMower_region = 0;
    posX_start_mower = width_square*posX_mower;
    posX_end_mower = posX_start_mower + width_icon_1;
    posY_start_mower = 0;
    posY_end_mower = 0;
    for (y = 0; y < mower_max; y = y + 1)
    begin
        posY_start_tmp = height_square*(posY_mower + y);
        posY_end_tmp = posY_start_tmp + height_icon_1;
        if(map_plants[0][y]==plants_mower & (posX_start_mower<h_cnt) & (h_cnt<=posX_end_mower) &
(posY_start_tmp<v_cnt) & (v_cnt<=posY_end_tmp))
        begin
            isInMower_region = 1;
            posY_start_mower = posY_start_tmp;
            posY_end_mower = posY_end_tmp;
        end
    end
end
```

```
reg isInSunshine_region;
always @(*)
begin
    isInSunshine_region = 0;
    posX_start_sunshine = 0;
    posX_end_sunshine = 0;
    posY_start_sunshine = height_square*posY_sunshine;
    posY_end_sunshine = posY_start_sunshine + height_icon_1;

    for (x = posX_sunshine; x <= sunshine_max; x = x + 1)
    begin
        posX_start_tmp = width_square*x;
        posX_end_tmp = posX_start_tmp + width_icon_1;
        if(map_plants[x][3]==plants_sunshine & (posX_start_tmp<h_cnt) & (h_cnt<=posX_end_tmp) &
(posY_start_sunshine<v_cnt) & (v_cnt<=posY_end_sunshine))
        begin
            isInSunshine_region = 1;
            posX_start_sunshine = posX_start_tmp;
            posX_end_sunshine = posX_end_tmp;
        end
    end
end

reg isInWallnut_region;
always @(*)
begin
    isInWallnut_region = 0;
    posX_start_wallnut = 0;
    posY_start_wallnut = 0;
    for (x = 0; x < posX_max; x = x + 1)
    begin
        posX_start_tmp = width_square*x;
        posX_end_tmp = posX_start_tmp + width_icon_1;
        for (y = 0; y < mower_max; y = y + 1)
        begin
            posY_start_tmp = height_square*(posY_map_start+y);
            posY_end_tmp = posY_start_tmp + height_icon_1;
            if(map_plants[x][y]==plants_wallnut & (posX_start_tmp<h_cnt) & (h_cnt<=posX_end_tmp) &
(posY_start_tmp<v_cnt) & (v_cnt<=posY_end_tmp))
            begin
                isInWallnut_region = 1;
                posX_start_wallnut = posX_start_tmp;
```

```
                posY_start_wallnut = posY_start_tmp;
            end
        end
    end
end

reg isInPeashooter_region;
always @(*)
begin
    isInPeashooter_region = 0;
    posX_start_peashooter = 0;
    posY_start_peashooter = 0;
    for (x = 0; x < posX_max; x = x + 1)
    begin
        posX_start_tmp = width_square*x;
        posX_end_tmp = posX_start_tmp + width_icon_1;
        for (y = 0; y < mower_max; y = y + 1)
        begin
            posY_start_tmp = height_square*(posY_map_start+y);
            posY_end_tmp = posY_start_tmp + height_icon_1;
            if(map_plants[x][y]==plants_peashooter & (posX_start_tmp<h_cnt) & (h_cnt<=posX_end_tmp) &
(posY_start_tmp<v_cnt) & (v_cnt<=posY_end_tmp))
            begin
                isInPeashooter_region = 1;
                posX_start_peashooter = posX_start_tmp;
                posY_start_peashooter = posY_start_tmp;
            end
        end
    end
end

reg isInPea_region;
always @(*)
begin
    isInPea_region = 0;
    posX_start_pea = 0;
    posY_start_pea = 0;
    for (x = 0; x < 2*(posX_max-posX_sunshine); x = x + 1)
    begin
        posX_start_tmp = width_square_half * x + width_square * posX_sunshine;
        posX_end_tmp = posX_start_tmp + width_icon_05;
        for (y = 0; y < mower_max; y = y + 1)
```

```
        begin
            posY_start_tmp = height_square*(posY_map_start+y);
            posY_end_tmp = posY_start_tmp + height_icon_05;
            if(map_bullets[x][y] & (posX_start_tmp<h_cnt) & (h_cnt<=posX_end_tmp) & (posY_start_tmp<v_cnt)
& (v_cnt<=posY_end_tmp))
            begin
                isInPea_region = 1;
                posX_start_pea = posX_start_tmp;
                posY_start_pea = posY_start_tmp;
            end
        end
    end
end
reg isInZombieNormal_region;
always @(*)
begin
    isInZombieNormal_region = 0;
    posX_start_zombie_normal = 0;
    posY_start_zombie_normal = 0;
    for (x = 0; x < 2*posX_max; x = x + 1)
    begin
        posX_start_tmp = width_square_half * x;
        posX_end_tmp = posX_start_tmp + width_icon_05;
        for (y = 0; y < mower_max; y = y + 1)
        begin
            posY_start_tmp = height_square*(posY_map_start+y);
            posY_end_tmp = posY_start_tmp + height_icon_1;
            if(map_zombies[x][y]==zombie_normal & (posX_start_tmp<h_cnt) & (h_cnt<=posX_end_tmp) &
(posY_start_tmp<v_cnt) & (v_cnt<=posY_end_tmp))
            begin
                isInZombieNormal_region = 1;
                posX_start_zombie_normal = posX_start_tmp;
                posY_start_zombie_normal = posY_start_tmp;
            end
        end
    end
end

reg isInZombieBuckethead_region;
always @(*)
begin
    isInZombieBuckethead_region = 0;
```

```
    posX_start_zombie_buckethead = 0;
    posY_start_zombie_buckethead = 0;
    for (x = 0; x < 2*posX_max; x = x + 1)
    begin
        posX_start_tmp = width_square_half * x;
        posX_end_tmp = posX_start_tmp + width_icon_05;
        for (y = 0; y < mower_max; y = y + 1)
        begin
            posY_start_tmp = height_square*(posY_map_start+y);
            posY_end_tmp = posY_start_tmp + height_icon_1;
            if(map_zombies[x][y]==zombie_buckethead & (posX_start_tmp<h_cnt) & (h_cnt<=posX_end_tmp) &
(posY_start_tmp<v_cnt) & (v_cnt<=posY_end_tmp))
            begin
                isInZombieBuckethead_region = 1;
                posX_start_zombie_buckethead = posX_start_tmp;
                posY_start_zombie_buckethead = posY_start_tmp;
            end
        end
    end
end

// pea movement
always @(*)
begin
    // copy from latch
    for (x = 0; x < 2*(posX_max-posX_sunshine); x = x + 1)
        for (y = 0; y < mower_max; y = y + 1)
            map_bullets_next[x][y] = map_bullets[x][y];

    // X = 1 has none pea
    for (y = 0; y < mower_max; y = y + 1)
        map_bullets_next[0][y] = 0;

    // offset all of pea from x to x+1
    for (x = 0; x < 2*(posX_max-posX_sunshine)-1; x = x + 1)
    begin
        for (y = 0; y < mower_max; y = y + 1)
        begin
            // 1. if bullet overlap with zombie
            // 2. position bullet+1==zombie, then next timing pea will intersect with zombie
            if(map_bullets[x][y] & (map_zombies[x+2*posX_sunshine][y]!=zombie_none |
map_zombies[x+2*posX_sunshine+1][y]!=zombie_none))
```

```
                map_bullets_next[x][y] = 0;
            else
                map_bullets_next[x+1][y] = map_bullets[x][y];
        end
    end

    // if peashooter does not shoot, then create a pea on the head of peashooter
    for (x = posX_sunshine; x < posX_max; x = x + 1)
        for (y = 0; y < mower_max; y = y + 1)
            if(map_plants[x][y]==plants_peashooter & clk_1Hz)
                map_bullets_next[2*(x-posX_sunshine)][y] = 1;
end

// zombie movement
reg posX_offset_minus;
reg [1:0] num_ZombieCanCreate;
always @(*)
begin
    // copy from latch
    for (x = 0; x < 2*posX_max; x = x + 1)
    begin
        for (y = 0; y < mower_max; y = y + 1)
        begin
            map_zombies_next[x][y] = map_zombies[x][y];
            map_zombies_life_next[x][y] = map_zombies_life[x][y];
        end
    end
    zombie_normal_created_next = zombie_normal_created;
    zombie_buckethead_created_next = zombie_buckethead_created;

    if(state==state_idle)
    begin
        // every 0.5 sec, offset all of zombie from x+1 to x
        for (x = posX_sunshine; x < posX_max ; x = x + 1)
        begin
            for (y = 0; y < mower_max; y = y + 1)
            begin
                for (z = 0; z < 2 ; z = z + 1)
                begin
                    // offset zombies
                    posX_offset_minus = (!(map_plants[x-1][y]==plants_wallnut & z==0) & map_zombies[2*x+z-
1][y]==zombie_none);
```

```
                map_zombies_next[2*x+z-posX_offset_minus][y] = map_zombies[2*x+z][y];
                map_zombies_life_next[2*x+z-posX_offset_minus][y] = map_zombies_life[2*x+z][y];

                if(posX_offset_minus)
                begin
                    map_zombies_next[2*x+z][y] = zombie_none;
                    map_zombies_life_next[2*x+z][y] = life_zombie_none;
                end

                // pea hit zombies
                if(x==posX_sunshine & z==0)
                begin
                    if(map_bullets[2*(x-posX_sunshine)+z][y] & map_zombies[2*x+z][y]!=zombie_none)
                        map_zombies_life_next[2*x+z-posX_offset_minus][y] = map_zombies_life[2*x+z][y] -
1;
                end
                else
                begin
                    if((map_bullets[2*(x-posX_sunshine)+z-1][y] | map_bullets[2*(x-
posX_sunshine)+z][y]) & map_zombies[2*x+z][y]!=zombie_none)
                        map_zombies_life_next[2*x+z-posX_offset_minus][y] = map_zombies_life[2*x+z][y] -
1;
                end

                if(map_zombies_life_next[2*x+z-posX_offset_minus][y]==life_zombie_none)
                    map_zombies_next[2*x+z-posX_offset_minus][y] = zombie_none;
            end
        end
    end

    num_ZombieCanCreate = 0;
    for (y = 0; y < mower_max; y = y + 1)
        if(map_zombies[2*posX_max-1][y]==zombie_none & map_plants[posX_max-1][y]!=plants_wallnut)
            num_ZombieCanCreate = num_ZombieCanCreate + 1;

    posY_rand_z1 = (num_rand[1:0]==0) ? 0 : (num_rand[1:0] - 1);
    posY_rand_z2 = (num_rand[1:0]==0) ? 1 : (posY_rand_z1 + num_rand[2] + 1)%3;

    // every 2 sec, create normal zombie
    if(clk_05Hz & !num_ZombieCanCreate==0 & zombie_normal_created<zombie_normal_max)
    begin
        if(map_zombies[2*posX_max-1][posY_rand_z1]!=zombie_none | map_plants[posX_max-
```

```
1][posY_rand_z1]==plants_wallnut)
            begin
                for (y = 0; y < mower_max; y = y + 1)
                    if(map_zombies[2*posX_max-1][y]==zombie_none & map_plants[posX_max-
1][y]!=plants_wallnut)
                        posY_rand_z1 = y;
            end
            zombie_normal_created_next = zombie_normal_created + 1;
            map_zombies_next[2*posX_max-1][posY_rand_z1] = zombie_normal;
            map_zombies_life_next[2*posX_max-1][posY_rand_z1] = life_zombie_normal;
        end

        // every 4 sec, create buckethead zombie
        if(clk_025Hz & num_ZombieCanCreate>1 & zombie_buckethead_created<zombie_buckethead_max)
        begin
            if(posY_rand_z1 == posY_rand_z2 | map_zombies[2*posX_max-1][posY_rand_z2]!=zombie_none |
map_plants[posX_max-1][posY_rand_z2]==plants_wallnut)
            begin
                for (y = 0; y < mower_max; y = y + 1)
                    if(y!=posY_rand_z1 & map_zombies[2*posX_max-1][y]==zombie_none & map_plants[posX_max-
1][y]!=plants_wallnut)
                        posY_rand_z2 = y;
            end
            zombie_buckethead_created_next = zombie_buckethead_created + 1;
            map_zombies_next[2*posX_max-1][posY_rand_z2] = zombie_buckethead;
            map_zombies_life_next[2*posX_max-1][posY_rand_z2] = life_zombie_buckethead;
        end

        // mower clean in a row of zombies
        if(isMowerActive)
        begin
            for (x = posX_sunshine; x < 2*posX_max ; x = x + 1)
            begin
                map_zombies_next[x][posY_selectBox_button-posY_map_start] = zombie_none;
                map_zombies_life_next[x][posY_selectBox_button-posY_map_start] = life_zombie_none;
            end
        end
    end
end

// zombie left
always @(*)
```

```
begin
    zombie_normal_left = 0;
    zombie_buckethead_left = 0;
    for (x = 0; x < 2*posX_max; x = x + 1)
    begin
        for (y = 0; y < mower_max; y = y + 1)
        begin
            if(map_zombies[x][y]==zombie_normal)
                zombie_normal_left = zombie_normal_left + 1;

            if(map_zombies[x][y]==zombie_buckethead)
                zombie_buckethead_left = zombie_buckethead_left + 1;
        end
    end
end

// button
always @(*)
begin
    posX_selectBox_button_next = posX_selectBox_button;
    posY_selectBox_button_next = posY_selectBox_button;
    isSelect_next = isSelected ? 0 : isSelect;
    // Current State : Released, Last State : Pressed
    if(!Button_PIN && Button && state!=state_stop)
    begin
        isSelect_next = Button[2];
        if(Button[4] & posY_selectBox_button>2) // S4 : Move Up
            posY_selectBox_button_next = posY_selectBox_button - 1;
        else if(Button[1] & posY_selectBox_button<(posY_max-1)) // S1 : Move Down
            posY_selectBox_button_next = posY_selectBox_button + 1;

        if(Button[3] & posX_selectBox_button>0) // S3 : Move Left
            posX_selectBox_button_next = posX_selectBox_button - 1;
        else if(Button[0] & (posX_selectBox_button<posX_max - 1)) // S0 : Move Right
            posX_selectBox_button_next = posX_selectBox_button + 1;
    end
end

// Seven Segment
always @(*)
begin
    case (Seg_idx)
```

```
        3'h0: Seg_num = level;
        3'h2: Seg_num = 0;
        3'h3: Seg_num = num_sunshine;
        3'h5: Seg_num = zombie_normal_left;
        3'h7: Seg_num = zombie_buckethead_left;
        default: Seg_num = Seg_Unknown;
    endcase
end

// keyboard
always @(*)
begin
    posX_selectBox_keyboard = 2;
    if(state!=state_stop)
    begin
        case(keyPressed)
            "Q" : if(level==2) posX_selectBox_keyboard = 0;
            "W" : posX_selectBox_keyboard = 2;
        endcase
    end
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
            // Background
            if(isInBrown_region)
                vga_data <= color_brown;
            else if(isInDeepGreen_region)
                vga_data <= color_deepGreen;
            else if(isInGreen_region)
                vga_data <= color_green;
            else if(isInYellow_region)
```

```
                vga_data <= color_yellow;
            else
                vga_data <= color_black;


            // Image
            if(isInPeashooter_large_region)
            begin
                rom_addr_peashooter_large <= width_icon_2*(v_cnt-posY_start_peashooter_large-
posY_ROM_offset) + (h_cnt-posX_start_peashooter_large);
                vga_data <= rom_data_peashooter_large;
            end

            if(isInWallnut_large_region)
            begin
                rom_addr_wallnut_large <= width_icon_2*(v_cnt-posY_start_wallnut_large-posY_ROM_offset) +
(h_cnt-posX_start_wallnut_large);
                vga_data <= rom_data_wallnut_large;
            end

            if(isInMower_region)
            begin
                rom_addr_mower <= width_icon_1*(v_cnt-posY_start_mower-posY_ROM_offset) + (h_cnt-
posX_start_mower);
                vga_data <= rom_data_mower;
            end

            if(isInSunshine_region)
            begin
                rom_addr_sunshine <= width_icon_1*(v_cnt-posY_start_sunshine-posY_ROM_offset) + (h_cnt-
posX_start_sunshine);
                vga_data <= rom_data_sunshine;
            end

            if(isInWallnut_region)
            begin
                rom_addr_wallnut <= width_icon_1*(v_cnt-posY_start_wallnut-posY_ROM_offset) + (h_cnt-
posX_start_wallnut);
                vga_data <= rom_data_wallnut;
            end

            if(isInPeashooter_region)
```

```
            begin
                rom_addr_peashooter <= width_icon_1*(v_cnt-posY_start_peashooter-posY_ROM_offset) + (h_cnt-
posX_start_peashooter);
                vga_data <= rom_data_peashooter;
            end

            if(isInPea_region)
            begin
                rom_addr_pea <= width_icon_05*(v_cnt-posY_start_pea-posY_ROM_offset) + (h_cnt-
posX_start_pea);
                vga_data <= rom_data_pea;
            end

            if(isInZombieNormal_region)
            begin
                rom_addr_zombie <= width_icon_05*(v_cnt-posY_start_zombie_normal-posY_ROM_offset) + (h_cnt-
posX_start_zombie_normal);
                vga_data <= rom_data_zombie;
            end

            if(isInZombieBuckethead_region)
            begin
                rom_addr_zombie_buckethead <= width_icon_05*(v_cnt-posY_start_zombie_buckethead-
posY_ROM_offset) + (h_cnt-posX_start_zombie_buckethead);
                vga_data <= rom_data_zombie_buckethead;
            end

            // Line
            if(isInSelectBox_button_region)
                vga_data <= color_red;
            else if(isInSelectBox_keyboard_region)
                vga_data <= color_deepBrown;
            else if(isInRealLine_region)
                vga_data <= color_white;
        end
        else
            vga_data <= color_black;
    end
end

// button
always @(posedge clk_3KHz or negedge Reset)
```

```
begin
    if(!Reset)
    begin
        posX_selectBox_button <= posX_selectBox_button_init;
        posY_selectBox_button <= posY_selectBox_button_init;
        isSelect <= 0;
        Button <= 0;
    end
    else
    begin
        posX_selectBox_button <= posX_selectBox_button_next;
        posY_selectBox_button <= posY_selectBox_button_next;
        isSelect <= isSelect_next;
        Button <= Button_PIN;
    end
end

// Seven Segment
always @(posedge clk_3KHz or negedge Reset)
begin
    if(!Reset)
        Seg_idx <= 0;
    else
        Seg_idx <= Seg_idx + 1;
end

// Map Plants
always @(posedge clk_100MHz or negedge Reset)
begin
    if(!Reset)
    begin
        for (x = 0; x < posX_max; x = x + 1)
            for (y = 0; y < mower_max; y = y + 1)
                map_plants[x][y] <= plants_none;

        for (y = 0; y < mower_max; y = y + 1)
            map_plants[0][y] <= plants_mower; // Mower initial

        for (x = posX_sunshine; x <= sunshine_max; x = x + 1)
            map_plants[x][3] <= plants_sunshine; // Sunshine initial

        num_sunshine <= 0;
```

```
        isSelected <= 0;
        isMowerActive <= 0;
    end
    else
    begin
        if(isSelect)
        begin
            isSelected <= 1;
            // mower
            if(posX_selectBox_button==posX_mower &
map_plants[posX_selectBox_button][posY_selectBox_button-posY_map_start] == plants_mower)
            begin
                map_plants[posX_selectBox_button][posY_selectBox_button-posY_map_start] <= plants_none;
                isMowerActive <= 1;
            end
            // sunshine
            else if(posY_selectBox_button==posY_sunshine &
map_plants[posX_selectBox_button][posY_selectBox_button-posY_map_start] == plants_sunshine)
            begin
                map_plants[posX_selectBox_button][posY_selectBox_button-posY_map_start] <= plants_none;
                num_sunshine <= num_sunshine + 1;
            end
            // zombie
            else if(posX_selectBox_button>posX_mower & posY_selectBox_button<posY_sunshine &
map_plants[posX_selectBox_button][posY_selectBox_button-posY_map_start] == plants_none)
            begin
                if(posX_selectBox_keyboard==2 & num_sunshine>=sunshine_plants_wallnut) // keyboard select
wallnut
                begin
                    num_sunshine <= num_sunshine - sunshine_plants_wallnut;
                    map_plants[posX_selectBox_button][posY_selectBox_button-posY_map_start] <=
plants_wallnut;
                end
                else if(num_sunshine>=sunshine_plants_peashooter)// keyboard select peashooter
                begin
                    num_sunshine <= num_sunshine - sunshine_plants_peashooter;
                    map_plants[posX_selectBox_button][posY_selectBox_button-posY_map_start] <=
plants_peashooter;
                end
            end
        end
        else
```

```
            isSelected <= 0;

        if(isMowerActived)
            isMowerActive <= 0;
    end
end

// pea
always @(posedge clk_2Hz or negedge Reset)
begin
    if(!Reset)
    begin
        for (x = 0; x < 2*(posX_max-posX_sunshine); x = x + 1)
            for (y = 0; y < mower_max; y = y + 1)
                map_bullets[x][y] <= 0;
    end
    else
    begin
        for (x = 0; x < 2*(posX_max-posX_sunshine); x = x + 1)
            for (y = 0; y < mower_max; y = y + 1)
                map_bullets[x][y] <= map_bullets_next[x][y];
    end
end

// zombie
always @(posedge clk_2Hz or negedge Reset)
begin
    if(!Reset)
    begin
        zombie_normal_created <= 0;
        zombie_buckethead_created <= 0;
        for (x = 0; x < 2*posX_max; x = x + 1)
        begin
            for (y = 0; y < mower_max; y = y + 1)
            begin
                map_zombies[x][y] <= zombie_none;
                map_zombies_life[x][y] <= life_zombie_none;
            end
        end
    end
    else
    begin
```

```
        isMowerActived <= isMowerActive;
        zombie_normal_created <= zombie_normal_created_next;
        zombie_buckethead_created <= zombie_buckethead_created_next;
        for (x = 0; x < 2*posX_max; x = x + 1)
        begin
            for (y = 0; y < mower_max; y = y + 1)
            begin
                map_zombies[x][y] <= map_zombies_next[x][y];
                map_zombies_life[x][y] <= map_zombies_life_next[x][y];
            end
        end
    end
end

// Random Integer Generator
always @(posedge clk_2Hz or negedge Reset)
begin
    if(!Reset)
    begin
        Set <= 1;
        seed <= num_rand==0?seed_init:num_rand;
    end
    else
    begin
        Set <= 0;
        num_rand <= num_rand_next;
    end
end

// State Machine
always @(posedge clk_100MHz or negedge Reset)
begin
    if(!Reset)
        state <= state_stop;
    else
    begin
        case(state)
            state_stop:
                if(start) state <= state_stop_idle;
            state_stop_idle:
                if(isState_stop_idle) state <= state_idle;
            state_idle: begin
```

```
                if( zombie_normal_created==zombie_normal_max &
zombie_buckethead_created==zombie_buckethead_max &
                    zombie_normal_left==0 & zombie_buckethead_left==0)
                        state <= state_clear;

                for (y = 0; y < mower_max; y = y + 1)
                begin
                    if(map_zombies[1][y]!=zombie_none)
                        state <= state_die;
                end
            end
        endcase
    end
end

// LED
reg isUpCounting;
always @(posedge clk_2Hz or negedge Reset)
begin
    if(!Reset)
    begin
        LED_idx <= 0;
        LED_PORT <= 0;
        isState_stop_idle <= 0;
        isUpCounting <= 1;
    end
    else
    begin
        case(state)
            state_stop_idle: begin
                LED_idx <= LED_idx + 1;
                for(x = 0; x < 16; x = x + 1)
                    LED_PORT[15-x] <= (x<LED_idx)?0:1;
                if(LED_idx==5'd16)
                begin
                    LED_idx <= 0;
                    isState_stop_idle <= 1;
                end
            end
            state_clear: begin
                isState_stop_idle <= 0;
                if(LED_idx==0) isUpCounting <= 1;
```

```
            if(LED_idx==3) isUpCounting <= 0;
            if(clk_1Hz)
            begin
                LED_idx <= isUpCounting?(LED_idx + 1):(LED_idx - 1);
                LED_PORT <= (1'b1 << LED_idx) | (1'b1 <<(15-LED_idx));
            end
        end
        state_die: begin
            if(clk_1Hz)
            begin
                LED_idx <= LED_idx + 1;
                LED_PORT <= LED_idx%2 ?16'hf0f0:16'h0f0f;
            end
        end
    endcase
  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 CLK_DIV(clk_100MHz, Reset, clk_3KHz, clk_100Hz, clk_2Hz, clk_1Hz, clk_05Hz, clk_025Hz);
input clk_100MHz, Reset;
output reg clk_3KHz, clk_100Hz, clk_2Hz, clk_1Hz, clk_05Hz, clk_025Hz;

// Zombie buckethead generate   0.25Hz => 2^29
// Zombie normal generate      0.50Hz => 2^28
// LED                        1Hz,2Hz => 2^25
// Zombie movement               2Hz => 2^25
// Pea                           2Hz => 2^25
// Button                      100Hz => 2^20
// Seven Segment                3KHz => 2^15

reg [27:0] counter_025Hz = 0;
reg [26:0] counter_05Hz = 0;
reg [25:0] counter_1Hz = 0;
reg [24:0] counter_2Hz = 0;
reg [18:0] counter_100Hz = 0;
reg [15:0] counter_3kHz = 0;


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

```
    counter_025Hz <= 0;
    counter_05Hz <= 0;
    counter_1Hz <= 0;
    counter_2Hz <= 0;
    counter_100Hz <= 0;
    counter_3kHz <= 0;

    clk_025Hz <= 0;
    clk_05Hz <= 0;
    clk_1Hz <= 0;
    clk_2Hz <= 0;
    clk_100Hz <= 0;
    clk_3KHz <= 0;
end
else
begin
    if(counter_025Hz==28'd200_000_000)
    begin
        counter_025Hz <= 0;
        clk_025Hz <= ~clk_025Hz;
    end
    else
        counter_025Hz <= counter_025Hz + 1;

    if(counter_05Hz==27'd100_000_000)
    begin
        counter_05Hz <= 0;
        clk_05Hz <= ~clk_05Hz;
    end
    else
        counter_05Hz <= counter_05Hz + 1;

    if(counter_1Hz==26'd50_000_000)
    begin
        counter_1Hz <= 0;
        clk_1Hz <= ~clk_1Hz;
    end
    else
        counter_1Hz <= counter_1Hz + 1;

    if(counter_2Hz==25'd25_000_000)
    begin
```

```
            counter_2Hz <= 0;
            clk_2Hz <= ~clk_2Hz;
        end
        else
            counter_2Hz <= counter_2Hz + 1;

        if(counter_100Hz==19'd500_000)
        begin
            counter_100Hz <= 0;
            clk_100Hz <= ~clk_100Hz;
        end
        else
            counter_100Hz <= counter_100Hz + 1;

        if(counter_3kHz==16'd16_667)
        begin
            counter_3kHz <= 0;
            clk_3KHz <= ~clk_3KHz;
        end
        else
            counter_3kHz <= counter_3kHz + 1;
    end
end
endmodule

module Switch(clk_100MHz, Reset, Switch_PIN, level, start, sunshine_max, zombie_normal_max,
zombie_buckethead_max);
input clk_100MHz, Reset;
input [7:0] Switch_PIN;

output reg start;
output reg [1:0] level;
output reg [2:0] sunshine_max;
output reg [2:0] zombie_normal_max;
output reg [1:0] zombie_buckethead_max;

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

```
        sunshine_max <= 0;
        zombie_normal_max <= 0;
        zombie_buckethead_max <= 0;
    end
    else
    begin
        start <= Switch_PIN[1];
        if(!Switch_PIN[1])
        begin
            level <= Switch_PIN[0]?2'd2:2'd1;
            sunshine_max <= Switch_PIN[2]?3'd7:3'd4;
            zombie_normal_max <= (Switch_PIN[5:3]>3&Switch_PIN[0]==0)?3:Switch_PIN[5:3];
            zombie_buckethead_max <= Switch_PIN[0]?Switch_PIN[7:6]:0;
        end
    end
end
endmodule

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

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_Q = 8'h15,
            MakeCode_W = 8'h1D;

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

reg isUARTTransmitComplete;
reg isKeyReleased, isKeyReleased_next;
reg [2:0] State_UART;
reg [2:0] UART_Data_num;
reg [7:0] UART_Data;


always @(negedge PS2_CLK or negedge Reset)
```

```
begin
    if(!Reset)
    begin
        State_UART <= State_UART_Start;
        UART_Data_num <= 0;
        UART_Data <= 0;
        isUARTTransmitComplete <= 0;
    end
    else
    begin
        case (State_UART)
            State_UART_Start:
            begin
                if(PS2_DATA == 0)
                    State_UART <= State_UART_Data;
                UART_Data <= 0;
                UART_Data_num <= 0;
                isUARTTransmitComplete <= 0;
            end
            State_UART_Data:
            begin
                if(UART_Data_num == 7)
                    State_UART <= State_UART_Parity;
                UART_Data[UART_Data_num] <= PS2_DATA;
                UART_Data_num <= UART_Data_num + 1;
            end
            State_UART_Parity:
            begin
                if(PS2_DATA == ~^UART_Data) // Odd Parity
                    State_UART <= State_UART_Stop;
                else
                    State_UART <= State_UART_Start;
            end
            State_UART_Stop:
            begin
                State_UART <= State_UART_Start;
                if(PS2_DATA == 1)
                    isUARTTransmitComplete <= 1;
            end
        endcase
    end
end
```

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

        // Current State : Break Code, Last State : Make Code
        if(isKeyReleased && !isKeyReleased_next)
        begin
            case (UART_Data)
                MakeCode_Q: keyPressed = "Q"; // Q
                MakeCode_W: keyPressed = "W"; // W
                default:    keyPressed = "-"; // -
            endcase
        end
    end
end

always @(*)
    isKeyReleased_next = isUARTTransmitComplete ? (UART_Data == BreakCode) : isKeyReleased;
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
```

```
module RandIntGenerator(clk, Reset, Set, Seed, num_3_bits);
input clk, Reset, Set;
input [2:0] Seed;
output [2:0] num_3_bits;

wire [2:0] mux_out;
wire [1:0] gate_xor_out;

// Instance
assign mux_out[2] = Set ? Seed[2] : num_3_bits[0];
DFF u1(.clk(clk), .Reset(Reset), .D(mux_out[2]), .Q(num_3_bits[2]));

assign gate_xor_out[1] = num_3_bits[2] ^ num_3_bits[0];
assign mux_out[1] = Set ? Seed[1] :gate_xor_out[1];
DFF u2(.clk(clk), .Reset(Reset), .D(mux_out[1]), .Q(num_3_bits[1]));

// assign gate_xor_out[0] = num_3_bits[1] ^ num_3_bits[0];
assign mux_out[0] = Set ? Seed[0] : num_3_bits[1];
DFF u3(.clk(clk), .Reset(Reset), .D(mux_out[0]), .Q(num_3_bits[0]));
endmodule

module DFF(clk, Reset, D, Q);
input clk, Reset;
input D;
output reg Q;
always @(posedge clk or negedge Reset)
    if(!Reset) Q <= 1'b0;
    else Q <= D;
endmodule
```