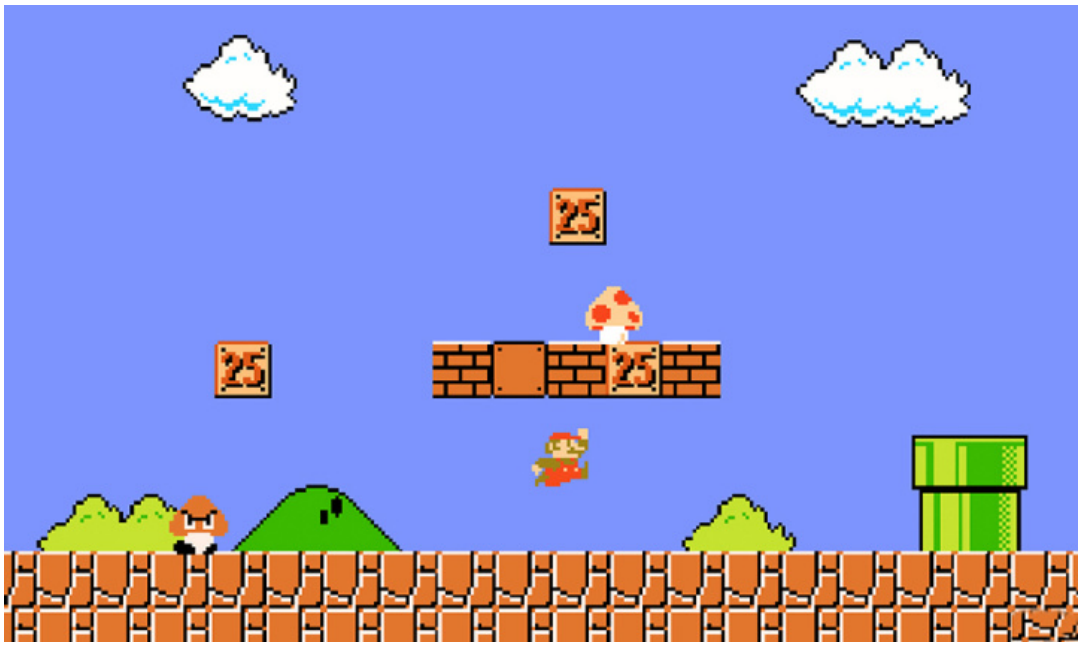
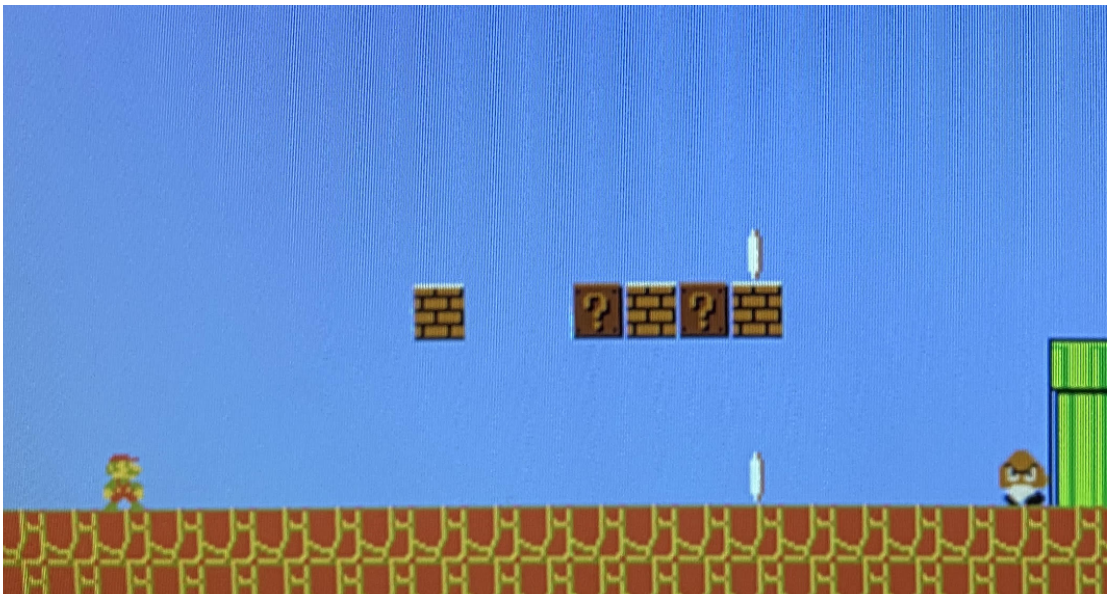
Super Mario Reconstruction
Final Report

Groupe members: Hangpu Cao (hc3346), Zeqi Li (zl3202), Shen Gao (sg4140), Han Yang (hy2759), Zhiyuan Liu (zl3208)



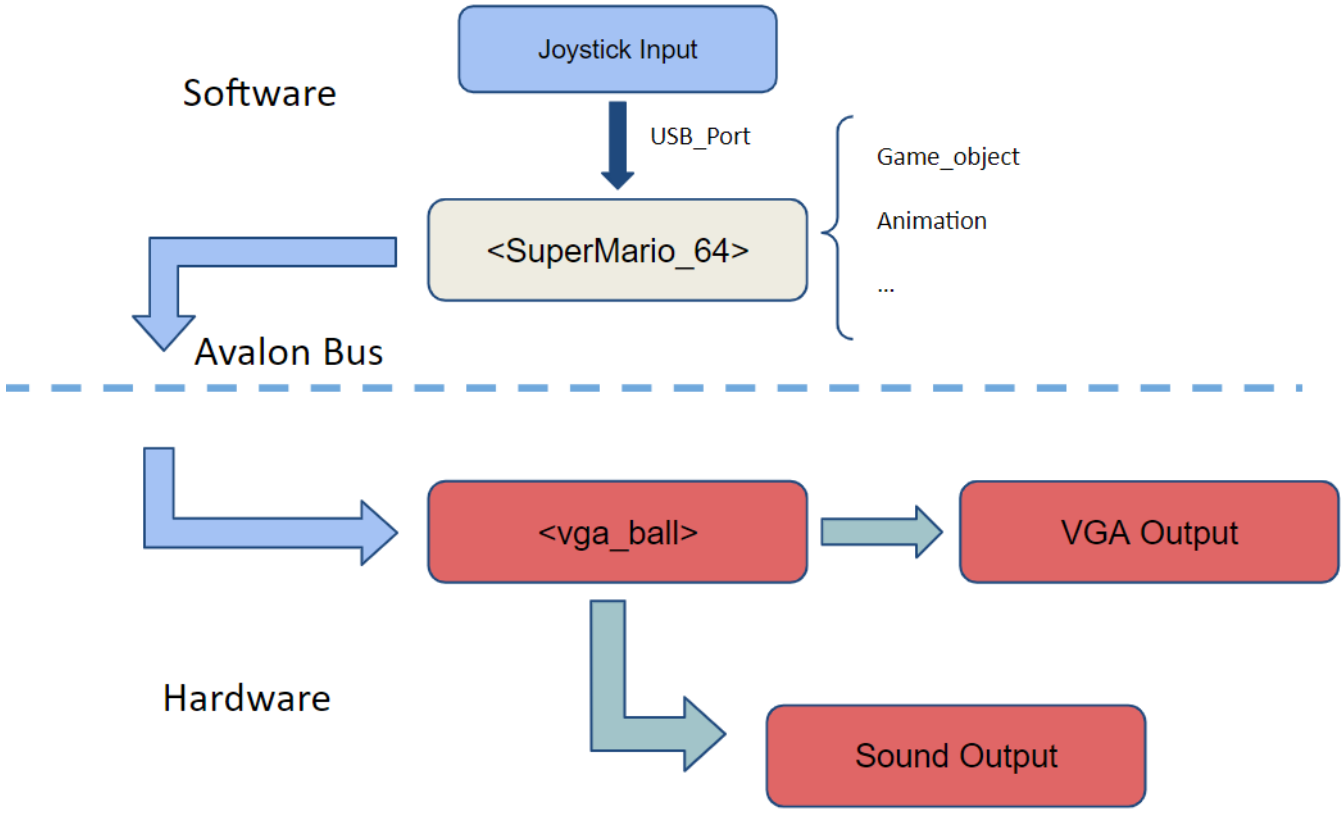


Well come to watch the demo video: https://youtu.be/4D5fFh1h7LI

Table of Content

Overview

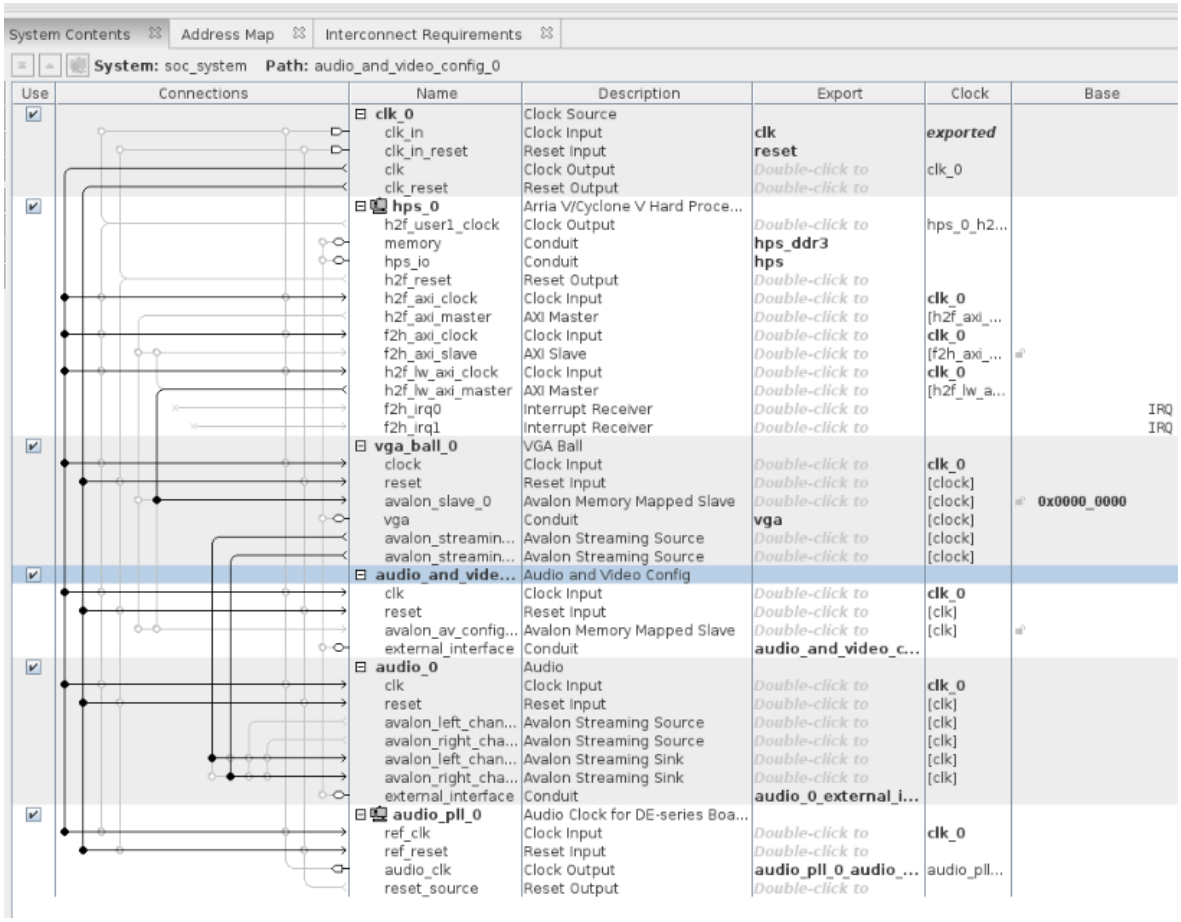
Our project goal is to construct a Mario-like 2D side-scroller game. We aim to attempt the reconstruction of core game features including 2D map scrolling, item acquisitions, and enemy interactions with the means of a collision detection algorithm.



Design Overview

- **Video output**
  - There are multiple sprites on the screen showing Mario, coins, goomba and the mush. The scene should be able to scroll horizontally and the character should stay in the center of the screen while scrolling.
- **Audio output**
  - The game should have BGM and audio effects (jump, enemy killed, coins)
- **Controller input**
  - The game has a USB Joystick as player input

- **Hitbox Detection**
    - The character should be blocked by the ground & walls. Enemies and coins should have the proper interaction when hitting the character.
- **Animation**
    - Add score animation, death animation, kill animation, and impact animation. Complete the character's interaction with terrain and enemies.



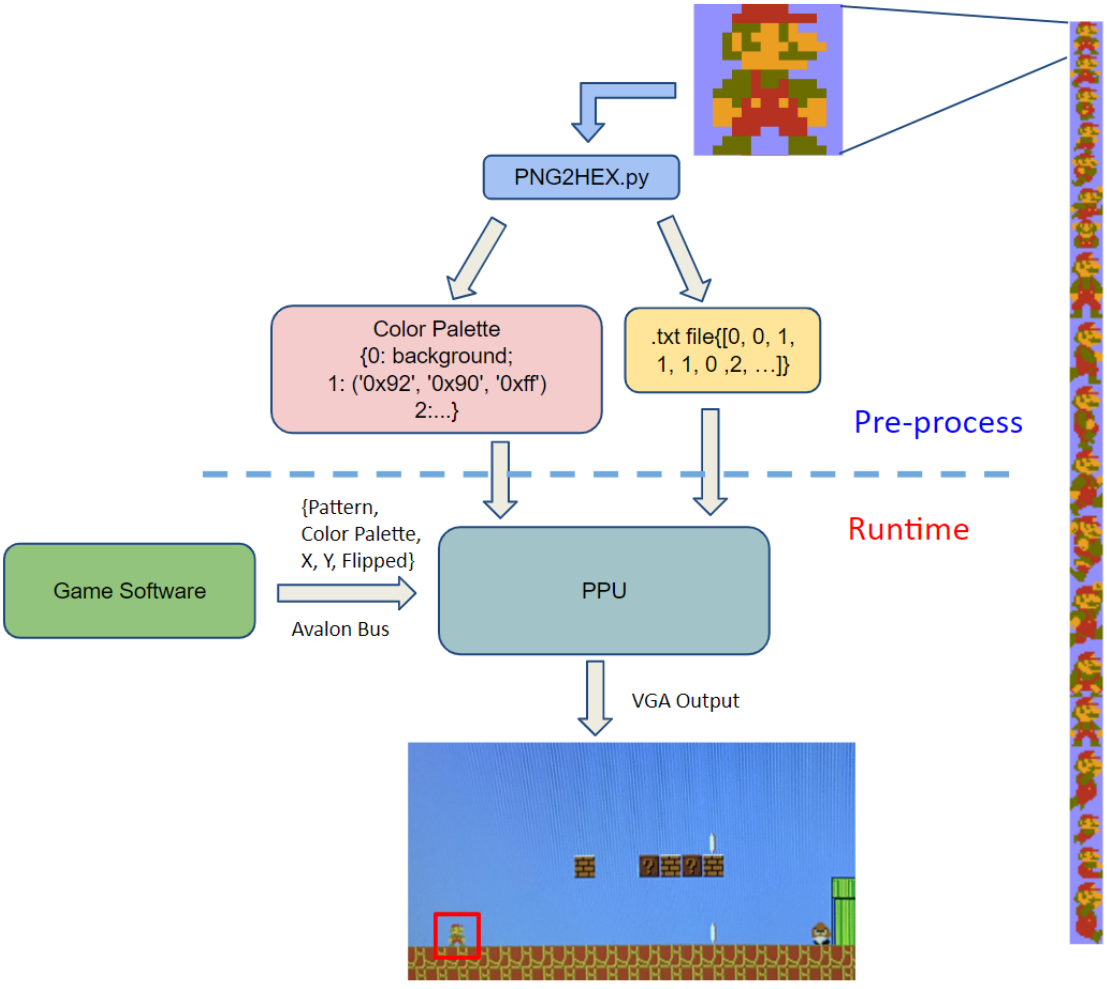
I/O Device

- Video Output: VGA
- Audio Output: 3.5mm audio jack
- Controller Input: USB Joystic

Video Output

The video output module is divided into two main tasks in our. The first part is to collect all the required image resources (Mario, bricks, goomba, coins, background elements, .etc). We rearrange those .PNG resources and encode them into sprite sheets (stored as .txt files) and a set of corresponding color palette.

The seconde part is implement a hardware PPU module which reads all the sprite sheets and color palettes, receives runtime data from the software through the avalon bus and print the image frames through the VGA output



- **Resource Image Pre-processing**
  - Convert from .PNG to .txt
    - Read RGBA value from .png files
    - Discover new RGBA color and save color palette

- Replace RGBA value with color palette index
- Write the byte stream into .txt files

- **Sprite Components**
  - Pattern
    - Which of the image is selected?
    - Color Palette
    - Image Size
  - Position
    - X Coordinate
    - Y Coordinate
    - Horizontal Flipped?

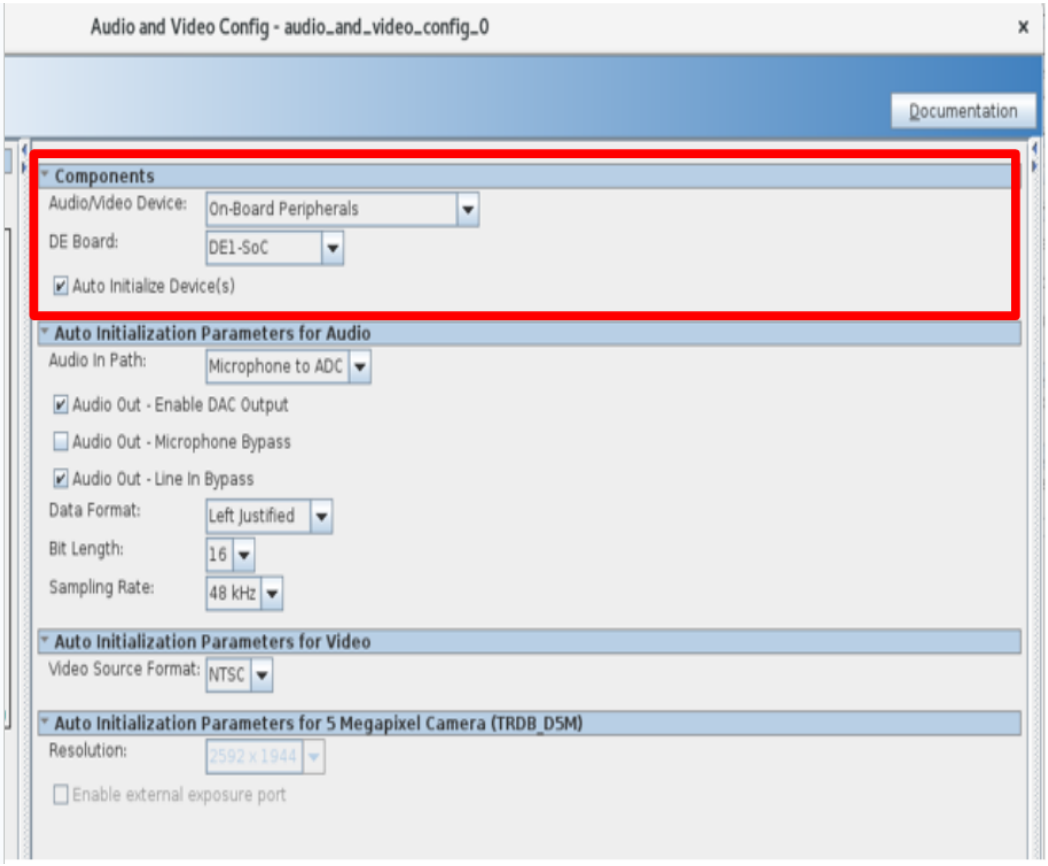
```
from PIL import Image

# Load the file & information
print("Enter File Name: ")
file_name = input()
im = Image.open(file_name)
pix = im.load()
print("Image Size:")
print(im.size)
x, y = im.size
print(im.mode)

# Setup the color palette & background color
pix_index_list = []
Color_Plate = [('0x92', '0x90', '0xff')]
byte_stream = []
byte_write = 0x00
for i in range(y):
        for j in range(x):
                R = hex(pix[j,i][0])
                G = hex(pix[j,i][1])
                B = hex(pix[j,i][2])
                RGB = (R, G, B)
                # If descover a new color
                # Save the color into the color palette
                if RGB not in Color_Plate:
                        Color_Plate.append(RGB)
                ind = Color_Plate.index(RGB)
                pix_index_list.append(ind)
                byte_write = ind
                byte_stream.append(byte_write)
print("Color Plate Info:")
for i in range(len(Color_Plate)):
        info = ""
        info += str(i) + ': ' + str(Color_Plate[i])
        print(info)
print("Coded File Length (Bytes): ")
print(len(byte_stream))
print("Enter Saved Filename: ")
# Save the file
saved_file_name = input()
s_f = open(saved_file_name, "w")
write_stream = '\n'.join('{:01x}'.format(a) for a in byte_stream)
s_f.write(write_stream)
s_f.close()
```

```
Enter File Name:
Cloud.png
Image Size:
(32, 24)
RGBA
Color Plate Info:
0: ('0x92', '0x90', '0xff')
1: ('0x0', '0x0', '0x0')
2: ('0xff', '0xff', '0xff')
3: ('0x63', '0xad', '0xff')
Coded File Length (Bytes):
768
Enter Saved Filename:
Cloud.txt
```

Audio Output

Components:

- **Audio/Video Device:** allows users to specify the target peripherals. Each option represents a distinct serial bus. The On-Board Peripherals option is for selecting the audio and video chips on the DE-series boards, while the other options are for selecting the various daughtercards.
- **DE Board:** allows users to specify the target board.
- **Auto Initialize Device(s):** allows users to add hardware to the core that will configure the target peripherals for the default operation required by other Intel FPGA University Program IP cores, which interact with those peripherals.

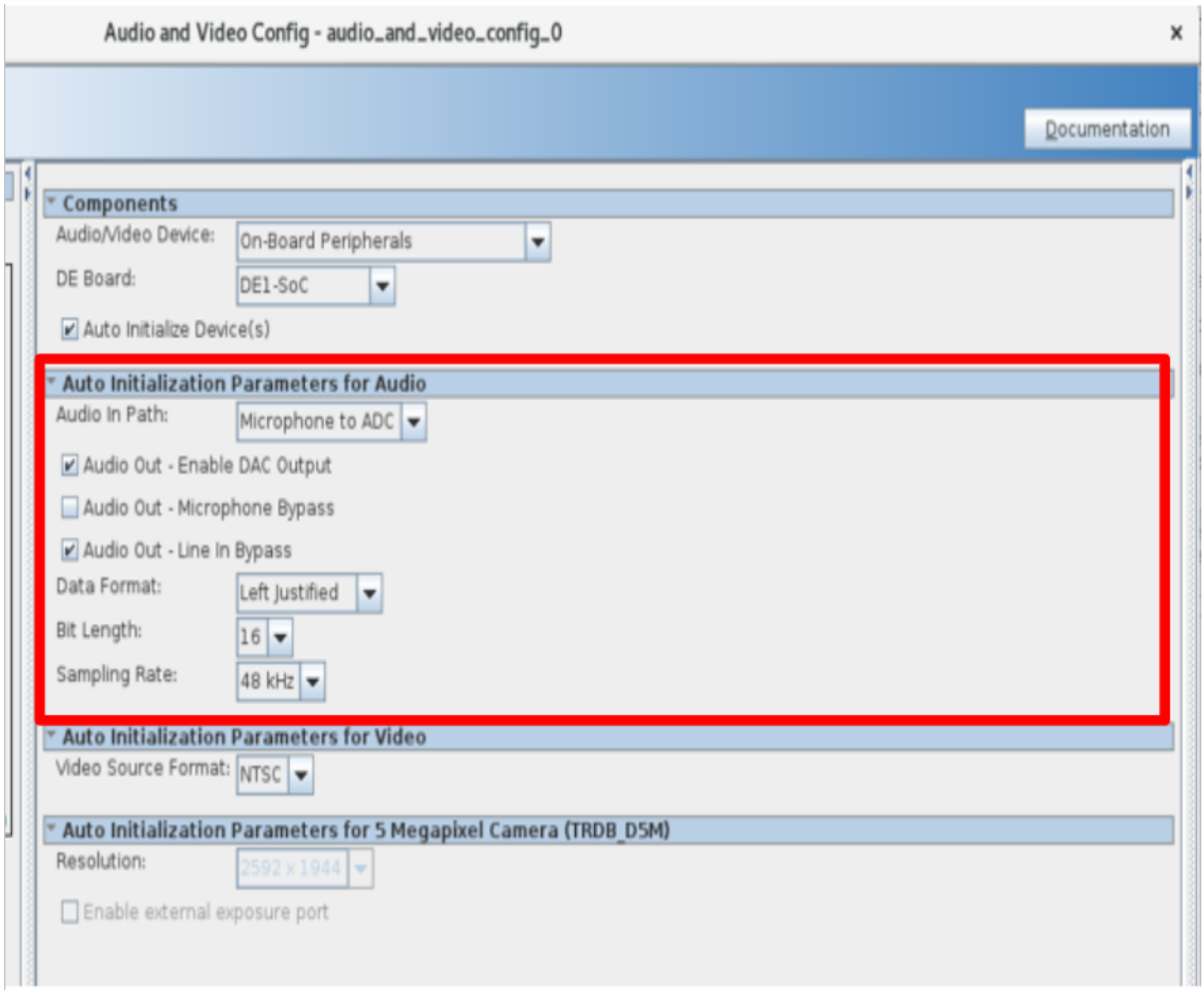


Auto Initialization Parameters for Audio: enabled when On-Board Peripherals and Auto Initialize Device(s) are both selected.

- **Audio In Path:** allows users to choose the microphone or the Line In as the input device for the ADC.
- **Audio Out - Enable DAC Output:** enables data from the DAC to pass to the Line Out.
- **Audio Out - Microphone Bypass:** If enabled the signal from the microphone

input jack bypasses the ADC and DAC, and goes directly to the output jack.

- **Audio Out - Line In Bypass:** If enabled the signal from the Line In input jack bypasses the ADC and DAC, and goes directly to the output jack.
- **Data Format:** allows users to choose the data format as DSP Mode, I2S Format, Left Justified or Right Justified. If using Intel's UP Audio core, then the Left Justified mode must be selected.
- **Bit Length:** allows the user to specify the number of bits of each audio sample. Valid values include 16, 20 and 24 bits for all modes, and 32 bits for I2S and Left Justified modes only. Both ADC and DAC operate on 24-bit data, so zero-padding or stripping of the least-significant bits happens when the bit length is not 24.
- **Sampling Rate:** allows the user to select the number of audio sample per second. Based on the selected rate, the audio chip's Base-Over Sampling Rate and Sample Rate control registers will be set appropriately, as per Wolfson* WM8731 Audio CODEC datasheet.



Controller Input

Our project use a USB input connected to the software program. The controller we use has a SNES-style setup and we can access the interface using the Linux *lsusb* C library and initialized with *openkeyboard()*

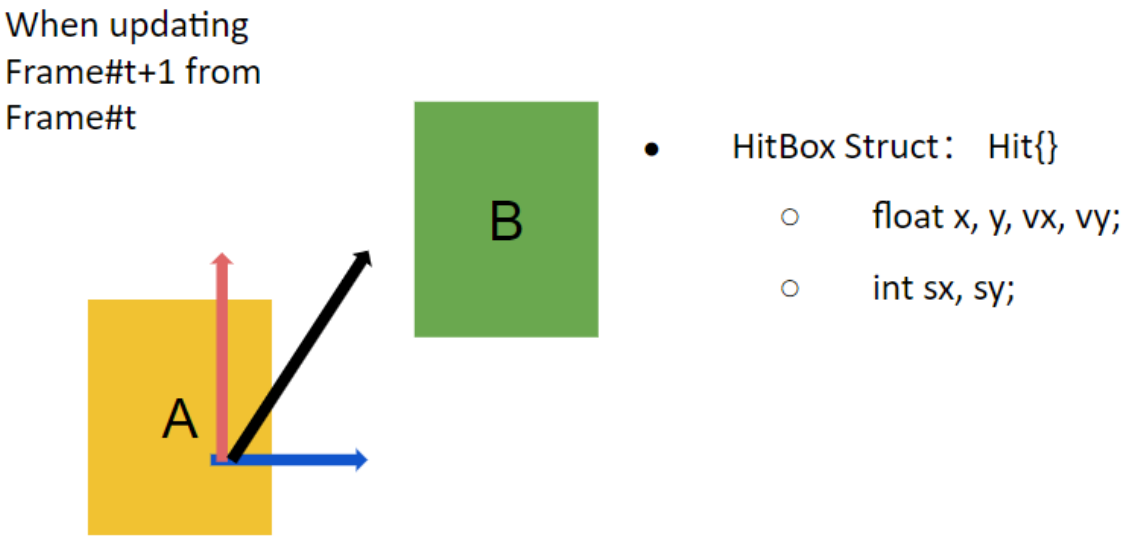
- Use *lsusb* command to find the following:
    - Vendor ID (VID) - 0x0079
    - Product ID (PID) - 0x0011
    - Endpoint Addr - 0x81
    - Packet Size - 8 bytes

```
mouse = libusb_open_device_with_vid_pid(ctx, 0x0079, 0x0011);
if (mouse == NULL)
{
    printf("%s\n", "Cannot open device");
    libusb_free_device_list(devs, 1); // free the list, unref the devices in it
    libusb_exit(ctx);                 // close the session
    return 0;
}
```

```
    size = 8;
    libusb_interrupt_transfer(mouse, 0x81, buff, 0x0008, &size, 0);
    if (size == 0x0008){
```

Hitbox Detection

Our game implements a hitbox detection algorithm comparing hiboxes of different types of game objects (Mario, bricks, goomba, coin, …) . The algorithm is written in C and would be called for multiple times to update object's states in every game frame.



Assumptions on this hitboxes A&B problem

- In Frame#t: A & B have no contact with each other
- Velocity vx & vy are bounded
- A & B are rectangles with edge = {sx_A, sy_A} {sx_B, sy_B}

From the assumptions, there are only 5 possible outputs from the function:

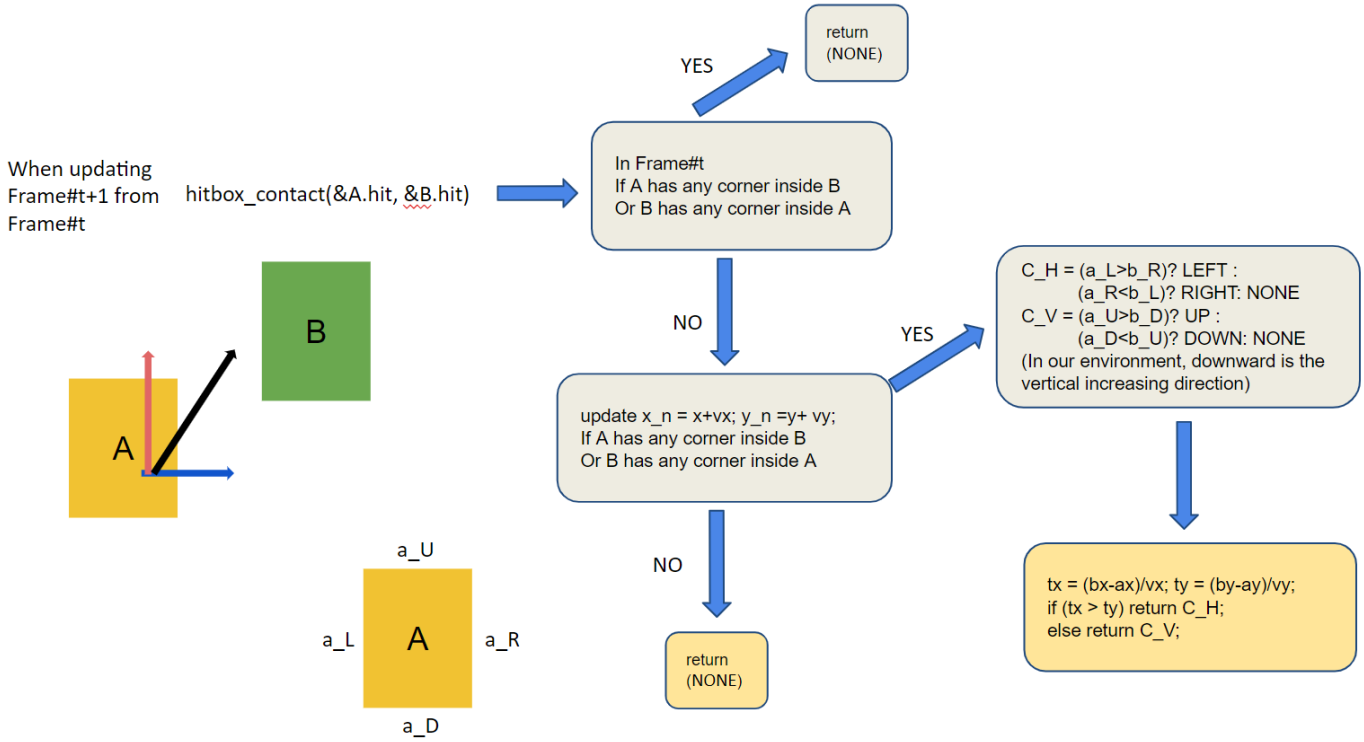
- NONE： A has no contact with B in Frame#t+1
- LEFT： A's left edge has contact with B in Frame#t+1
- RIGHT： A's right edge has contact with B in Frame#t+1
- UP： A's upper edge has contact with B in Frame#t+1
- DOWN： A's lower edge has contact with B in Frame#t+1

```
enum contact hitbox_contact(const hit_box *A, const hit_box *B);
```

```
enum contact{NONE, LEFT, RIGHT, UP, DOWN};
```

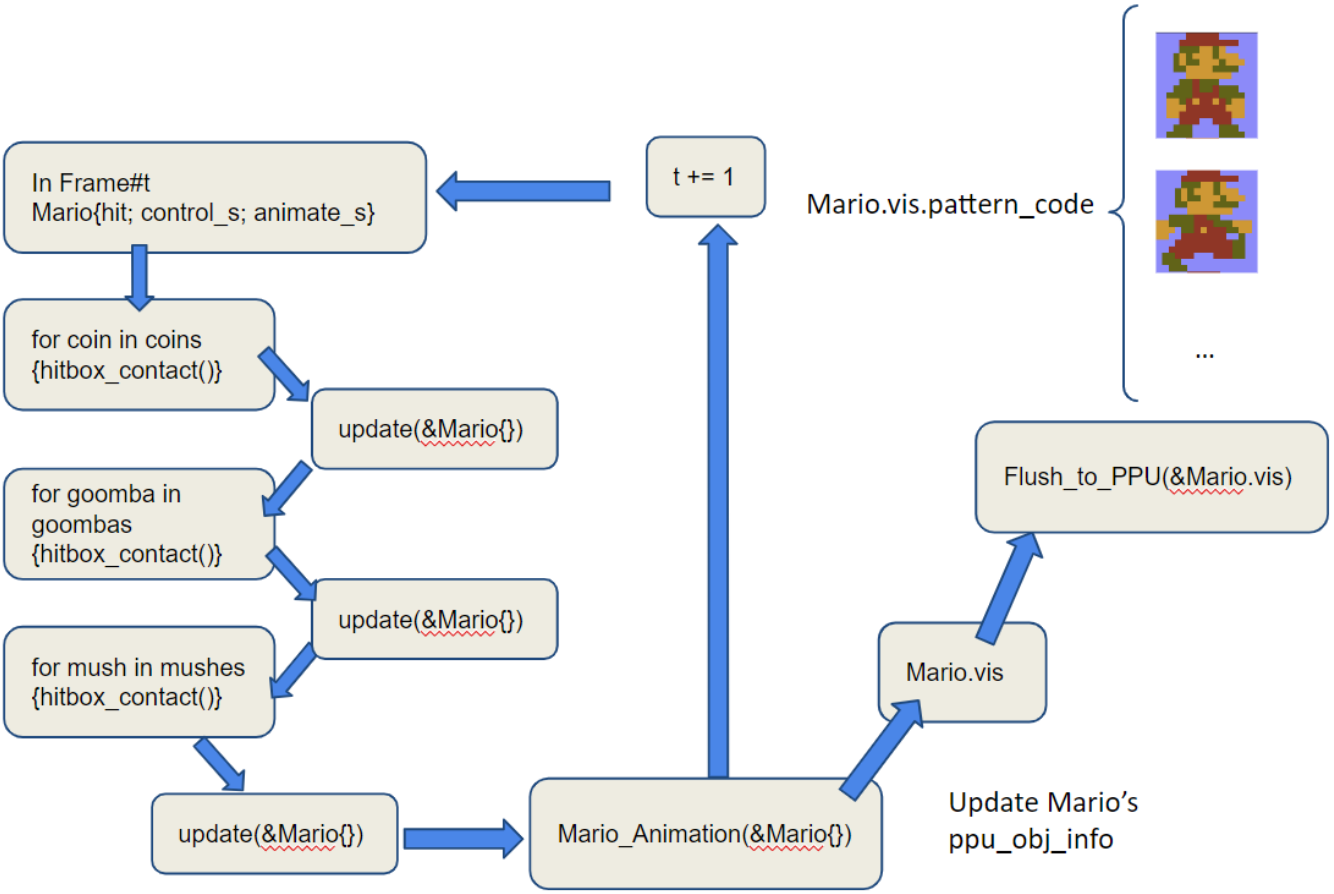
The workflow of the hitbox detection function *enum contact hitbox_contact(const hit_box *A, const hit_box *B)* is the following:

- Notes that this game using **downward** as the vertical increasing direction



Animation

Our game updates the video output at a frame rate around ~40FPS. In each new frame, the software game logic would run hitbox contact check with all the game objects and update their object states. Then the software would call a function to determine which animation sprite image should be shown fro each game object in the new frame and flush the corresponding spride information into the hardware PPU through the avalon bus. (Use the animation for Mario_game_obj as an example)




```
typedef struct{
    unsigned int pattern_code;
    int visible, flip;
    int x, y;
    int extra_info;
} ppu_obj_info;
```

```c
enum mario_control_state{MARIO_NORMAL, MARIO_ANIMATE};
enum mario_animate_state{HIT, ENLARGE, ANI_DEAD};
enum mario_game_state{MARIO_GAME_NORMAL, MARIO_GMAE_HIT};
enum mario_state{SMALL, LARGE, DEAD};

typedef struct {
    int enable;
    int x_block;
    int y_block;
    hit_box hit;
    ppu_obj_info vis;
    enum mario_control_state control_s;
    enum mario_game_state game_s;
    enum mario_state mario_s;
    enum mario_animate_state animate_s;
    float acc_x, acc_y; // Accelerate
    int animate_frame_counter;
} mario;
```

Contributions of Each Team Member

- Zeqi Li (zl3202): image resource collection, .PNG image preprocessing
- Zhiyuan Liu (zl3208): video hardware module (PPU) programing, software game logic programing
- Han Yang (hy2759) & Hangpu Cao (hc3346): sound resource collection, .wav sound file preprocessing, hardware sound module programing
- Shen Gao (sg4140): Player input device programing

Code List:

Hardware :

addr_cal_shift.sv :

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module addr_cal (
                    input logic [79:0]  pattern_info,
        input logic [31:0]  sprite_info,
        input logic [9:0]   hcount,
                    input logic [9:0]   vcount,

                    output logic [15:0]       addr_output,
        output logic        valid);

    // sprite param
    logic [9:0]   x_pos;
          logic [9:0]   y_pos;
    logic [9:0]   shift_amount;
    logic         flip;
    logic         visible;

    assign visible = sprite_info[31];
    assign flip = sprite_info[30];
    assign x_pos = sprite_info[29:20];
    assign y_pos = sprite_info[19:10];
    assign shift_amount = sprite_info[9:0];

    // pattern param
    logic [15:0]    pattern_append, pattern_res_h, pattern_res_v, pattern_act_h, pattern_act_v;
    logic [15:0]    x_pos_16, y_pos_16, shift_amount_16;
    assign pattern_append = pattern_info[79:64];
    assign pattern_res_h = pattern_info[63:48];
    assign pattern_res_v = pattern_info[47:32];
    assign pattern_act_h = pattern_info[31:16];
    assign pattern_act_v = pattern_info[15:0];
    assign x_pos_16[9:0] = x_pos;
```

```
assign y_pos_16[9:0] = y_pos;
assign shift_amount_16[9:0] = shift_amount;

//Calculation param
logic [15:0]    pattern_addr_x, pattern_addr_y;
logic [15:0]    res_x, res_y;

always_comb begin
   if (visible
      && vcount >= y_pos_16 && vcount < y_pos_16 + pattern_act_v
      && hcount >= x_pos_16 && hcount < x_pos_16 + pattern_act_h - shift_amount_16
      ) begin
      res_x = (hcount - x_pos_16 + shift_amount_16) % pattern_res_h;
      res_y = (vcount - y_pos_16) % pattern_res_v;
      //  non filped
      if (flip == 1'b0)begin
         pattern_addr_y = res_y * pattern_res_h;
         pattern_addr_x = res_x;
      end
      // filped
      else begin
         pattern_addr_y = res_y * pattern_res_h;
         pattern_addr_x = pattern_res_h - 1'b_1 - res_x;
                                 // pattern_addr_x = pattern_res_h - res_x;
      end
      addr_output = pattern_append + pattern_addr_y + pattern_addr_x;
      valid = 1'b1;
   end

      else if (visible
      && vcount >= y_pos_16 && vcount < y_pos_16 + pattern_act_v
      ) begin
      res_x = (shift_amount_16) % pattern_res_h;
      res_y = (vcount - y_pos_16) % pattern_res_v;
      pattern_addr_y = res_y * pattern_res_h;
      pattern_addr_x = res_x;
      addr_output = pattern_append + pattern_addr_y + pattern_addr_x;
      valid = 1'b0;
   end

   else begin
      addr_output = 16'd0;
      valid = 1'b0;
                        pattern_addr_x = 16'd0;
```

```
                        pattern_addr_y = 16'd0;
                        res_x = 16'd0;
                        res_y = 16'd0;
        end
    end


endmodule
```

addr_cal_cal.sv :

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module addr_cal (
                    input logic [79:0]  pattern_info,
        input logic [31:0]  sprite_info,
        input logic [9:0]   hcount,
                    input logic [9:0]   vcount,

                    output logic [15:0]       addr_output,
        output logic        valid);

    // sprite param
    logic [9:0]   x_pos;
          logic [9:0]   y_pos;
    logic [9:0]   shift_amount;
    logic         flip;
    logic         visible;

    assign visible = sprite_info[31];
    assign flip = sprite_info[30];
    assign x_pos = sprite_info[29:20];
    assign y_pos = sprite_info[19:10];
    assign shift_amount = sprite_info[9:0];
```

```
// pattern param
logic [15:0]    pattern_append, pattern_res_h, pattern_res_v, pattern_act_h, pattern_act_v;
logic [15:0]    x_pos_16, y_pos_16;
assign pattern_append = pattern_info[79:64];
assign pattern_res_h = pattern_info[63:48];
assign pattern_res_v = pattern_info[47:32];
assign pattern_act_h = pattern_info[31:16];
assign pattern_act_v = pattern_info[15:0];
assign x_pos_16[9:0] = x_pos;
assign y_pos_16[9:0] = y_pos;

//Calculation param
logic [15:0]    pattern_addr_x, pattern_addr_y;
logic [15:0]    res_x, res_y;

always_comb begin
    if (visible
        && vcount >= y_pos_16 && vcount < y_pos_16 + pattern_act_v
        && hcount >= x_pos_16 && hcount < x_pos_16 + pattern_act_h
        ) begin
        res_x = (hcount - x_pos_16) % pattern_res_h;
        res_y = (vcount - y_pos_16) % pattern_res_v;
        //  non filped
        if (flip == 1'b0)begin
            pattern_addr_y = res_y * pattern_res_h;
            pattern_addr_x = res_x;
        end
        // filped
        else begin
            pattern_addr_y = res_y * pattern_res_h;
            pattern_addr_x = pattern_res_h - 1'b_1 - res_x;
                                    // pattern_addr_x = pattern_res_h - res_x;
        end
        addr_output = pattern_append + pattern_addr_y + pattern_addr_x;
        valid = 1'b1;
    end

        else if (visible
        && vcount >= y_pos_16 && vcount < y_pos_16 + pattern_act_v
        ) begin
        res_x = 16'd0;
        res_y = (vcount - y_pos_16) % pattern_res_v;
        pattern_addr_y = res_y * pattern_res_h;
        pattern_addr_x = res_x;
```

```
        addr_output = pattern_append + pattern_addr_y + pattern_addr_x;
        valid = 1'b0;
      end

      else begin
        addr_output = 16'd0;
        valid = 1'b0;
                        pattern_addr_x = 16'd0;
                        pattern_addr_y = 16'd0;
                        res_x = 16'd0;
                        res_y = 16'd0;
      end
   End
Endmodule
```

Block_display.sv:

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module Block_display (input logic        clk,
              input logic       reset,
                    input logic [31:0]  writedata,
                    input logic [9:0]   hcount,
                    input logic [9:0]   vcount,

                    output logic [23:0]       RGB_output);

   // Change for other type=========================
   parameter [5:0] sub_comp_ID = 6'b000010;
   parameter [4:0] pattern_num = 5'd_17;
   parameter [15:0] addr_limit = 16'd_2304;
        parameter [4:0] child_limit = 5'd_9;
        logic [3:0] mem [0:2303];
        logic [23:0] color_plate [0:9];
   logic [79:0] pattern_table [0:16];

   assign pattern_table[0] = {16'd_0, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
   assign pattern_table[1] = {16'd_256, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
```

```
assign pattern_table[2] = {16'd_512, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
assign pattern_table[3] = {16'd_768, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
assign pattern_table[4] = {16'd_1024, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
assign pattern_table[5] = {16'd_1280, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
assign pattern_table[6] = {16'd_1536, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
assign pattern_table[7] = {16'd_1792, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
assign pattern_table[8] = {16'd_2048, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
assign pattern_table[9] = {16'd_1280, 16'd_16, 16'd_16, 16'd_32, 16'd_16};
assign pattern_table[10] = {16'd_1280, 16'd_16, 16'd_16, 16'd_48, 16'd_16};
assign pattern_table[11] = {16'd_1280, 16'd_16, 16'd_16, 16'd_128, 16'd_16};
assign pattern_table[12] = {16'd_2048, 16'd_16, 16'd_16, 16'd_16, 16'd_32};
assign pattern_table[13] = {16'd_2048, 16'd_16, 16'd_16, 16'd_16, 16'd_48};
assign pattern_table[14] = {16'd_2048, 16'd_16, 16'd_16, 16'd_16, 16'd_64};
assign pattern_table[15] = {16'd_2048, 16'd_16, 16'd_16, 16'd_64, 16'd_64};
    assign pattern_table[16] = {16'd_2048, 16'd_16, 16'd_16, 16'd_16, 16'd_64};

    assign color_plate[0] = 24'h9290ff;
    assign color_plate[1] = 24'h908fff;
    assign color_plate[2] = 24'h9a4b00;
    assign color_plate[3] = 24'he69a25;
    assign color_plate[4] = 24'h000000;
    assign color_plate[5] = 24'h974f00;
    assign color_plate[6] = 24'h572200;
    assign color_plate[7] = 24'h1a9300;
    assign color_plate[8] = 24'hfce1ce;
    assign color_plate[9] = 24'hffcdc4;

//================================================

// logic [79:0] ping_pong_pattern_input[0:1];
logic [23:0] ping_pong_RGB_output[0:1][0:8];
logic [15:0] ping_pong_addr_output[0:1][0:8];
logic        ping_pong_addr_out_valid[0:1][0:8];
logic [111:0] ping_pong_stateholder[0:1][0:8];
logic        ping_pong = 1'b0;

    logic [5:0] sub_comp;
    logic [4:0] child_comp;
    logic [3:0] info;
    logic [2:0] input_type;
    logic [12:0] input_msg;
    logic               pp_selc;

    assign sub_comp = writedata[31:26];
```

```
        assign child_comp = writedata[25:21];
        assign info = writedata[20:17];
        assign input_type = writedata[16:14];
        assign pp_selc = writedata[13];
        assign input_msg = writedata[12:0];

        // loop ============================
        integer i, j, k;
        // ================================

    //=========================================
    // Address_calculater change to adapt different setting
    //=========================================
    addr_cal AC_ping_0(.pattern_info(ping_pong_stateholder[0][0][111:32]),
.sprite_info(ping_pong_stateholder[0][0][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][0]), .valid(ping_pong_addr_out_valid[0][0]));
    addr_cal AC_ping_1(.pattern_info(ping_pong_stateholder[0][1][111:32]),
.sprite_info(ping_pong_stateholder[0][1][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][1]), .valid(ping_pong_addr_out_valid[0][1]));
    addr_cal AC_ping_2(.pattern_info(ping_pong_stateholder[0][2][111:32]),
.sprite_info(ping_pong_stateholder[0][2][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][2]), .valid(ping_pong_addr_out_valid[0][2]));
    addr_cal AC_ping_3(.pattern_info(ping_pong_stateholder[0][3][111:32]),
.sprite_info(ping_pong_stateholder[0][3][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][3]), .valid(ping_pong_addr_out_valid[0][3]));
    addr_cal AC_ping_4(.pattern_info(ping_pong_stateholder[0][4][111:32]),
.sprite_info(ping_pong_stateholder[0][4][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][4]), .valid(ping_pong_addr_out_valid[0][4]));
    addr_cal AC_ping_5(.pattern_info(ping_pong_stateholder[0][5][111:32]),
.sprite_info(ping_pong_stateholder[0][5][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][5]), .valid(ping_pong_addr_out_valid[0][5]));
    addr_cal AC_ping_6(.pattern_info(ping_pong_stateholder[0][6][111:32]),
.sprite_info(ping_pong_stateholder[0][6][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][6]), .valid(ping_pong_addr_out_valid[0][6]));
    addr_cal AC_ping_7(.pattern_info(ping_pong_stateholder[0][7][111:32]),
.sprite_info(ping_pong_stateholder[0][7][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][7]), .valid(ping_pong_addr_out_valid[0][7]));
    addr_cal AC_ping_8(.pattern_info(ping_pong_stateholder[0][8][111:32]),
.sprite_info(ping_pong_stateholder[0][8][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][8]), .valid(ping_pong_addr_out_valid[0][8]));

    addr_cal AC_pong_0(.pattern_info(ping_pong_stateholder[1][0][111:32]),
.sprite_info(ping_pong_stateholder[1][0][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][0]), .valid(ping_pong_addr_out_valid[1][0]));
```

```
   addr_cal AC_pong_1(.pattern_info(ping_pong_stateholder[1][1][111:32]),
.sprite_info(ping_pong_stateholder[1][1][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][1]), .valid(ping_pong_addr_out_valid[1][1]));
   addr_cal AC_pong_2(.pattern_info(ping_pong_stateholder[1][2][111:32]),
.sprite_info(ping_pong_stateholder[1][2][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][2]), .valid(ping_pong_addr_out_valid[1][2]));
   addr_cal AC_pong_3(.pattern_info(ping_pong_stateholder[1][3][111:32]),
.sprite_info(ping_pong_stateholder[1][3][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][3]), .valid(ping_pong_addr_out_valid[1][3]));
   addr_cal AC_pong_4(.pattern_info(ping_pong_stateholder[1][4][111:32]),
.sprite_info(ping_pong_stateholder[1][4][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][4]), .valid(ping_pong_addr_out_valid[1][4]));
   addr_cal AC_pong_5(.pattern_info(ping_pong_stateholder[1][5][111:32]),
.sprite_info(ping_pong_stateholder[1][5][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][5]), .valid(ping_pong_addr_out_valid[1][5]));
   addr_cal AC_pong_6(.pattern_info(ping_pong_stateholder[1][6][111:32]),
.sprite_info(ping_pong_stateholder[1][6][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][6]), .valid(ping_pong_addr_out_valid[1][6]));
   addr_cal AC_pong_7(.pattern_info(ping_pong_stateholder[1][7][111:32]),
.sprite_info(ping_pong_stateholder[1][7][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][7]), .valid(ping_pong_addr_out_valid[1][7]));
   addr_cal AC_pong_8(.pattern_info(ping_pong_stateholder[1][8][111:32]),
.sprite_info(ping_pong_stateholder[1][8][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][8]), .valid(ping_pong_addr_out_valid[1][8]));
   // FF Input================================
         always_ff @(posedge clk) begin
      case (info)
         // 4'b1111 to flush *PING(0)/PONG(1)* buffer and clear *THE OPPOSIT* buffer
         4'b1111: begin
            ping_pong = pp_selc;
                                 for (i = 0; i < child_limit; i = i + 1) begin
                                          ping_pong_stateholder[~pp_selc][i][31] = 1'b0;
                                 end

         end

         // 4;b0001 normal write to state holder **ping_pong(pp_selc)**
               4'h0001 : begin
            // *****Change the sub_comp code to match the input
                  if (sub_comp == sub_comp_ID) begin
                                          if (child_comp < child_limit) begin
                              case (input_type)
                                 3'b001: begin
                                    // visible
```

```
                                ping_pong_stateholder[pp_selc][child_comp][31] = input_msg[12];
                                // fliped
                                ping_pong_stateholder[pp_selc][child_comp][30] = input_msg[11];
                                // pattern code
                                if (input_msg[4:0] < pattern_num)
                                    ping_pong_stateholder[pp_selc][child_comp][111:32] =
pattern_table[input_msg[4:0]];
                            end
                            3'b010: begin
                                // x_coordinate
                                ping_pong_stateholder[pp_selc][child_comp][29:20] =
input_msg[9:0];
                            end
                            3'b011: begin
                                // y_coordinate
                                ping_pong_stateholder[pp_selc][child_comp][19:10] =
input_msg[9:0];
                            end
                            3'b100: begin
                                // shift_amount
                                ping_pong_stateholder[pp_selc][child_comp][9:0] =
input_msg[9:0];
                            end
                        endcase
                                    end
                    end
        end
    endcase
        end


     // Output ==================================

        always_comb begin
                for (j = 0; j < child_limit; j = j + 1) begin
                        ping_pong_RGB_output[0][j] =  (ping_pong_addr_output[0][j] <
addr_limit)? color_plate[mem[ping_pong_addr_output[0][j]]] : color_plate[mem[0]];

                        ping_pong_RGB_output[1][j] =  (ping_pong_addr_output[1][j] <
addr_limit)? color_plate[mem[ping_pong_addr_output[1][j]]] : color_plate[mem[0]];
                end

                RGB_output = 24'h9290ff;
                for (k = 0; k < child_limit; k = k + 1) begin
```

```
                        if ((ping_pong_RGB_output[ping_pong][k] != 24'h9290ff) &&
ping_pong_addr_out_valid[ping_pong][k]) begin
                                RGB_output = ping_pong_RGB_output[ping_pong][k];
                                break;
                        end
                end
        end

initial begin
        $readmemh("/user/stud/fall22/hy2759/4840/pro_test/lab3-hw/on_chip_mem/Block.txt",
mem);
end


Endmodule
```

Cloud_diplay.sv:

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module Cloud_display (input logic        clk,
              input logic       reset,
                    input logic [31:0]  writedata,
                    input logic [9:0]   hcount,
                    input logic [9:0]   vcount,

                    output logic [23:0]       RGB_output);

   // Change for other type=========================
   parameter [5:0] sub_comp_ID = 6'b001110; // #14
   parameter [4:0] pattern_num = 5'd_1;
   parameter [15:0] addr_limit = 16'd_768;  // 768
        parameter [4:0] child_limit = 5'd_1;
        logic [3:0] mem [0:383];
        logic [23:0] color_plate [0:3];
   logic [79:0] pattern_table [0:2];

   assign pattern_table[0] = {16'd_0, 16'd_32, 16'd_24, 16'd_32, 16'd_24};
```

```
        assign color_plate[0] = 24'h9290ff;
        assign color_plate[1] = 24'h000000;
        assign color_plate[2] = 24'hffffff;
        assign color_plate[3] = 24'h63adff;

    //============================================

    // logic [79:0] ping_pong_pattern_input[0:1];
    logic [23:0] ping_pong_RGB_output[0:1];
    logic [15:0] ping_pong_addr_output[0:1];
    logic        ping_pong_addr_out_valid[0:1];
    logic [111:0] ping_pong_stateholder[0:1];
    logic        ping_pong = 1'b0;

        logic [5:0] sub_comp;
        logic [4:0] child_comp;
        logic [3:0] info;
        logic [2:0] input_type;
        logic [12:0] input_msg;
        logic               pp_selc;

        assign sub_comp = writedata[31:26];
        assign child_comp = writedata[25:21];
        assign info = writedata[20:17];
        assign input_type = writedata[16:14];
        assign pp_selc = writedata[13];
        assign input_msg = writedata[12:0];

        // loop ============================
        integer i, j, k;
        // ================================

    //============================================
    // Address_calculater change to adapt different setting
    //============================================
        addr_cal AC_ping_0(.pattern_info(ping_pong_stateholder[0][111:32]),
.sprite_info(ping_pong_stateholder[0][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0]), .valid(ping_pong_addr_out_valid[0]));
    addr_cal AC_pong_0(.pattern_info(ping_pong_stateholder[1][111:32]),
.sprite_info(ping_pong_stateholder[1][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1]), .valid(ping_pong_addr_out_valid[1]));
    // FF Input================================
        always_ff @(posedge clk) begin
```

```
 case (info)
    // 4'b1111 to flush *PING(0)/PONG(1)* buffer and clear *THE OPPOSIT* buffer
    4'b1111: begin
       ping_pong = pp_selc;
       ping_pong_stateholder[~pp_selc][31] = 1'b0;
    end

    // 4;b0001 normal write to state holder **ping_pong(pp_selc)**
          4'h0001 : begin
       // *****Change the sub_comp code to match the input
             if (sub_comp == sub_comp_ID) begin
          case (input_type)
             3'b001: begin
                // visible
                ping_pong_stateholder[pp_selc][31] = input_msg[12];
                // fliped
                ping_pong_stateholder[pp_selc][30] = input_msg[11];
                // pattern code
                if (input_msg[4:0] < pattern_num)
                   ping_pong_stateholder[pp_selc][111:32] = pattern_table[input_msg[4:0]];
             end
             3'b010: begin
                // x_coordinate
                ping_pong_stateholder[pp_selc][29:20] = input_msg[9:0];
             end
             3'b011: begin
                // y_coordinate
                ping_pong_stateholder[pp_selc][19:10] = input_msg[9:0];
             end
             3'b100: begin
                // shift_amount
                ping_pong_stateholder[pp_selc][9:0] = input_msg[9:0];
             end
          endcase
                   end
    end
endcase
    end

 // Output ==================================

    always_comb begin
             ping_pong_RGB_output[0] =  (ping_pong_addr_output[0] < addr_limit)?
                      (
```

```
                        (ping_pong_addr_output[0][0])?
                                color_plate[mem[(ping_pong_addr_output[0][15:1])][3:2]] :
                                color_plate[mem[(ping_pong_addr_output[0][15:1])][1:0]]
                ) :
                color_plate[mem[0]];

        ping_pong_RGB_output[1] =  (ping_pong_addr_output[1] < addr_limit)?
                (
                        (ping_pong_addr_output[1][0])?
                                color_plate[mem[(ping_pong_addr_output[1][15:1])][3:2]] :
                                color_plate[mem[(ping_pong_addr_output[1][15:1])][1:0]]
                ) :
                color_plate[mem[0]];

        // assign ping_pong_RGB_output[0] =  (ping_pong_addr_output[0] < addr_limit)?
color_plate[mem[ping_pong_addr_output[0]]] : color_plate[mem[0]];

        // assign ping_pong_RGB_output[1] =  (ping_pong_addr_output[1] < addr_limit)?
color_plate[mem[ping_pong_addr_output[1]]] : color_plate[mem[0]];

                RGB_output = ping_pong_addr_out_valid[ping_pong]?
ping_pong_RGB_output[ping_pong] : 24'h9290ff;
        end

initial begin

$readmemh("/user/stud/fall22/hy2759/4840/pro_test/lab3-hw/on_chip_mem/Cloud_2bit.txt",
mem);
end


endmodule
```

Coin_display.sv:

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */
```

```
module Coin_display (input logic        clk,
                     input logic        reset,
                           input logic [31:0]  writedata,
                           input logic [9:0]   hcount,
                           input logic [9:0]   vcount,

                           output logic [23:0]       RGB_output);

    // Change for other type=========================
    parameter [5:0] sub_comp_ID = 6'b000011;
    parameter [4:0] pattern_num = 5'd_4;
    parameter [15:0] addr_limit = 16'd_512;
            parameter [4:0] child_limit = 5'd_4;
            logic [3:0] mem [0:511];
            logic [23:0] color_plate [0:4];
    logic [79:0] pattern_table [0:3];

    assign pattern_table[0] = {16'd_0, 16'd_8, 16'd_16, 16'd_8, 16'd_16};
    assign pattern_table[1] = {16'd_128, 16'd_8, 16'd_16, 16'd_8, 16'd_16};
    assign pattern_table[2] = {16'd_256, 16'd_8, 16'd_16, 16'd_8, 16'd_16};
    assign pattern_table[3] = {16'd_384, 16'd_8, 16'd_16, 16'd_8, 16'd_16};

            assign color_plate[0] = 24'h9290ff;
            assign color_plate[1] = 24'h908fff;
            assign color_plate[2] = 24'he59a25;
            assign color_plate[3] = 24'hfffeff;
            assign color_plate[4] = 24'hb33425;

    //==============================================

    // logic [79:0] ping_pong_pattern_input[0:1];
    logic [23:0] ping_pong_RGB_output[0:1][0:3];
    logic [15:0] ping_pong_addr_output[0:1][0:3];
    logic        ping_pong_addr_out_valid[0:1][0:3];
    logic [111:0] ping_pong_stateholder[0:1][0:3];
    logic        ping_pong = 1'b0;

            logic [5:0] sub_comp;
            logic [4:0] child_comp;
            logic [3:0] info;
            logic [2:0] input_type;
            logic [12:0] input_msg;
            logic                pp_selc;
```

```
        assign sub_comp = writedata[31:26];
        assign child_comp = writedata[25:21];
        assign info = writedata[20:17];
        assign input_type = writedata[16:14];
        assign pp_selc = writedata[13];
        assign input_msg = writedata[12:0];

        // loop ============================
        integer i, j, k;
        // ================================

    //=========================================
    // Address_calculater change to adapt different setting
    //=========================================
    addr_cal AC_ping_0(.pattern_info(ping_pong_stateholder[0][0][111:32]),
.sprite_info(ping_pong_stateholder[0][0][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][0]), .valid(ping_pong_addr_out_valid[0][0]));
    addr_cal AC_ping_1(.pattern_info(ping_pong_stateholder[0][1][111:32]),
.sprite_info(ping_pong_stateholder[0][1][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][1]), .valid(ping_pong_addr_out_valid[0][1]));
    addr_cal AC_ping_2(.pattern_info(ping_pong_stateholder[0][2][111:32]),
.sprite_info(ping_pong_stateholder[0][2][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][2]), .valid(ping_pong_addr_out_valid[0][2]));
    addr_cal AC_ping_3(.pattern_info(ping_pong_stateholder[0][3][111:32]),
.sprite_info(ping_pong_stateholder[0][3][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][3]), .valid(ping_pong_addr_out_valid[0][3]));

    addr_cal AC_pong_0(.pattern_info(ping_pong_stateholder[1][0][111:32]),
.sprite_info(ping_pong_stateholder[1][0][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][0]), .valid(ping_pong_addr_out_valid[1][0]));
    addr_cal AC_pong_1(.pattern_info(ping_pong_stateholder[1][1][111:32]),
.sprite_info(ping_pong_stateholder[1][1][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][1]), .valid(ping_pong_addr_out_valid[1][1]));
    addr_cal AC_pong_2(.pattern_info(ping_pong_stateholder[1][2][111:32]),
.sprite_info(ping_pong_stateholder[1][2][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][2]), .valid(ping_pong_addr_out_valid[1][2]));
    addr_cal AC_pong_3(.pattern_info(ping_pong_stateholder[1][3][111:32]),
.sprite_info(ping_pong_stateholder[1][3][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][3]), .valid(ping_pong_addr_out_valid[1][3]));

    // FF Input================================
        always_ff @(posedge clk) begin
      case (info)
```

```
        // 4'b1111 to flush *PING(0)/PONG(1)* buffer and clear *THE OPPOSIT* buffer
        4'b1111: begin
           ping_pong = pp_selc;
                                for (i = 0; i < child_limit; i = i + 1) begin
                                        ping_pong_stateholder[~pp_selc][i][31] = 1'b0;
                                end

        end

        // 4;b0001 normal write to state holder **ping_pong(pp_selc)**
              4'h0001 : begin
           // *****Change the sub_comp code to match the input
                 if (sub_comp == sub_comp_ID) begin
                                        if (child_comp < child_limit) begin
                           case (input_type)
                              3'b001: begin
                                 // visible
                                 ping_pong_stateholder[pp_selc][child_comp][31] = input_msg[12];
                                 // fliped
                                 ping_pong_stateholder[pp_selc][child_comp][30] = input_msg[11];
                                 // pattern code
                                 if (input_msg[4:0] < pattern_num)
                                    ping_pong_stateholder[pp_selc][child_comp][111:32] =
pattern_table[input_msg[4:0]];
                              end
                              3'b010: begin
                                 // x_coordinate
                                 ping_pong_stateholder[pp_selc][child_comp][29:20] =
input_msg[9:0];
                              end
                              3'b011: begin
                                 // y_coordinate
                                 ping_pong_stateholder[pp_selc][child_comp][19:10] =
input_msg[9:0];
                              end
                              3'b100: begin
                                 // shift_amount
                                 ping_pong_stateholder[pp_selc][child_comp][9:0] =
input_msg[9:0];
                              end
                           endcase
                                        end
                     end
        end
```

```
    endcase
        end


     // Output ================================

        always_comb begin
                for (j = 0; j < child_limit; j = j + 1) begin
                        ping_pong_RGB_output[0][j] =  (ping_pong_addr_output[0][j] <
addr_limit)? color_plate[mem[ping_pong_addr_output[0][j]]] : color_plate[mem[0]];

                        ping_pong_RGB_output[1][j] =  (ping_pong_addr_output[1][j] <
addr_limit)? color_plate[mem[ping_pong_addr_output[1][j]]] : color_plate[mem[0]];
                end

                RGB_output = 24'h9290ff;
                for (k = 0; k < child_limit; k = k + 1) begin
                        if ((ping_pong_RGB_output[ping_pong][k] != 24'h9290ff) &&
ping_pong_addr_out_valid[ping_pong][k]) begin
                                RGB_output = ping_pong_RGB_output[ping_pong][k];
                                break;
                        end
                end
        end

initial begin
        $readmemh("/user/stud/fall22/hy2759/4840/pro_test/lab3-hw/on_chip_mem/Coin.txt",
mem);
end


endmodule
```

Flag_display.sv:

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */
```

```
module Coin_display (input logic        clk,
                     input logic        reset,
                           input logic [31:0]  writedata,
                           input logic [9:0]   hcount,
                           input logic [9:0]   vcount,

                           output logic [23:0]       RGB_output);

    // Change for other type=========================
    parameter [5:0] sub_comp_ID = 6'b000011;
    parameter [4:0] pattern_num = 5'd_4;
    parameter [15:0] addr_limit = 16'd_512;
            parameter [4:0] child_limit = 5'd_4;
            logic [3:0] mem [0:511];
            logic [23:0] color_plate [0:4];
    logic [79:0] pattern_table [0:3];

    assign pattern_table[0] = {16'd_0, 16'd_8, 16'd_16, 16'd_8, 16'd_16};
    assign pattern_table[1] = {16'd_128, 16'd_8, 16'd_16, 16'd_8, 16'd_16};
    assign pattern_table[2] = {16'd_256, 16'd_8, 16'd_16, 16'd_8, 16'd_16};
    assign pattern_table[3] = {16'd_384, 16'd_8, 16'd_16, 16'd_8, 16'd_16};

            assign color_plate[0] = 24'h9290ff;
            assign color_plate[1] = 24'h908fff;
            assign color_plate[2] = 24'he59a25;
            assign color_plate[3] = 24'hfffeff;
            assign color_plate[4] = 24'hb33425;

    //==============================================

    // logic [79:0] ping_pong_pattern_input[0:1];
    logic [23:0] ping_pong_RGB_output[0:1][0:3];
    logic [15:0] ping_pong_addr_output[0:1][0:3];
    logic        ping_pong_addr_out_valid[0:1][0:3];
    logic [111:0] ping_pong_stateholder[0:1][0:3];
    logic        ping_pong = 1'b0;

            logic [5:0] sub_comp;
            logic [4:0] child_comp;
            logic [3:0] info;
            logic [2:0] input_type;
            logic [12:0] input_msg;
            logic                pp_selc;
```

```
        assign sub_comp = writedata[31:26];
        assign child_comp = writedata[25:21];
        assign info = writedata[20:17];
        assign input_type = writedata[16:14];
        assign pp_selc = writedata[13];
        assign input_msg = writedata[12:0];

        // loop ============================
        integer i, j, k;
        // ================================

   //=========================================
   // Address_calculater change to adapt different setting
   //=========================================
   addr_cal AC_ping_0(.pattern_info(ping_pong_stateholder[0][0][111:32]),
.sprite_info(ping_pong_stateholder[0][0][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][0]), .valid(ping_pong_addr_out_valid[0][0]));
   addr_cal AC_ping_1(.pattern_info(ping_pong_stateholder[0][1][111:32]),
.sprite_info(ping_pong_stateholder[0][1][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][1]), .valid(ping_pong_addr_out_valid[0][1]));
   addr_cal AC_ping_2(.pattern_info(ping_pong_stateholder[0][2][111:32]),
.sprite_info(ping_pong_stateholder[0][2][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][2]), .valid(ping_pong_addr_out_valid[0][2]));
   addr_cal AC_ping_3(.pattern_info(ping_pong_stateholder[0][3][111:32]),
.sprite_info(ping_pong_stateholder[0][3][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][3]), .valid(ping_pong_addr_out_valid[0][3]));

   addr_cal AC_pong_0(.pattern_info(ping_pong_stateholder[1][0][111:32]),
.sprite_info(ping_pong_stateholder[1][0][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][0]), .valid(ping_pong_addr_out_valid[1][0]));
   addr_cal AC_pong_1(.pattern_info(ping_pong_stateholder[1][1][111:32]),
.sprite_info(ping_pong_stateholder[1][1][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][1]), .valid(ping_pong_addr_out_valid[1][1]));
   addr_cal AC_pong_2(.pattern_info(ping_pong_stateholder[1][2][111:32]),
.sprite_info(ping_pong_stateholder[1][2][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][2]), .valid(ping_pong_addr_out_valid[1][2]));
   addr_cal AC_pong_3(.pattern_info(ping_pong_stateholder[1][3][111:32]),
.sprite_info(ping_pong_stateholder[1][3][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][3]), .valid(ping_pong_addr_out_valid[1][3]));

   // FF Input=================================
        always_ff @(posedge clk) begin
      case (info)
        // 4'b1111 to flush *PING(0)/PONG(1)* buffer and clear *THE OPPOSIT* buffer
```

```
        4'b1111: begin
           ping_pong = pp_selc;
                                for (i = 0; i < child_limit; i = i + 1) begin
                                        ping_pong_stateholder[~pp_selc][i][31] = 1'b0;
                                end

        end

        // 4;b0001 normal write to state holder **ping_pong(pp_selc)**
              4'h0001 : begin
           // *****Change the sub_comp code to match the input
                 if (sub_comp == sub_comp_ID) begin
                                        if (child_comp < child_limit) begin
                              case (input_type)
                                 3'b001: begin
                                    // visible
                                    ping_pong_stateholder[pp_selc][child_comp][31] = input_msg[12];
                                    // fliped
                                    ping_pong_stateholder[pp_selc][child_comp][30] = input_msg[11];
                                    // pattern code
                                    if (input_msg[4:0] < pattern_num)
                                       ping_pong_stateholder[pp_selc][child_comp][111:32] =
pattern_table[input_msg[4:0]];
                                 end
                                 3'b010: begin
                                    // x_coordinate
                                    ping_pong_stateholder[pp_selc][child_comp][29:20] =
input_msg[9:0];
                                 end
                                 3'b011: begin
                                    // y_coordinate
                                    ping_pong_stateholder[pp_selc][child_comp][19:10] =
input_msg[9:0];
                                 end
                                 3'b100: begin
                                    // shift_amount
                                    ping_pong_stateholder[pp_selc][child_comp][9:0] =
input_msg[9:0];
                                 end
                              endcase
                                        end
                     end
        end
     endcase
```

```
        end


    // Output ======================================

        always_comb begin
                for (j = 0; j < child_limit; j = j + 1) begin
                        ping_pong_RGB_output[0][j] =  (ping_pong_addr_output[0][j] <
addr_limit)? color_plate[mem[ping_pong_addr_output[0][j]]] : color_plate[mem[0]];

                        ping_pong_RGB_output[1][j] =  (ping_pong_addr_output[1][j] <
addr_limit)? color_plate[mem[ping_pong_addr_output[1][j]]] : color_plate[mem[0]];
                end

                RGB_output = 24'h9290ff;
                for (k = 0; k < child_limit; k = k + 1) begin
                        if ((ping_pong_RGB_output[ping_pong][k] != 24'h9290ff) &&
ping_pong_addr_out_valid[ping_pong][k]) begin
                                RGB_output = ping_pong_RGB_output[ping_pong][k];
                                break;
                        end
                end
        end

initial begin
        $readmemh("/user/stud/fall22/hy2759/4840/pro_test/lab3-hw/on_chip_mem/Coin.txt",
mem);
end


endmodule
```

Goomba_display.sv

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module Goomba_display (input logic        clk,
            input logic       reset,
                input logic [31:0]  writedata,
```

```
                input logic [9:0]   hcount,
                input logic [9:0]   vcount,

                output logic [23:0]       RGB_output);

// Change for other type=========================
parameter [5:0] sub_comp_ID = 6'b000101; // #5
parameter [4:0] pattern_num = 5'd_2;
parameter [15:0] addr_limit = 16'd_384;  // 384
        parameter [4:0] child_limit = 5'd_2;
        logic [3:0] mem [0:191];
        logic [23:0] color_plate [0:3];
logic [79:0] pattern_table [0:1];

assign pattern_table[0] = {16'd_0, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
assign pattern_table[1] = {16'd_256, 16'd_16, 16'd_8, 16'd_16, 16'd_8};

        assign color_plate[0] = 24'h9290ff;
        assign color_plate[1] = 24'h9c4a00;
        assign color_plate[2] = 24'h000000;
        assign color_plate[3] = 24'hffcec5;

//===============================================

// logic [79:0] ping_pong_pattern_input[0:1];
logic [23:0] ping_pong_RGB_output[0:1][0:1];
logic [15:0] ping_pong_addr_output[0:1][0:1];
logic        ping_pong_addr_out_valid[0:1][0:1];
logic [111:0] ping_pong_stateholder[0:1][0:1];
logic        ping_pong = 1'b0;

        logic [5:0] sub_comp;
        logic [4:0] child_comp;
        logic [3:0] info;
        logic [2:0] input_type;
        logic [12:0] input_msg;
        logic               pp_selc;

        assign sub_comp = writedata[31:26];
        assign child_comp = writedata[25:21];
        assign info = writedata[20:17];
        assign input_type = writedata[16:14];
        assign pp_selc = writedata[13];
        assign input_msg = writedata[12:0];
```

```
        // loop ============================
        integer i, j, k;
        // ================================

    //=========================================
    // Address_calculater change to adapt different setting
    //=========================================
        addr_cal AC_ping_0(.pattern_info(ping_pong_stateholder[0][0][111:32]),
.sprite_info(ping_pong_stateholder[0][0][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][0]), .valid(ping_pong_addr_out_valid[0][0]));
    addr_cal AC_ping_1(.pattern_info(ping_pong_stateholder[0][1][111:32]),
.sprite_info(ping_pong_stateholder[0][1][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][1]), .valid(ping_pong_addr_out_valid[0][1]));

    addr_cal AC_pong_0(.pattern_info(ping_pong_stateholder[1][0][111:32]),
.sprite_info(ping_pong_stateholder[1][0][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][0]), .valid(ping_pong_addr_out_valid[1][0]));
    addr_cal AC_pong_1(.pattern_info(ping_pong_stateholder[1][1][111:32]),
.sprite_info(ping_pong_stateholder[1][1][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][1]), .valid(ping_pong_addr_out_valid[1][1]));
    // FF Input================================
        always_ff @(posedge clk) begin
      case (info)
        // 4'b1111 to flush *PING(0)/PONG(1)* buffer and clear *THE OPPOSIT* buffer
        4'b1111: begin
            ping_pong = pp_selc;
                                for (i = 0; i < child_limit; i = i + 1) begin
                                        ping_pong_stateholder[~pp_selc][i][31] = 1'b0;
                                end

        end

        // 4;b0001 normal write to state holder **ping_pong(pp_selc)**
              4'h0001 : begin
            // *****Change the sub_comp code to match the input
                if (sub_comp == sub_comp_ID) begin
                                        if (child_comp < child_limit) begin
                            case (input_type)
                               3'b001: begin
                                  // visible
                                  ping_pong_stateholder[pp_selc][child_comp][31] = input_msg[12];
                                  // fliped
                                  ping_pong_stateholder[pp_selc][child_comp][30] = input_msg[11];
```

```
                                    // pattern code
                                    if (input_msg[4:0] < pattern_num)
                                        ping_pong_stateholder[pp_selc][child_comp][111:32] =
pattern_table[input_msg[4:0]];
                                end
                                3'b010: begin
                                    // x_coordinate
                                    ping_pong_stateholder[pp_selc][child_comp][29:20] =
input_msg[9:0];
                                end
                                3'b011: begin
                                    // y_coordinate
                                    ping_pong_stateholder[pp_selc][child_comp][19:10] =
input_msg[9:0];
                                end
                                3'b100: begin
                                    // shift_amount
                                    ping_pong_stateholder[pp_selc][child_comp][9:0] =
input_msg[9:0];
                                end
                            endcase
                                        end
                        end
            end
        endcase
            end


        // Output ==================================

            always_comb begin
                    for (j = 0; j < child_limit; j = j + 1) begin
                            ping_pong_RGB_output[0][j] =  (ping_pong_addr_output[0][j] <
addr_limit)?
                            (
                                    (ping_pong_addr_output[0][j][0])?
                                            color_plate[mem[(ping_pong_addr_output[0][j][15:1])][3:2]]
:
                                            color_plate[mem[(ping_pong_addr_output[0][j][15:1])][1:0]]
                            ) :
                            color_plate[mem[0]];

                            ping_pong_RGB_output[1][j] =  (ping_pong_addr_output[1][j] <
addr_limit)?
```

```
                        (
                                (ping_pong_addr_output[1][j][0])?
                                        color_plate[mem[(ping_pong_addr_output[1][j][15:1])][3:2]]
:
                                        color_plate[mem[(ping_pong_addr_output[1][j][15:1])][1:0]]
                        ) :
                        color_plate[mem[0]];
                end

                RGB_output = 24'h9290ff;
                for (k = 0; k < child_limit; k = k + 1) begin
                        if ((ping_pong_RGB_output[ping_pong][k] != 24'h9290ff) &&
ping_pong_addr_out_valid[ping_pong][k]) begin
                                RGB_output = ping_pong_RGB_output[ping_pong][k];
                                break;
                        end
                end
        end

initial begin

$readmemh("/user/stud/fall22/hy2759/4840/pro_test/lab3-hw/on_chip_mem/Goomba_2bit.txt",
mem);
end


endmodule
```

Ground_display.sv:

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module Ground_display (input logic        clk,
            input logic       reset,
```

```
                input logic [31:0]  writedata,
                input logic [9:0]   hcount,
                input logic [9:0]   vcount,

                output logic [23:0]       RGB_output);

    // Change for other type=========================
    parameter [5:0] sub_comp_ID = 6'b001111; //#15
    parameter [4:0] pattern_num = 5'd_1;
    parameter [15:0] addr_limit = 16'd_256;
        logic [3:0] mem [0:127];
        logic [23:0] color_plate [0:3];
    logic [79:0] pattern_table [0:1];

    assign pattern_table[0] = {16'd_0, 16'd_16, 16'd_16, 16'd_650, 16'd_32};

        assign color_plate[0] = 24'h9290ff;
        assign color_plate[1] = 24'hb53120;
        assign color_plate[2] = 24'h6b6d00;
        assign color_plate[3] = 24'hea9e22;

        parameter [9:0] ground_height = 10'd_368;   // The 24&25 blocks on the floor. Total heigh
is 25 block
    //==============================================

    // logic [79:0] ping_pong_pattern_input[0:1];
    logic [23:0] ping_pong_RGB_output[0:1];
    logic [15:0] ping_pong_addr_output[0:1];
    logic        ping_pong_addr_out_valid[0:1];
    logic [111:0] ping_pong_stateholder[0:1];
    logic        ping_pong = 1'b0;

        logic [5:0] sub_comp;
        logic [4:0] child_comp;
        logic [3:0] info;
        logic [2:0] input_type;
        logic [12:0] input_msg;
        logic               pp_selc;

        assign sub_comp = writedata[31:26];
        assign child_comp = writedata[25:21];
        assign info = writedata[20:17];
        assign input_type = writedata[16:14];
        assign pp_selc = writedata[13];
```

```
        assign input_msg = writedata[12:0];
        // Fixed Height
        assign ping_pong_stateholder[0][19:10] = ground_height;
        assign ping_pong_stateholder[1][19:10] = ground_height;
        logic [9:0] l_edge = 10'd0;
        logic [9:0] r_edge = 10'd0;

    //=========================================
    // Address_calculater change to adapt different setting
    //=========================================
    addr_cal AC_ping_0(.pattern_info(ping_pong_stateholder[0][111:32]),
.sprite_info(ping_pong_stateholder[0][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0]), .valid(ping_pong_addr_out_valid[0]));
    addr_cal AC_pong_0(.pattern_info(ping_pong_stateholder[1][111:32]),
.sprite_info(ping_pong_stateholder[1][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1]), .valid(ping_pong_addr_out_valid[1]));
    // FF Input=================================
        always_ff @(posedge clk) begin
      case (info)
        // 4'b1111 to flush *PING(0)/PONG(1)* buffer and clear *THE OPPOSIT* buffer
        4'b1111: begin
            ping_pong = pp_selc;
            ping_pong_stateholder[~pp_selc][31] = 1'b0;
        end

        // 4;b0001 normal write to state holder **ping_pong(pp_selc)**
              4'h0001 : begin
            // *****Change the sub_comp code to match the input
                  if (sub_comp == sub_comp_ID) begin
                case (input_type)
                    3'b001: begin
                        // visible
                        ping_pong_stateholder[pp_selc][31] = input_msg[12];
                        // fliped
                        ping_pong_stateholder[pp_selc][30] = input_msg[11];
                        // pattern code
                        if (input_msg[4:0] < pattern_num)
                            ping_pong_stateholder[pp_selc][111:32] = pattern_table[input_msg[4:0]];
                    end
                    3'b010: begin
                        // x_coordinate
                        ping_pong_stateholder[pp_selc][29:20] = input_msg[9:0];
                    end
                    3'b011: begin
```

```
                // y_coordinate
                // ping_pong_stateholder[pp_selc][19:10] = input_msg[9:0];
                                                    l_edge = input_msg[9:0];
            end
            3'b100: begin
                // shift_amount
                // ping_pong_stateholder[pp_selc][9:0] = input_msg[9:0];
                                                    r_edge = input_msg[9:0];
            end
          endcase
                end
    end
  endcase
    end


   // Output ====================================

    assign ping_pong_RGB_output[0] =  (ping_pong_addr_output[0] < addr_limit)?
                    (
                        (ping_pong_addr_output[0][0])?
                            color_plate[mem[(ping_pong_addr_output[0][15:1])][3:2]] :
                            color_plate[mem[(ping_pong_addr_output[0][15:1])][1:0]]
                    ) :
                    color_plate[mem[0]];

    assign ping_pong_RGB_output[1] =  (ping_pong_addr_output[1] < addr_limit)?
                    (
                        (ping_pong_addr_output[1][0])?
                            color_plate[mem[(ping_pong_addr_output[1][15:1])][3:2]] :
                            color_plate[mem[(ping_pong_addr_output[1][15:1])][1:0]]
                    ) :
                    color_plate[mem[0]];

    assign RGB_output = ping_pong_addr_out_valid[ping_pong]?
                                        (((hcount < l_edge)||(hcount > r_edge))?
ping_pong_RGB_output[ping_pong] : 24'h9290ff)
                                        : 24'h9290ff;



initial begin
```

```
$readmemh("/user/stud/fall22/hy2759/4840/pro_test/lab3-hw/on_chip_mem/Ground_2bit.txt",
mem);
end


endmodule
```

Ppu.sv:

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module ppu (input logic        clk,
                input logic             reset,
                   input logic [31:0]  writedata,
                   input logic [2:0]  address,
                   input logic [9:0]  hcount,
                   input logic [9:0]   vcount,

                   output logic [23:0]       RGB_output);

          logic [23:0] background = 24'd_0;
          logic [23:0] RGB_list[0:40];

          // loop ============================
          integer i;
          // ================================

          Mario_display Mario_0(.clk(clk), .reset(reset), .writedata(writedata), .hcount(hcount),
.vcount(vcount), .RGB_output(RGB_list[3]));
          Block_display Block_0(.clk(clk), .reset(reset), .writedata(writedata), .hcount(hcount),
.vcount(vcount), .RGB_output(RGB_list[7]));
          Coin_display Coin_0(.clk(clk), .reset(reset), .writedata(writedata), .hcount(hcount),
.vcount(vcount), .RGB_output(RGB_list[9]));
```

```
	Cloud_display Cloud_0(.clk(clk), .reset(reset), .writedata(writedata), .hcount(hcount),
.vcount(vcount), .RGB_output(RGB_list[16]));
	Tube_display Tube_0(.clk(clk), .reset(reset), .writedata(writedata), .hcount(hcount),
.vcount(vcount), .RGB_output(RGB_list[15]));
	Mush_display Mush_0(.clk(clk), .reset(reset), .writedata(writedata), .hcount(hcount),
.vcount(vcount), .RGB_output(RGB_list[12]));
	Ground_display Ground_0(.clk(clk), .reset(reset), .writedata(writedata), .hcount(hcount),
.vcount(vcount), .RGB_output(RGB_list[14]));
	Goomba_display Goomba_0(.clk(clk), .reset(reset), .writedata(writedata),
.hcount(hcount), .vcount(vcount), .RGB_output(RGB_list[5]));
	always_comb begin

		RGB_list[0] = 24'h9290ff;
		RGB_list[1] = 24'h9290ff;
		RGB_list[2] = 24'h9290ff;
		// RGB_list[3] = 24'h9290ff;	//Mario
		RGB_list[4] = 24'h9290ff;
		// RGB_list[5] = 24'h9290ff;	//Goomba
		RGB_list[6] = 24'h9290ff;
		// RGB_list[7] = 24'h9290ff;	//Block
		RGB_list[8] = 24'h9290ff;
		// RGB_list[9] = 24'h9290ff;	//Coin
		RGB_list[10] = 24'h9290ff;
		RGB_list[11] = 24'h9290ff;
		// RGB_list[12] = 24'h9290ff;  //Mush
		RGB_list[13] = 24'h9290ff;
		// RGB_list[14] = 24'h9290ff;  //Ground
		// RGB_list[15] = 24'h9290ff;  //Tube
		// RGB_list[16] = 24'h9290ff;  //Cloud
		RGB_list[17] = 24'h9290ff;
		RGB_list[18] = 24'h9290ff;
		RGB_list[19] = 24'h9290ff;
		RGB_output = 24'h9290ff;
		for (i = 0; i < 20; i = i + 1) begin
			if (RGB_list[i] != 24'h9290ff) begin
				RGB_output = RGB_list[i];
				break;
			end
		end
	end

endmodule
```

Mario_display.sv:

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module Mario_display (input logic        clk,
                      input logic        reset,
                            input logic [31:0]  writedata,
                            input logic [9:0]   hcount,
                            input logic [9:0]   vcount,

                            output logic [23:0]       RGB_output);

    // Change for other type=========================
    parameter [5:0] sub_comp_ID = 6'b000001;
    parameter [4:0] pattern_num = 5'd_19;
    parameter [15:0] addr_limit = 16'd_7168;
          logic [3:0] mem [0:3583];
          logic [23:0] color_plate [0:3];
    logic [79:0] pattern_table [0:18];

    assign pattern_table[0] = {16'd_0, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
    assign pattern_table[1] = {16'd_256, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
    assign pattern_table[2] = {16'd_512, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
    assign pattern_table[3] = {16'd_768, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
    assign pattern_table[4] = {16'd_1024, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
    assign pattern_table[5] = {16'd_1280, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
    assign pattern_table[6] = {16'd_1536, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
    assign pattern_table[7] = {16'd_1792, 16'd_16, 16'd_32, 16'd_16, 16'd_32};
    assign pattern_table[8] = {16'd_2304, 16'd_16, 16'd_32, 16'd_16, 16'd_32};
    assign pattern_table[9] = {16'd_2816, 16'd_16, 16'd_32, 16'd_16, 16'd_32};
    assign pattern_table[10] = {16'd_3328, 16'd_16, 16'd_32, 16'd_16, 16'd_32};
    assign pattern_table[11] = {16'd_3840, 16'd_16, 16'd_32, 16'd_16, 16'd_32};
    assign pattern_table[12] = {16'd_4352, 16'd_16, 16'd_32, 16'd_16, 16'd_32};
    assign pattern_table[13] = {16'd_4864, 16'd_16, 16'd_24, 16'd_16, 16'd_24};
    assign pattern_table[14] = {16'd_5248, 16'd_16, 16'd_24, 16'd_16, 16'd_24};
    assign pattern_table[15] = {16'd_5632, 16'd_16, 16'd_32, 16'd_16, 16'd_32};
```

```
    assign pattern_table[16] = {16'd_6144, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
    assign pattern_table[17] = {16'd_6400, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
    assign pattern_table[18] = {16'd_6656, 16'd_16, 16'd_32, 16'd_16, 16'd_32};

        assign color_plate[0] = 24'h9290ff;
        assign color_plate[1] = 24'hb53120;
        assign color_plate[2] = 24'h6b6d00;
        assign color_plate[3] = 24'hea9e22;

    //============================================

    // logic [79:0] ping_pong_pattern_input[0:1];
    logic [23:0] ping_pong_RGB_output[0:1];
    logic [15:0] ping_pong_addr_output[0:1];
    logic        ping_pong_addr_out_valid[0:1];
    logic [111:0] ping_pong_stateholder[0:1];
    logic        ping_pong = 1'b0;

        logic [5:0] sub_comp;
        logic [4:0] child_comp;
        logic [3:0] info;
        logic [2:0] input_type;
        logic [12:0] input_msg;
        logic               pp_selc;

        assign sub_comp = writedata[31:26];
        assign child_comp = writedata[25:21];
        assign info = writedata[20:17];
        assign input_type = writedata[16:14];
        assign pp_selc = writedata[13];
        assign input_msg = writedata[12:0];

    //==========================================
    // Address_calculater change to adapt different setting
    //==========================================
    addr_cal AC_ping_0(.pattern_info(ping_pong_stateholder[0][111:32]),
.sprite_info(ping_pong_stateholder[0][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0]), .valid(ping_pong_addr_out_valid[0]));
    addr_cal AC_pong_0(.pattern_info(ping_pong_stateholder[1][111:32]),
.sprite_info(ping_pong_stateholder[1][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1]), .valid(ping_pong_addr_out_valid[1]));
    // FF Input================================
        always_ff @(posedge clk) begin
       case (info)
```

```
    // 4'b1111 to flush *PING(0)/PONG(1)* buffer and clear *THE OPPOSIT* buffer
    4'b1111: begin
       ping_pong = pp_selc;
       ping_pong_stateholder[~pp_selc][31] = 1'b0;
    end

    // 4;b0001 normal write to state holder **ping_pong(pp_selc)**
          4'h0001 : begin
       // *****Change the sub_comp code to match the input
             if (sub_comp == sub_comp_ID) begin
          case (input_type)
             3'b001: begin
                // visible
                ping_pong_stateholder[pp_selc][31] = input_msg[12];
                // fliped
                ping_pong_stateholder[pp_selc][30] = input_msg[11];
                // pattern code
                if (input_msg[4:0] < pattern_num)
                   ping_pong_stateholder[pp_selc][111:32] = pattern_table[input_msg[4:0]];
             end
             3'b010: begin
                // x_coordinate
                ping_pong_stateholder[pp_selc][29:20] = input_msg[9:0];
             end
             3'b011: begin
                // y_coordinate
                ping_pong_stateholder[pp_selc][19:10] = input_msg[9:0];
             end
             3'b100: begin
                // shift_amount
                ping_pong_stateholder[pp_selc][9:0] = input_msg[9:0];
             end
          endcase
                   end
    end
endcase
    end


 // Output ====================================
    always_comb begin
             ping_pong_RGB_output[0] =  (ping_pong_addr_output[0] < addr_limit)?
                      (
                               (ping_pong_addr_output[0][0])?
```

```
                                color_plate[mem[(ping_pong_addr_output[0][15:1])][3:2]] :
                                color_plate[mem[(ping_pong_addr_output[0][15:1])][1:0]]
                        ) :
                        color_plate[mem[0]];

                ping_pong_RGB_output[1] =  (ping_pong_addr_output[1] < addr_limit)?
                        (
                                (ping_pong_addr_output[1][0])?
                                        color_plate[mem[(ping_pong_addr_output[1][15:1])][3:2]] :
                                        color_plate[mem[(ping_pong_addr_output[1][15:1])][1:0]]
                        ) :
                        color_plate[mem[0]];

        // assign ping_pong_RGB_output[0] =  (ping_pong_addr_output[0] < addr_limit)?
color_plate[mem[ping_pong_addr_output[0]]] : color_plate[mem[0]];

        // assign ping_pong_RGB_output[1] =  (ping_pong_addr_output[1] < addr_limit)?
color_plate[mem[ping_pong_addr_output[1]]] : color_plate[mem[0]];

                RGB_output = ping_pong_addr_out_valid[ping_pong]?
ping_pong_RGB_output[ping_pong] : 24'h9290ff;
        end



initial begin

$readmemh("/user/stud/fall22/hy2759/4840/pro_test/lab3-hw/on_chip_mem/Mario_2bit.txt",
mem);
end


endmodule
```

Mush_display.sv:

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module Mush_display (input logic        clk,
```

```
        input logic       reset,
              input logic [31:0]  writedata,
              input logic [9:0]   hcount,
              input logic [9:0]   vcount,

              output logic [23:0]       RGB_output);

// Change for other type=========================
parameter [5:0] sub_comp_ID = 6'b001001; // #9
parameter [4:0] pattern_num = 5'd_2;
parameter [15:0] addr_limit = 16'd_512;  // 512
      parameter [4:0] child_limit = 5'd_2;
      logic [3:0] mem [0:255];
      logic [23:0] color_plate [0:3];
logic [79:0] pattern_table [0:1];

assign pattern_table[0] = {16'd_0, 16'd_16, 16'd_16, 16'd_16, 16'd_16};
      assign pattern_table[1] = {16'd_256, 16'd_16, 16'd_16, 16'd_16, 16'd_16};

      assign color_plate[0] = 24'h9290ff;
      assign color_plate[1] = 24'he69c21;
      assign color_plate[2] = 24'h0c9300;
      assign color_plate[3] = 24'hfffeff;

//=============================================

// logic [79:0] ping_pong_pattern_input[0:1];
logic [23:0] ping_pong_RGB_output[0:1][0:1];
logic [15:0] ping_pong_addr_output[0:1][0:1];
logic       ping_pong_addr_out_valid[0:1][0:1];
logic [111:0] ping_pong_stateholder[0:1][0:1];
logic       ping_pong = 1'b0;

      logic [5:0] sub_comp;
      logic [4:0] child_comp;
      logic [3:0] info;
      logic [2:0] input_type;
      logic [12:0] input_msg;
      logic               pp_selc;

      assign sub_comp = writedata[31:26];
      assign child_comp = writedata[25:21];
      assign info = writedata[20:17];
      assign input_type = writedata[16:14];
```

```
        assign pp_selc = writedata[13];
        assign input_msg = writedata[12:0];

        // loop ============================
        integer i, j, k;
        // ================================

    //=========================================
    // Address_calculater change to adapt different setting
    //=========================================
    addr_cal AC_ping_0(.pattern_info(ping_pong_stateholder[0][0][111:32]),
.sprite_info(ping_pong_stateholder[0][0][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][0]), .valid(ping_pong_addr_out_valid[0][0]));
    addr_cal AC_ping_1(.pattern_info(ping_pong_stateholder[0][1][111:32]),
.sprite_info(ping_pong_stateholder[0][1][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][1]), .valid(ping_pong_addr_out_valid[0][1]));

    addr_cal AC_pong_0(.pattern_info(ping_pong_stateholder[1][0][111:32]),
.sprite_info(ping_pong_stateholder[1][0][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][0]), .valid(ping_pong_addr_out_valid[1][0]));
    addr_cal AC_pong_1(.pattern_info(ping_pong_stateholder[1][1][111:32]),
.sprite_info(ping_pong_stateholder[1][1][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][1]), .valid(ping_pong_addr_out_valid[1][1]));

    // FF Input================================
        always_ff @(posedge clk) begin
      case (info)
        // 4'b1111 to flush *PING(0)/PONG(1)* buffer and clear *THE OPPOSIT* buffer
        4'b1111: begin
            ping_pong = pp_selc;
                                for (i = 0; i < child_limit; i = i + 1) begin
                                        ping_pong_stateholder[~pp_selc][i][31] = 1'b0;
                                end

        end

        // 4;b0001 normal write to state holder **ping_pong(pp_selc)**
              4'h0001 : begin
            // *****Change the sub_comp code to match the input
                if (sub_comp == sub_comp_ID) begin
                                        if (child_comp < child_limit) begin
                            case (input_type)
                               3'b001: begin
                                  // visible
```

```
                                ping_pong_stateholder[pp_selc][child_comp][31] = input_msg[12];
                                // fliped
                                ping_pong_stateholder[pp_selc][child_comp][30] = input_msg[11];
                                // pattern code
                                if (input_msg[4:0] < pattern_num)
                                    ping_pong_stateholder[pp_selc][child_comp][111:32] =
pattern_table[input_msg[4:0]];
                            end
                            3'b010: begin
                                // x_coordinate
                                ping_pong_stateholder[pp_selc][child_comp][29:20] =
input_msg[9:0];
                            end
                            3'b011: begin
                                // y_coordinate
                                ping_pong_stateholder[pp_selc][child_comp][19:10] =
input_msg[9:0];
                            end
                            3'b100: begin
                                // shift_amount
                                ping_pong_stateholder[pp_selc][child_comp][9:0] =
input_msg[9:0];
                            end
                        endcase
                                    end
                    end
        end
    endcase
        end


    // Output ==================================

        always_comb begin
                for (j = 0; j < child_limit; j = j + 1) begin
                        ping_pong_RGB_output[0][j] =  (ping_pong_addr_output[0][j] <
addr_limit)?
                        (
                                (ping_pong_addr_output[0][j][0])?
                                        color_plate[mem[(ping_pong_addr_output[0][j][15:1])][3:2]]
:
                                        color_plate[mem[(ping_pong_addr_output[0][j][15:1])][1:0]]
                        ) :
                        color_plate[mem[0]];
```

```
                        ping_pong_RGB_output[1][j] =  (ping_pong_addr_output[1][j] <
addr_limit)?
                        (
                                (ping_pong_addr_output[1][j][0])?
                                        color_plate[mem[(ping_pong_addr_output[1][j][15:1])][3:2]]
:
                                        color_plate[mem[(ping_pong_addr_output[1][j][15:1])][1:0]]
                        ) :
                        color_plate[mem[0]];
                end

                RGB_output = 24'h9290ff;
                for (k = 0; k < child_limit; k = k + 1) begin
                        if ((ping_pong_RGB_output[ping_pong][k] != 24'h9290ff) &&
ping_pong_addr_out_valid[ping_pong][k]) begin
                                RGB_output = ping_pong_RGB_output[ping_pong][k];
                                break;
                        end
                end
        end

initial begin

$readmemh("/user/stud/fall22/hy2759/4840/pro_test/lab3-hw/on_chip_mem/Mush_2bit.txt",
mem);
end


endmodule
```

Soc_system_top.sv:

```
// ====================================================================
// Copyright (c) 2013 by Terasic Technologies Inc.
// ====================================================================
//
// Modified 2019 by Stephen A. Edwards
//
// Permission:
//
```

```
//   Terasic grants permission to use and modify this code for use
//   in synthesis for all Terasic Development Boards and Altera
//   Development Kits made by Terasic.  Other use of this code,
//   including the selling ,duplication, or modification of any
//   portion is strictly prohibited.
//
// Disclaimer:
//
//   This VHDL/Verilog or C/C++ source code is intended as a design
//   reference which illustrates how these types of functions can be
//   implemented.  It is the user's responsibility to verify their
//   design for consistency and functionality through the use of
//   formal verification methods.  Terasic provides no warranty
//   regarding the use or functionality of this code.
//
// ============================================================
//
//  Terasic Technologies Inc

//  9F., No.176, Sec.2, Gongdao 5th Rd, East Dist, Hsinchu City, 30070. Taiwan
//
//
//                     web: http://www.terasic.com/
//                     email: support@terasic.com
module soc_system_top(

 ///////// ADC /////////
 inout        ADC_CS_N,
 output       ADC_DIN,
 input        ADC_DOUT,
 output       ADC_SCLK,

 ///////// AUD /////////
 input        AUD_ADCDAT,
 inout        AUD_ADCLRCK,
 inout        AUD_BCLK,
 output       AUD_DACDAT,
 inout        AUD_DACLRCK,
 output       AUD_XCK,

 ///////// CLOCK2 /////////
 input        CLOCK2_50,

 ///////// CLOCK3 /////////
```

```
input         CLOCK3_50,

////////// CLOCK4 //////////
input         CLOCK4_50,

////////// CLOCK //////////
input         CLOCK_50,

////////// DRAM //////////
output [12:0] DRAM_ADDR,
output [1:0]  DRAM_BA,
output        DRAM_CAS_N,
output        DRAM_CKE,
output        DRAM_CLK,
output        DRAM_CS_N,
inout [15:0]  DRAM_DQ,
output        DRAM_LDQM,
output        DRAM_RAS_N,
output        DRAM_UDQM,
output        DRAM_WE_N,

////////// FAN //////////
output        FAN_CTRL,

////////// FPGA //////////
output        FPGA_I2C_SCLK,
inout         FPGA_I2C_SDAT,

////////// GPIO //////////
inout [35:0]  GPIO_0,
inout [35:0]  GPIO_1,

////////// HEX0 //////////
output [6:0]  HEX0,

////////// HEX1 //////////
output [6:0]  HEX1,

////////// HEX2 //////////
output [6:0]  HEX2,

////////// HEX3 //////////
output [6:0]  HEX3,
```

```
//////// HEX4 ////////
output [6:0]  HEX4,

//////// HEX5 ////////
output [6:0]  HEX5,

//////// HPS ////////
inout         HPS_CONV_USB_N,
output [14:0] HPS_DDR3_ADDR,
output [2:0]  HPS_DDR3_BA,
output        HPS_DDR3_CAS_N,
output        HPS_DDR3_CKE,
output        HPS_DDR3_CK_N,
output        HPS_DDR3_CK_P,
output        HPS_DDR3_CS_N,
output [3:0]  HPS_DDR3_DM,
inout [31:0]  HPS_DDR3_DQ,
inout [3:0]   HPS_DDR3_DQS_N,
inout [3:0]   HPS_DDR3_DQS_P,
output        HPS_DDR3_ODT,
output        HPS_DDR3_RAS_N,
output        HPS_DDR3_RESET_N,
input         HPS_DDR3_RZQ,
output        HPS_DDR3_WE_N,
output        HPS_ENET_GTX_CLK,
inout         HPS_ENET_INT_N,
output        HPS_ENET_MDC,
inout         HPS_ENET_MDIO,
input         HPS_ENET_RX_CLK,
input [3:0]   HPS_ENET_RX_DATA,
input         HPS_ENET_RX_DV,
output [3:0]  HPS_ENET_TX_DATA,
output        HPS_ENET_TX_EN,
inout         HPS_GSENSOR_INT,
inout         HPS_I2C1_SCLK,
inout         HPS_I2C1_SDAT,
inout         HPS_I2C2_SCLK,
inout         HPS_I2C2_SDAT,
inout         HPS_I2C_CONTROL,
inout         HPS_KEY,
inout         HPS_LED,
inout         HPS_LTC_GPIO,
output        HPS_SD_CLK,
inout         HPS_SD_CMD,
```

```
inout [3:0]   HPS_SD_DATA,
output        HPS_SPIM_CLK,
input         HPS_SPIM_MISO,
output        HPS_SPIM_MOSI,
inout         HPS_SPIM_SS,
input         HPS_UART_RX,
output        HPS_UART_TX,
input         HPS_USB_CLKOUT,
inout [7:0]   HPS_USB_DATA,
input         HPS_USB_DIR,
input         HPS_USB_NXT,
output        HPS_USB_STP,

///////// IRDA /////////
input         IRDA_RXD,
output        IRDA_TXD,

///////// KEY /////////
input [3:0]   KEY,

///////// LEDR /////////
output [9:0]  LEDR,

///////// PS2 /////////
inout         PS2_CLK,
inout         PS2_CLK2,
inout         PS2_DAT,
inout         PS2_DAT2,

///////// SW /////////
input [9:0]   SW,

///////// TD /////////
input         TD_CLK27,
input [7:0]   TD_DATA,
input         TD_HS,
output        TD_RESET_N,
input         TD_VS,


///////// VGA /////////
output [7:0]  VGA_B,
output        VGA_BLANK_N,
output        VGA_CLK,
```

```
 output [7:0]  VGA_G,
 output        VGA_HS,
 output [7:0]  VGA_R,
 output        VGA_SYNC_N,
 output        VGA_VS
);

   soc_system soc_system0(
     .clk_clk                       ( CLOCK_50 ),
     .reset_reset_n                 ( 1'b1 ),

     .hps_ddr3_mem_a                ( HPS_DDR3_ADDR ),
     .hps_ddr3_mem_ba               ( HPS_DDR3_BA ),
     .hps_ddr3_mem_ck               ( HPS_DDR3_CK_P ),
     .hps_ddr3_mem_ck_n             ( HPS_DDR3_CK_N ),
     .hps_ddr3_mem_cke              ( HPS_DDR3_CKE ),
     .hps_ddr3_mem_cs_n             ( HPS_DDR3_CS_N ),
     .hps_ddr3_mem_ras_n            ( HPS_DDR3_RAS_N ),
     .hps_ddr3_mem_cas_n            ( HPS_DDR3_CAS_N ),
     .hps_ddr3_mem_we_n             ( HPS_DDR3_WE_N ),
     .hps_ddr3_mem_reset_n          ( HPS_DDR3_RESET_N ),
     .hps_ddr3_mem_dq               ( HPS_DDR3_DQ ),
     .hps_ddr3_mem_dqs              ( HPS_DDR3_DQS_P ),
     .hps_ddr3_mem_dqs_n            ( HPS_DDR3_DQS_N ),
     .hps_ddr3_mem_odt              ( HPS_DDR3_ODT ),
     .hps_ddr3_mem_dm               ( HPS_DDR3_DM ),
     .hps_ddr3_oct_rzqin            ( HPS_DDR3_RZQ ),

     .hps_hps_io_emac1_inst_TX_CLK ( HPS_ENET_GTX_CLK ),
     .hps_hps_io_emac1_inst_TXD0   ( HPS_ENET_TX_DATA[0] ),
     .hps_hps_io_emac1_inst_TXD1   ( HPS_ENET_TX_DATA[1] ),
     .hps_hps_io_emac1_inst_TXD2   ( HPS_ENET_TX_DATA[2] ),
     .hps_hps_io_emac1_inst_TXD3   ( HPS_ENET_TX_DATA[3] ),
     .hps_hps_io_emac1_inst_RXD0   ( HPS_ENET_RX_DATA[0] ),
     .hps_hps_io_emac1_inst_MDIO   ( HPS_ENET_MDIO  ),
     .hps_hps_io_emac1_inst_MDC    ( HPS_ENET_MDC   ),
     .hps_hps_io_emac1_inst_RX_CTL ( HPS_ENET_RX_DV ),
     .hps_hps_io_emac1_inst_TX_CTL ( HPS_ENET_TX_EN ),
     .hps_hps_io_emac1_inst_RX_CLK ( HPS_ENET_RX_CLK ),
     .hps_hps_io_emac1_inst_RXD1   ( HPS_ENET_RX_DATA[1]  ),
     .hps_hps_io_emac1_inst_RXD2   ( HPS_ENET_RX_DATA[2]  ),
     .hps_hps_io_emac1_inst_RXD3   ( HPS_ENET_RX_DATA[3]  ),

     .hps_hps_io_sdio_inst_CMD     ( HPS_SD_CMD         ),
```

```
.hps_hps_io_sdio_inst_D0      ( HPS_SD_DATA[0]      ),
.hps_hps_io_sdio_inst_D1      ( HPS_SD_DATA[1]      ),
.hps_hps_io_sdio_inst_CLK     ( HPS_SD_CLK          ),
.hps_hps_io_sdio_inst_D2      ( HPS_SD_DATA[2]      ),
.hps_hps_io_sdio_inst_D3      ( HPS_SD_DATA[3]      ),

.hps_hps_io_usb1_inst_D0      ( HPS_USB_DATA[0]     ),
.hps_hps_io_usb1_inst_D1      ( HPS_USB_DATA[1]     ),
.hps_hps_io_usb1_inst_D2      ( HPS_USB_DATA[2]     ),
.hps_hps_io_usb1_inst_D3      ( HPS_USB_DATA[3]     ),
.hps_hps_io_usb1_inst_D4      ( HPS_USB_DATA[4]     ),
.hps_hps_io_usb1_inst_D5      ( HPS_USB_DATA[5]     ),
.hps_hps_io_usb1_inst_D6      ( HPS_USB_DATA[6]     ),
.hps_hps_io_usb1_inst_D7      ( HPS_USB_DATA[7]     ),
.hps_hps_io_usb1_inst_CLK     ( HPS_USB_CLKOUT      ),
.hps_hps_io_usb1_inst_STP     ( HPS_USB_STP         ),
.hps_hps_io_usb1_inst_DIR     ( HPS_USB_DIR         ),
.hps_hps_io_usb1_inst_NXT     ( HPS_USB_NXT         ),

.hps_hps_io_spim1_inst_CLK    ( HPS_SPIM_CLK  ),
.hps_hps_io_spim1_inst_MOSI   ( HPS_SPIM_MOSI ),
.hps_hps_io_spim1_inst_MISO   ( HPS_SPIM_MISO ),
.hps_hps_io_spim1_inst_SS0    ( HPS_SPIM_SS   ),

.hps_hps_io_uart0_inst_RX     ( HPS_UART_RX     ),
.hps_hps_io_uart0_inst_TX     ( HPS_UART_TX     ),

.hps_hps_io_i2c0_inst_SDA     ( HPS_I2C1_SDAT     ),
.hps_hps_io_i2c0_inst_SCL     ( HPS_I2C1_SCLK     ),

.hps_hps_io_i2c1_inst_SDA     ( HPS_I2C2_SDAT     ),
.hps_hps_io_i2c1_inst_SCL     ( HPS_I2C2_SCLK     ),

.hps_hps_io_gpio_inst_GPIO09  ( HPS_CONV_USB_N ),
.hps_hps_io_gpio_inst_GPIO35  ( HPS_ENET_INT_N ),
.hps_hps_io_gpio_inst_GPIO40  ( HPS_LTC_GPIO ),

.hps_hps_io_gpio_inst_GPIO48  ( HPS_I2C_CONTROL ),
.hps_hps_io_gpio_inst_GPIO53  ( HPS_LED ),
.hps_hps_io_gpio_inst_GPIO54  ( HPS_KEY ),
.hps_hps_io_gpio_inst_GPIO61  ( HPS_GSENSOR_INT ),
                .vga_r (VGA_R),
                .vga_g (VGA_G),
                .vga_b (VGA_B),
```

```
                .vga_clk (VGA_CLK),
                .vga_hs (VGA_HS),
                .vga_vs (VGA_VS),
                .vga_blank_n (VGA_BLANK_N),
                .vga_sync_n (VGA_SYNC_N),
    .audio_0_external_interface_BCLK                (AUD_BCLK),
    .audio_0_external_interface_DACDAT              (AUD_DACDAT),
    .audio_0_external_interface_DACLRCK             (AUD_DACLRCK),
    .audio_pll_0_audio_clk_clk                      (AUD_XCK),
    .audio_and_video_config_0_external_interface_SDAT (FPGA_I2C_SDAT),
    .audio_and_video_config_0_external_interface_SCLK (FPGA_I2C_SCLK)
  );

    // The following quiet the "no driver" warnings for output
    // pins and should be removed if you use any of these peripherals

    assign ADC_CS_N = SW[1] ? SW[0] : 1'bZ;
    assign ADC_DIN = SW[0];
    assign ADC_SCLK = SW[0];

   // assign AUD_ADCLRCK = SW[1] ? SW[0] : 1'bZ;
   // assign AUD_BCLK = SW[1] ? SW[0] : 1'bZ;
   // assign AUD_DACDAT = SW[0];
   // assign AUD_DACLRCK = SW[1] ? SW[0] : 1'bZ;
   // assign AUD_XCK = SW[0];

    assign DRAM_ADDR = { 13{ SW[0] } };
    assign DRAM_BA = { 2{ SW[0] } };
    assign DRAM_DQ = SW[1] ? { 16{ SW[0] } } : 16'bZ;
    assign {DRAM_CAS_N, DRAM_CKE, DRAM_CLK, DRAM_CS_N,
            DRAM_LDQM, DRAM_RAS_N, DRAM_UDQM, DRAM_WE_N} = { 8{SW[0]} };

    assign FAN_CTRL = SW[0];

//    assign FPGA_I2C_SCLK = SW[0];
//    assign FPGA_I2C_SDAT = SW[1] ? SW[0] : 1'bZ;

    assign GPIO_0 = SW[1] ? { 36{ SW[0] } } : 36'bZ;
    assign GPIO_1 = SW[1] ? { 36{ SW[0] } } : 36'bZ;

    assign HEX0 = { 7{ SW[1] } };
    assign HEX1 = { 7{ SW[2] } };
    assign HEX2 = { 7{ SW[3] } };
    assign HEX3 = { 7{ SW[4] } };
```

```
   assign HEX4 = { 7{ SW[5] } };
   assign HEX5 = { 7{ SW[6] } };

//   assign IRDA_TXD = SW[0];

   assign LEDR = { 10{SW[7]} };

   assign PS2_CLK = SW[1] ? SW[0] : 1'bZ;
   assign PS2_CLK2 = SW[1] ? SW[0] : 1'bZ;
   assign PS2_DAT = SW[1] ? SW[0] : 1'bZ;
   assign PS2_DAT2 = SW[1] ? SW[0] : 1'bZ;

//   assign TD_RESET_N = SW[0];

   // assign {VGA_R, VGA_G, VGA_B} = { 24{ SW[0] } };
   // assign {VGA_BLANK_N, VGA_CLK,
         //    VGA_HS, VGA_SYNC_N, VGA_VS} = { 5{ SW[0] } };


endmodule
```

Tube_display.sv:

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module Tube_display (input logic        clk,
               input logic       reset,
                     input logic [31:0]  writedata,
                     input logic [9:0]   hcount,
                     input logic [9:0]   vcount,

                     output logic [23:0]       RGB_output);

    // Change for other type=========================
    parameter [5:0] sub_comp_ID = 6'b001010; // #10
    parameter [4:0] pattern_num = 5'd_2;
    parameter [15:0] addr_limit = 16'd_576;  // 768
          parameter [4:0] child_limit = 5'd_2;
```

```
        logic [3:0] mem [0:287];
        logic [23:0] color_plate [0:3];
logic [79:0] pattern_table [0:1];

assign pattern_table[0] = {16'd_0, 16'd_32, 16'd_16, 16'd_32, 16'd_16};
        assign pattern_table[1] = {16'd_544, 16'd_32, 16'd_1, 16'd_32, 16'd_128};

        assign color_plate[0] = 24'h9290ff;
        assign color_plate[1] = 24'h000000;
        assign color_plate[2] = 24'h8cd600;
        assign color_plate[3] = 24'h109400;

//================================================

// logic [79:0] ping_pong_pattern_input[0:1];
logic [23:0] ping_pong_RGB_output[0:1][0:1];
logic [15:0] ping_pong_addr_output[0:1][0:1];
logic        ping_pong_addr_out_valid[0:1][0:1];
logic [111:0] ping_pong_stateholder[0:1][0:1];
logic        ping_pong = 1'b0;

        logic [5:0] sub_comp;
        logic [4:0] child_comp;
        logic [3:0] info;
        logic [2:0] input_type;
        logic [12:0] input_msg;
        logic               pp_selc;

        assign sub_comp = writedata[31:26];
        assign child_comp = writedata[25:21];
        assign info = writedata[20:17];
        assign input_type = writedata[16:14];
        assign pp_selc = writedata[13];
        assign input_msg = writedata[12:0];

        // loop ============================
        integer i, j, k;
        // ================================

//================================================
// Address_calculater change to adapt different setting
//================================================
```

```
        addr_cal AC_ping_0(.pattern_info(ping_pong_stateholder[0][0][111:32]),
.sprite_info(ping_pong_stateholder[0][0][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][0]), .valid(ping_pong_addr_out_valid[0][0]));
   addr_cal AC_ping_1(.pattern_info(ping_pong_stateholder[0][1][111:32]),
.sprite_info(ping_pong_stateholder[0][1][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[0][1]), .valid(ping_pong_addr_out_valid[0][1]));

   addr_cal AC_pong_0(.pattern_info(ping_pong_stateholder[1][0][111:32]),
.sprite_info(ping_pong_stateholder[1][0][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][0]), .valid(ping_pong_addr_out_valid[1][0]));
   addr_cal AC_pong_1(.pattern_info(ping_pong_stateholder[1][1][111:32]),
.sprite_info(ping_pong_stateholder[1][1][31:0]), .hcount(hcount), .vcount(vcount),
.addr_output(ping_pong_addr_output[1][1]), .valid(ping_pong_addr_out_valid[1][1]));


   // FF Input================================
        always_ff @(posedge clk) begin
      case (info)
        // 4'b1111 to flush *PING(0)/PONG(1)* buffer and clear *THE OPPOSIT* buffer
        4'b1111: begin
           ping_pong = pp_selc;
                                for (i = 0; i < child_limit; i = i + 1) begin
                                        ping_pong_stateholder[~pp_selc][i][31] = 1'b0;
                                end

        end

        // 4;b0001 normal write to state holder **ping_pong(pp_selc)**
             4'h0001 : begin
           // *****Change the sub_comp code to match the input
                if (sub_comp == sub_comp_ID) begin
                                        if (child_comp < child_limit) begin
                        case (input_type)
                           3'b001: begin
                              // visible
                              ping_pong_stateholder[pp_selc][child_comp][31] = input_msg[12];
                              // fliped
                              ping_pong_stateholder[pp_selc][child_comp][30] = input_msg[11];
                              // pattern code
                              if (input_msg[4:0] < pattern_num)
                                 ping_pong_stateholder[pp_selc][child_comp][111:32] =
pattern_table[input_msg[4:0]];
                                end
                                3'b010: begin
```

```
                                // x_coordinate
                                ping_pong_stateholder[pp_selc][child_comp][29:20] =
input_msg[9:0];
                            end
                            3'b011: begin
                                // y_coordinate
                                ping_pong_stateholder[pp_selc][child_comp][19:10] =
input_msg[9:0];
                            end
                            3'b100: begin
                                // shift_amount
                                ping_pong_stateholder[pp_selc][child_comp][9:0] =
input_msg[9:0];
                            end
                        endcase
                                end
                    end
            end
        endcase
            end


        // Output ====================================

            always_comb begin
                for (j = 0; j < child_limit; j = j + 1) begin
                    ping_pong_RGB_output[0][j] =  (ping_pong_addr_output[0][j] <
addr_limit)?
                    (
                        (ping_pong_addr_output[0][j][0])?
                            color_plate[mem[(ping_pong_addr_output[0][j][15:1])][3:2]]
:
                            color_plate[mem[(ping_pong_addr_output[0][j][15:1])][1:0]]
                    ) :
                    color_plate[mem[0]];

                    ping_pong_RGB_output[1][j] =  (ping_pong_addr_output[1][j] <
addr_limit)?
                    (
                        (ping_pong_addr_output[1][j][0])?
                            color_plate[mem[(ping_pong_addr_output[1][j][15:1])][3:2]]
:
                            color_plate[mem[(ping_pong_addr_output[1][j][15:1])][1:0]]
                    ) :
```

```
                        color_plate[mem[0]];
                end

                RGB_output = 24'h9290ff;
                for (k = 0; k < child_limit; k = k + 1) begin
                        if ((ping_pong_RGB_output[ping_pong][k] != 24'h9290ff) &&
ping_pong_addr_out_valid[ping_pong][k]) begin
                                RGB_output = ping_pong_RGB_output[ping_pong][k];
                                break;
                        end
                end
        end

initial begin

$readmemh("/user/stud/fall22/hy2759/4840/pro_test/lab3-hw/on_chip_mem/Tube_2bit.txt",
mem);
end


endmodule
```

Vga_ball.sv:

```
/*
 * Avalon memory-mapped peripheral that generates VGA
 *
 * Stephen A. Edwards
 * Columbia University
 */

module vga_ball(input logic        clk,
                input logic              reset,
                   input logic [31:0]  writedata,
                   input logic          write,
                   input                chipselect,
                   input logic [2:0]  address,

                   input left_chan_ready,
                   input right_chan_ready,

                   output logic [15:0] sample_data_l,
                   output logic sample_valid_l,
                   output logic [15:0] sample_data_r,
```

```
                output logic sample_valid_r,

                output logic [7:0] VGA_R, VGA_G, VGA_B,
                output logic       VGA_CLK, VGA_HS, VGA_VS,
                                   VGA_BLANK_n,
                output logic       VGA_SYNC_n);

  logic [10:0]     hcount;
  logic [9:0]    vcount;
  logic [31:0] ppu_info,sound_buff;

  vga_counters counters(.clk50(clk), .*);

  always_ff @(posedge clk)
    if (reset) begin
        ppu_info <= 32'd_0;
        sound_buff <= 32'd_0;
    end else if (chipselect && write)
      case (address)
         3'h0 : ppu_info <= writedata;
         3'h1 : sound_buff <= writedata;// sound

      endcase

//----------------Sound----------------
        reg [13:0] counter;
        logic flag1;
        logic flag2;
        logic flag3;
        logic flag4;

        reg      [9:0]  address1;
        wire [15:0]  q1;
        smb_breakblock_ROM audio1(.address(address1), .clock(clk), .q(q1));//12877

        reg      [9:0]  address2;
        wire [15:0]  q2;
        smb_jump_ROM  audio2(.address(address2), .clock(clk), .q(q2));//12066

        reg      [9:0]  address3;
        wire [15:0]  q3;
        //smb_gameover_ROM audio3(.address(address3), .clock(clk), .q(q3));//59806
        smb_gameover_ROM audio3(.address(address3), .clock(clk), .q(q3));//59806
        reg [9:0] address4;
```

```
	wire [15:0] q4;
	smb_coin_ROM audio4(.address(address4), .clock(clk), .q(q4));//20848

	always_ff @(posedge clk) begin
		if(reset) begin
			counter <= 0;
			sample_valid_l <= 0; sample_valid_r <= 0;
		end
		else if(left_chan_ready == 1 && right_chan_ready == 1 && counter < 6250) begin
			counter <= counter + 1;
			sample_valid_l <= 0; sample_valid_r <= 0;
		end
		else if(left_chan_ready == 1 && right_chan_ready == 1 && counter == 6250)
begin
			counter <= 0;
			sample_valid_l <= 1; sample_valid_r <= 1;
			//------Game over------
			if (sound_buff[2:0]==3'd3 || flag3 ==1'b0) begin
				if (address3 < 10'd1000) begin   // 59806
					address3 <= address3+1;
					flag3 <= 1'b0;
				end
				else begin
					address3 <=0;
					flag3 <= 1'b1;
				end
				sample_data_l <= q3;
				sample_data_r <= q3;
			end
			//------Break Block------
			else if (sound_buff[2:0]==3'd1 || flag1 ==1'b0) begin
				if (address1 < 10'd1000) begin //12066
					address1 <= address1+1;
					flag1 <= 1'b0;
				end
				else begin
					address1 <=0;
					flag1 <= 1'b1;
				end
				sample_data_l <= q1;
				sample_data_r <= q1;
			end
			//------Jump Surper------
			else if (sound_buff[2:0]==3'd2 || flag2 ==1'b0) begin
```

```
                if (address2 < 10'd1000) begin //12877
                        address2 <= address2+1;
                        flag2 <= 1'b0;
                end
                else begin
                        address2 <=0;
                        flag2 <= 1'b1;
                end
                sample_data_l <= q2;
                sample_data_r <= q2;
        end
        //------coin------
        else if (sound_buff[2:0]==3'd4 || flag4 == 1'b0) begin
                if (address4 < 10'd1000) begin   //20848
                        address4 <= address4 + 1;
                        flag4 <= 1'b0;
                end else begin
                        address4 <= 0;
                        flag4 <= 1'b1;
                end
                sample_data_l <= q4;
                sample_data_r <= q4;
        end
        else if (sound_buff[2:0]==3'd5 ) begin
                        address1 <= 0;
                        address2 <= 0;
                        address3 <= 0;
                        address4 <= 0;
                        flag1 <= 1'b0;
                        flag2 <= 1'b0;
                        flag3 <= 1'b0;
                        flag4 <= 1'b0;

                sample_data_l <= 0;
                sample_data_r <= 0;
        end


        else begin
                sample_data_l <= 0;
                sample_data_r <= 0;
        end
end
else begin
```

```
                sample_valid_l <= 0; sample_valid_r <= 0;
                end
        end


  //test =========
  logic [23:0] PPU_out;
  ppu game_ppu(.clk(clk), .reset(reset), .writedata(ppu_info), .address(3'd_0),
.hcount(hcount[10:1]),
                .vcount(vcount), .RGB_output(PPU_out));
  //==============

  always_comb begin
    {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
    if (VGA_BLANK_n ) begin


    end

        if (hcount < 11'd_1120 && hcount >= 11'd_160 && vcount < 10'd_400)
                        {VGA_R, VGA_G, VGA_B} = PPU_out;
        else
                        {VGA_R, VGA_G, VGA_B} =
                                {8'h00, 8'h00, 8'h00};

  end

endmodule

module vga_counters(
 input logic          clk50, reset,
 output logic [10:0] hcount,  // hcount[10:1] is pixel column
 output logic [9:0]  vcount,  // vcount[9:0] is pixel row
 output logic        VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 * HCOUNT 1599 0            1279       1599 0
 *           _________________           ________
 * ______________|    Video      |_____________|  Video
 *
 *
 * |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
```

```
*      _____________________      _____________
* |____|       VGA_HS          |____|
*/
  // Parameters for hcount
  parameter HACTIVE      = 11'd 1280,
            HFRONT_PORCH = 11'd 32,
            HSYNC        = 11'd 192,
            HBACK_PORCH  = 11'd 96,
            HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                           HBACK_PORCH; // 1600

  // Parameters for vcount
  parameter VACTIVE      = 10'd 480,
            VFRONT_PORCH = 10'd 10,
            VSYNC        = 10'd 2,
            VBACK_PORCH  = 10'd 33,
            VTOTAL       = VACTIVE + VFRONT_PORCH + VSYNC +
                           VBACK_PORCH; // 525

  logic endOfLine;

  always_ff @(posedge clk50 or posedge reset)
    if (reset)          hcount <= 0;
    else if (endOfLine) hcount <= 0;
    else                hcount <= hcount + 11'd 1;

  assign endOfLine = hcount == HTOTAL - 1;

  logic endOfField;

  always_ff @(posedge clk50 or posedge reset)
    if (reset)          vcount <= 0;
    else if (endOfLine)
      if (endOfField)   vcount <= 0;
      else              vcount <= vcount + 10'd 1;

  assign endOfField = vcount == VTOTAL - 1;

  // Horizontal sync: from 0x520 to 0x5DF (0x57F)
  // 101 0010 0000 to 101 1101 1111
  assign VGA_HS = !( (hcount[10:8] == 3'b101) &
                     !(hcount[7:5] == 3'b111));
  assign VGA_VS = !( vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);
```

```
  assign VGA_SYNC_n = 1'b0; // For putting sync on the green signal; unused

  // Horizontal active: 0 to 1279     Vertical active: 0 to 479
  // 101 0000 0000  1280              01 1110 0000  480
  // 110 0011 1111  1599              10 0000 1100  524
  assign VGA_BLANK_n = !( hcount[10] & (hcount[9] | hcount[8]) ) &
                       !( vcount[9] | (vcount[8:5] == 4'b1111) );

  /* VGA_CLK is 25 MHz
   *             __    __    __
   * clk50    __|  |__|  |__|
   *
   *           _____       __
   * hcount[0]__|     |_____|
   */
  assign VGA_CLK = hcount[0]; // 25 MHz clock: rising edge sensitive

endmodule
```

Software :

game_animation.c:

```
#include "game_animation.h"
#include "game_struct.h"


void mario_animation(mario_game *game_0, mario* mario_0, int f_counter){
        int counter, ani_div, rel_counter;
        float dead_acc, dead_v;
        dead_v = -4.7; dead_acc = 0.16;
        counter = (f_counter >= mario_0->animate_frame_counter)? f_counter : f_counter +
FRAME_LIMIT;
        rel_counter = counter - mario_0->animate_frame_counter;
        if (mario_0->control_s == MARIO_NORMAL) {
                if (mario_0->y_block == 1){
                        if (mario_0->acc_x == 0) mario_0->vis.pattern_code = (mario_0->mario_s
== SMALL)? ANI_MARIO_S_NORMAL : ANI_MARIO_L_NORMAL;
                        else if ((mario_0->acc_x) * (mario_0->hit.vx) < 0)
                                mario_0->vis.pattern_code = (mario_0->mario_s == SMALL)?
ANI_MARIO_S_SHUT : ANI_MARIO_L_SHUT;
```

```
                        else {
                                ani_div = (counter/6)%3;
                                switch (ani_div){
                                        case 0:
                                                mario_0->vis.pattern_code = (mario_0->mario_s ==
SMALL)? ANI_MARIO_S_WALK1 : ANI_MARIO_L_WALK1;
                                                break;
                                        case 1:
                                                mario_0->vis.pattern_code = (mario_0->mario_s ==
SMALL)? ANI_MARIO_S_WALK2 : ANI_MARIO_L_WALK2;
                                                break;
                                        case 2:
                                                mario_0->vis.pattern_code = (mario_0->mario_s ==
SMALL)? ANI_MARIO_S_WALK3 : ANI_MARIO_L_WALK3;
                                                break;
                                        default:
                                                mario_0->vis.pattern_code = (mario_0->mario_s ==
SMALL)? ANI_MARIO_S_NORMAL : ANI_MARIO_L_NORMAL;
                                }
                        }
                }
                else
                        mario_0->vis.pattern_code = (mario_0->mario_s == SMALL)?
ANI_MARIO_S_JUMP : ANI_MARIO_L_JUMP;
        }
        else{
                switch (mario_0->animate_s){
                        case HIT:
                                ani_div = (rel_counter/3)%2;
                                if(rel_counter == 0) mario_0->hit.y += 3;
                                else if (rel_counter <= 20) mario_0->vis.pattern_code = (ani_div
== 0)? ANI_MARIO_L_HIT : ANI_MARIO_S_HIT;
                                else {mario_0->control_s = MARIO_NORMAL; mario_0->mario_s
= SMALL; mario_0->hit.sy = 16; mario_0->game_s = MARIO_GMAE_HIT;}
                                break;
                        case ENLARGE:
                                ani_div = (rel_counter/3)%2;
                                if(rel_counter == 0) mario_0->hit.y -= 20;
                                else if (rel_counter <= 15) mario_0->vis.pattern_code = (ani_div
== 0)? ANI_MARIO_S_NORMAL : ANI_MARIO_M_NORMAL;
                                else if (rel_counter <= 30) mario_0->vis.pattern_code = (ani_div
== 0)? ANI_MARIO_L_NORMAL : ANI_MARIO_M_NORMAL;
                                else {mario_0->control_s = MARIO_NORMAL; mario_0->mario_s
= LARGE; mario_0->hit.sy = 32;}
```

```
                        break;
                case ANI_DEAD:
                        mario_0->vis.pattern_code =ANI_MARIO_S_DEAD;
                        if(rel_counter == 0) {mario_0->hit.vy = dead_v;}
                        else if(rel_counter >= 30){
                                mario_0->hit.y += mario_0->hit.vy; mario_0->hit.vy +=
dead_acc;
                                if (mario_0->hit.y > 500) mario_0->mario_s = DEAD;
                        }
                        break;
                default:
                        break;

        }
    }

    if (rel_counter > 120 && mario_0->game_s == MARIO_GMAE_HIT) {mario_0->game_s
= MARIO_GAME_NORMAL;}

    mario_0->vis.visible = (mario_0->game_s == MARIO_GAME_NORMAL)? 1 :
((counter/8)%2 == 0)? 1 : 0;
    // game_0.mario_0.vis.flip = 0;
    // game_0.mario_0.vis.pattern_code = 0;
    mario_0->vis.x = mario_0->hit.x - game_0->camera_pos;
    mario_0->vis.y = mario_0->hit.y;
    return;
}
void mush_animation(mario_game *game_0, mush* mush_0, int f_counter){
    int counter, ani_div, rel_counter;
    counter = (f_counter >= mush_0->animate_frame_counter)? f_counter : f_counter +
FRAME_LIMIT;
    rel_counter = counter - mush_0->animate_frame_counter;
    if (mush_0->mush_s == MUSH_NORMAL) {
        mush_0->vis.pattern_code = ANI_MUSH_NORMAL;
    }
    else{
        mush_0->vis.pattern_code = ANI_MUSH_NORMAL;
        if (rel_counter < 40) mush_0->hit.y -= 0.5;
        else mush_0->mush_s = MUSH_NORMAL;
    }

    // mush_0->vis.visible = 1;
    mush_0->vis.x = mush_0->hit.x - game_0->camera_pos;
    mush_0->vis.y = mush_0->hit.y;
```

```
	return;
}
void goomba_animation(mario_game *game_0, goomba* goomba_0, int f_counter){
	int counter, ani_div, rel_counter;
	counter = (f_counter >= goomba_0->animate_frame_counter)? f_counter : f_counter +
FRAME_LIMIT;
	rel_counter = counter - goomba_0->animate_frame_counter;

	if (goomba_0->goomba_s == GOOMBA_NORMAL){
		ani_div = (counter/7)%2;
		goomba_0->vis.pattern_code = ANI_GOOMBA_NORMAL;
		goomba_0->vis.flip = ani_div;
	}

	else{
		if (rel_counter == 0){ goomba_0->hit.y += 8;}
		else if (rel_counter >= 25){ goomba_0->enable = 0;}
		goomba_0->vis.pattern_code = ANI_GOOMBA_HIT;
	}

	goomba_0->vis.x = goomba_0->hit.x - game_0->camera_pos;
	goomba_0->vis.y = goomba_0->hit.y;
}
void tube_animation(mario_game *game_0, tube* tube_0, int f_counter){
	int counter, ani_div, rel_counter;
	tube_0->vis.visible = 1;
	tube_0->vis.x = tube_0->hit.x - game_0->camera_pos;
	tube_0->vis.y = tube_0->hit.y;
}
void block_animation(mario_game *game_0, block* block_0, int f_counter){
	int counter, ani_div, rel_counter;
	counter = (f_counter >= block_0->animate_frame_counter)? f_counter : f_counter +
FRAME_LIMIT;
	rel_counter = counter - block_0->animate_frame_counter;
	switch (block_0->block_t){
		case TYPE_A:
			if (block_0->block_s == BLOCK_NORMAL) {
				block_0->vis.pattern_code = ANI_BLOCK_A1;
			}
			else{
				if (rel_counter <= 8) block_0->hit.y += (rel_counter - 8) * 0.3;
				else if (rel_counter <= 12) block_0->vis.pattern_code =
ANI_BLOCK_A_HIT;
				else block_0->enable = 0;
```

```
                }
                break;
        case TYPE_B_1:
                block_0->vis.pattern_code = ANI_BLOCK_B;
                break;
        case TYPE_B_2:
                block_0->vis.pattern_code = ANI_BLOCK_B_V2;
                break;
        case TYPE_B_3:
                block_0->vis.pattern_code = ANI_BLOCK_B_V3;
                break;
        case TYPE_B_4:
                block_0->vis.pattern_code = ANI_BLOCK_B_V4;
                break;
        case TYPE_B_16:
                block_0->vis.pattern_code = ANI_BLOCK_B_16;
                break;
        case TYPE_A_H_8:
                block_0->vis.pattern_code = ANI_BLOCK_A_H8;
                break;

        case OBJ_C:
                ani_div = (counter/8)%4;
                if (block_0->block_s == BLOCK_NORMAL) {
                        switch (ani_div){
                                case 0:
                                        block_0->vis.pattern_code = ANI_BLOCK_ITEM1;
                                        break;
                                case 1:
                                        block_0->vis.pattern_code = ANI_BLOCK_ITEM2;
                                        break;
                                case 2:
                                        block_0->vis.pattern_code = ANI_BLOCK_ITEM3;
                                        break;
                                default:
                                        block_0->vis.pattern_code = ANI_BLOCK_ITEM2;
                        }
                }
                else{
                        if (rel_counter < 4) {block_0->hit.y -= 1; block_0->vis.pattern_code
= ANI_BLOCK_ITEM_HIT;}
                        else if (rel_counter < 8) {block_0->hit.y += 1;
block_0->vis.pattern_code = ANI_BLOCK_ITEM_HIT;}
                        else {block_0->block_t = EMP;}
```

```
                }
                break;

            case OBJ_M:
                ani_div = (counter/8)%4;
                if (block_0->block_s == BLOCK_NORMAL) {
                    switch (ani_div){
                        case 0:
                            block_0->vis.pattern_code = ANI_BLOCK_ITEM1;
                            break;
                        case 1:
                            block_0->vis.pattern_code = ANI_BLOCK_ITEM2;
                            break;
                        case 2:
                            block_0->vis.pattern_code = ANI_BLOCK_ITEM3;
                            break;
                        default:
                            block_0->vis.pattern_code = ANI_BLOCK_ITEM2;
                    }
                }
                else{
                    if (rel_counter < 4) {block_0->hit.y -= 1; block_0->vis.pattern_code
= ANI_BLOCK_ITEM_HIT;}
                    else if (rel_counter < 8) {block_0->hit.y += 1;
block_0->vis.pattern_code = ANI_BLOCK_ITEM_HIT;}
                    else {block_0->block_t = EMP;}
                }
                break;

            default:
                block_0->vis.pattern_code = ANI_BLOCK_ITEM_EMP;
                break;
        }
        // block_0->vis.visible = 1;
        block_0->vis.x = block_0->hit.x - game_0->camera_pos;
        block_0->vis.y = block_0->hit.y;
        return;
}
void cloud_animation(mario_game *game_0, cloud* cloud_0, int f_counter){
        int counter, ani_div, rel_counter;
        cloud_0->vis.visible = 1;
        cloud_0->vis.x = cloud_0->hit.x - game_0->camera_pos;
        cloud_0->vis.y = cloud_0->hit.y;
}
```

```
void coin_animation(mario_game *game_0, coin* coin_0, int f_counter){
	int counter, ani_div, rel_counter;
	counter = (f_counter >= coin_0->animate_frame_counter)? f_counter : f_counter +
FRAME_LIMIT;
	rel_counter = counter - coin_0->animate_frame_counter;
	if (coin_0->coin_s == COIN_NORMAL) {
		ani_div = (counter/6)%4;
	}
	else{
		ani_div = (counter/2)%4;
		if (rel_counter <= 20) coin_0->hit.y += (rel_counter - 20) * 0.17;
		else if (rel_counter > 35) coin_0->enable = 0;
	}

	switch (ani_div){
		case 0:
			coin_0->vis.pattern_code = ANI_COIN_1;
			break;
		case 1:
			coin_0->vis.pattern_code = ANI_COIN_2;
			break;
		case 2:
			coin_0->vis.pattern_code = ANI_COIN_3;
			break;
		default:
			coin_0->vis.pattern_code = ANI_COIN_4;
	}

	// coin_0->vis.visible = 1;
	coin_0->vis.x = coin_0->hit.x - game_0->camera_pos;
	coin_0->vis.y = coin_0->hit.y;
	return;
}
```

Game_animation.h :

```
#ifndef _MARIO_GAME_ANIMATION
#define _MARIO_GAME_ANIMATION

#include "game_struct.h"

// Frame Limit
#define FRAME_LIMIT 6000
```

```
// Mario Animation
#define ANI_MARIO_S_NORMAL 0
#define ANI_MARIO_S_WALK1 1
#define ANI_MARIO_S_WALK2 2
#define ANI_MARIO_S_WALK3 3
#define ANI_MARIO_S_SHUT 4
#define ANI_MARIO_S_JUMP 5
#define ANI_MARIO_S_DEAD 6
#define ANI_MARIO_L_NORMAL 7
#define ANI_MARIO_L_WALK1 8
#define ANI_MARIO_L_WALK2 9
#define ANI_MARIO_L_WALK3 10
#define ANI_MARIO_L_SHUT 11
#define ANI_MARIO_L_JUMP 12
#define ANI_MARIO_L_SIT 13
#define ANI_MARIO_M_NORMAL 14
#define ANI_MARIO_L_HIT 15
#define ANI_MARIO_S_HIT 16
#define ANI_MARIO_S_HANG 17
#define ANI_MARIO_L_HANG 18

// Mush Animation
#define ANI_MUSH_NORMAL 0

// Goomba Animation
#define ANI_GOOMBA_NORMAL 0
#define ANI_GOOMBA_HIT 1

// Tube Animation
#define ANI_TUBE_H 0
#define ANI_TUBE_B 1

// Block Animation
#define ANI_BLOCK_ITEM1 0
#define ANI_BLOCK_ITEM2 1
#define ANI_BLOCK_ITEM3 2
#define ANI_BLOCK_ITEM_HIT 3
#define ANI_BLOCK_ITEM_EMP 4
#define ANI_BLOCK_A1 5
#define ANI_BLOCK_A2 6
#define ANI_BLOCK_A_HIT 7
#define ANI_BLOCK_B 8
#define ANI_BLOCK_A_H2 9
```

```
#define ANI_BLOCK_A_H3 10
#define ANI_BLOCK_A_H8 11
#define ANI_BLOCK_B_V2 12
#define ANI_BLOCK_B_V3 13
#define ANI_BLOCK_B_V4 14
#define ANI_BLOCK_B_16 15

// Cloud Animation
#define ANI_CLOUD_NORMAL 0

// Coin Animation
#define ANI_COIN_1 0
#define ANI_COIN_2 1
#define ANI_COIN_3 2
#define ANI_COIN_4 3

void mario_animation(mario_game *game_0, mario* mario_0, int f_counter);
void mush_animation(mario_game *game_0, mush* mush_0, int f_counter);
void goomba_animation(mario_game *game_0, goomba* goomba_0, int f_counter);
void tube_animation(mario_game *game_0, tube* tube_0, int f_counter);
void block_animation(mario_game *game_0, block* block_0, int f_counter);
void cloud_animation(mario_game *game_0, cloud* cloud_0, int f_counter);
void coin_animation(mario_game *game_0, coin* coin_0, int f_counter);
#endif
```

Game_animation.h:

```
#ifndef _MARIO_GAME_ANIMATION
#define _MARIO_GAME_ANIMATION

#include "game_struct.h"

// Frame Limit
#define FRAME_LIMIT 6000

// Mario Animation
#define ANI_MARIO_S_NORMAL 0
#define ANI_MARIO_S_WALK1 1
#define ANI_MARIO_S_WALK2 2
#define ANI_MARIO_S_WALK3 3
#define ANI_MARIO_S_SHUT 4
#define ANI_MARIO_S_JUMP 5
#define ANI_MARIO_S_DEAD 6
#define ANI_MARIO_L_NORMAL 7
```

```
#define ANI_MARIO_L_WALK1 8
#define ANI_MARIO_L_WALK2 9
#define ANI_MARIO_L_WALK3 10
#define ANI_MARIO_L_SHUT 11
#define ANI_MARIO_L_JUMP 12
#define ANI_MARIO_L_SIT 13
#define ANI_MARIO_M_NORMAL 14
#define ANI_MARIO_L_HIT 15
#define ANI_MARIO_S_HIT 16
#define ANI_MARIO_S_HANG 17
#define ANI_MARIO_L_HANG 18

// Mush Animation
#define ANI_MUSH_NORMAL 0

// Goomba Animation
#define ANI_GOOMBA_NORMAL 0
#define ANI_GOOMBA_HIT 1

// Tube Animation
#define ANI_TUBE_H 0
#define ANI_TUBE_B 1

// Block Animation
#define ANI_BLOCK_ITEM1 0
#define ANI_BLOCK_ITEM2 1
#define ANI_BLOCK_ITEM3 2
#define ANI_BLOCK_ITEM_HIT 3
#define ANI_BLOCK_ITEM_EMP 4
#define ANI_BLOCK_A1 5
#define ANI_BLOCK_A2 6
#define ANI_BLOCK_A_HIT 7
#define ANI_BLOCK_B 8
#define ANI_BLOCK_A_H2 9
#define ANI_BLOCK_A_H3 10
#define ANI_BLOCK_A_H8 11
#define ANI_BLOCK_B_V2 12
#define ANI_BLOCK_B_V3 13
#define ANI_BLOCK_B_V4 14
#define ANI_BLOCK_B_16 15

// Cloud Animation
#define ANI_CLOUD_NORMAL 0
```

```
// Coin Animation
#define ANI_COIN_1 0
#define ANI_COIN_2 1
#define ANI_COIN_3 2
#define ANI_COIN_4 3

void mario_animation(mario_game *game_0, mario* mario_0, int f_counter);
void mush_animation(mario_game *game_0, mush* mush_0, int f_counter);
void goomba_animation(mario_game *game_0, goomba* goomba_0, int f_counter);
void tube_animation(mario_game *game_0, tube* tube_0, int f_counter);
void block_animation(mario_game *game_0, block* block_0, int f_counter);
void cloud_animation(mario_game *game_0, cloud* cloud_0, int f_counter);
void coin_animation(mario_game *game_0, coin* coin_0, int f_counter);
#endif
```

Game_struct.c:

```
#include "game_struct.h"

enum contact hitbox_contact(const hit_box *A, const hit_box *B){
	float AL, AR, AU, AD, BL, BR, BU, BD, dx, dy;
	float ALP, ARP, AUP, ADP;
	float xv_r, yv_r;
	float t_h, t_v;
	int v_c, h_c; // vc pos: DOWN, neg: UP    hc pos RIGHT, neg : LEFT
	// BL = B->x; BR = (B->x) + B->sx; Bu = B->y; BD = (B->y) + B->sy;
	// Zero out
	BL = 0; BR = B->sx; BU = 0; BD = B->sy;
	AL = A->x - B->x; AR = A->x + A->sx - B->x;
	AU = A->y - B->y; AD = A->y + A->sy - B->y;
	// check if any corner inside before the update
	if ((AU > BU && AU < BD && AL > BL && AL < BR) // A left up corner
	||	(AU > BU && AU < BD && AR > BL && AR < BR) // A right up corner
	||	(AD > BU && AD < BD && AL > BL && AL < BR) // A left down corner
	||	(AD > BU && AD < BD && AR > BL && AR < BR) // A right down corner
		){
		return NONE; // Assume no contact
	}
	else{
		// check if any corner inside AFTER the update
		xv_r = A->vx - B->vx; yv_r = A->vy - B->vy;
```

```
                ALP = AL + xv_r; ARP = AR + xv_r; AUP = AU + yv_r; ADP = AD + yv_r;
                t_h = 0; t_v = 0;
                if ((AUP >= BU && AUP <= BD && ALP >= BL && ALP <= BR) // A left up corner
                ||      (AUP >= BU && AUP <= BD && ARP >= BL && ARP <= BR) // A right up
corner
                ||      (ADP >= BU && ADP <= BD && ALP >= BL && ALP <= BR) // A left down
corner
                ||      (ADP >= BU && ADP <= BD && ARP >= BL && ARP <= BR) // A right
down corner
                ){
                        if (AR <= BL && xv_r != 0) {h_c = 1; dx = BL - AR; t_h = dx/xv_r;}
                        else if (AL >= BR  && xv_r != 0) {h_c = -1; dx = BR - AL; t_h = dx/xv_r;}
                        else h_c = 0;

                        if (AD <= BU && yv_r != 0) {v_c = 1; dy = BU -AD; t_v = dy/yv_r;}
                        else if (AU >= BD && yv_r != 0) {v_c = -1; dy = BD - AU; t_v = dy/yv_r;}
                        else v_c = 0;

                        if (t_v >= 0 && v_c != 0){
                                if (t_h < 0 || h_c == 0){
                                        return (v_c==1)? DOWN : UP;
                                }
                                else if (t_h >= 0 && h_c != 0){
                                        if (t_v >= t_h){
                                                return (v_c==1)? DOWN : UP;
                                        }
                                        return (h_c==1)? RIGHT : LEFT;
                                }
                                return (v_c==1)? DOWN : UP;
                        }

                        else if (t_h >=   0 && h_c != 0){
                                return (h_c==1)? RIGHT : LEFT;
                        }

                        return NONE;

                }
                else{
                        return NONE;
                }
                return NONE;
        }
        return NONE;
```

```
}

void new_game(mario_game *game){
	// camera
	game->camera_pos = 0;
	//Mario
	game->mario_0.enable = 1;
	game->mario_0.x_block = 0;
	game->mario_0.y_block = 0;
	game->mario_0.hit.sx = 16;
	game->mario_0.hit.sy = 16;
	game->mario_0.hit.x = 128;
	game->mario_0.hit.y = 128;
	game->mario_0.hit.vx = 0;
	game->mario_0.hit.vy = 0;
	game->mario_0.control_s = MARIO_NORMAL;
	game->mario_0.animate_s = HIT;
	game->mario_0.game_s = MARIO_GAME_NORMAL;
	game->mario_0.mario_s = SMALL;
	// Ground
	game->ground_list[0].hit.x = 0;
	game->ground_list[0].hit.y = 368;
	game->ground_list[0].hit.vx = 0;
	game->ground_list[0].hit.vy = 0;
	game->ground_list[0].hit.sx = 448;
	game->ground_list[0].hit.sy = 32;

	game->ground_list[1].hit.x = 528;
	game->ground_list[1].hit.y = 368;
	game->ground_list[1].hit.vx = 0;
	game->ground_list[1].hit.vy = 0;
	game->ground_list[1].hit.sx = 944;
	game->ground_list[1].hit.sy = 32;

	game->ground_list[2].hit.x = 1504;
	game->ground_list[2].hit.y = 368;
	game->ground_list[2].hit.vx = 0;
	game->ground_list[2].hit.vy = 0;
	game->ground_list[2].hit.sx = 640;
	game->ground_list[2].hit.sy = 32;

	game->ground_list[3].hit.x = 2176;
	game->ground_list[3].hit.y = 368;
	game->ground_list[3].hit.vx = 0;
```

```
game->ground_list[3].hit.vy = 0;
game->ground_list[3].hit.sx = 800;
game->ground_list[3].hit.sy = 32;

//Block
int b_i;
for (b_i = 0; b_i < BLOCK_NUM; b_i++){
        game->block_list[b_i].hit.x = 0;
        game->block_list[b_i].hit.y = 0;
        game->block_list[b_i].hit.vx = 0;
        game->block_list[b_i].hit.vy = 0;
        game->block_list[b_i].hit.sx = 16;
        game->block_list[b_i].hit.sy = 16;
        game->block_list[b_i].enable = 0;
        game->block_list[b_i].loaded = 0;
        game->block_list[b_i].L = 0;
        game->block_list[b_i].R = 0;
        game->block_list[b_i].U = 0;
        game->block_list[b_i].D = 0;
}

game->block_list[0].hit.x = 320;
game->block_list[0].hit.y = 304;
game->block_list[0].hit.vx = 0;
game->block_list[0].hit.vy = 0;
game->block_list[0].hit.sx = 16;
game->block_list[0].hit.sy = 16;
game->block_list[0].enable = 1;
game->block_list[0].loaded = 0;
game->block_list[0].L = 1;
game->block_list[0].R = 0;
game->block_list[0].U = 0;
game->block_list[0].D = 0;
game->block_list[0].block_t = TYPE_A;
game->block_list[0].block_s = BLOCK_NORMAL;
game->block_list[0].vis.visible = 1;

game->block_list[1].hit.x = 304;
game->block_list[1].hit.y = 304;
game->block_list[1].hit.vx = 0;
game->block_list[1].hit.vy = 0;
game->block_list[1].hit.sx = 16;
game->block_list[1].hit.sy = 16;
game->block_list[1].enable = 1;
```

```
game->block_list[1].loaded = 0;
game->block_list[1].L = 1;
game->block_list[1].R = 1;
game->block_list[1].U = 0;
game->block_list[1].D = 0;
game->block_list[1].block_t = OBJ_C;
game->block_list[1].block_s = BLOCK_NORMAL;
game->block_list[1].vis.visible = 1;

game->block_list[2].hit.x = 288;
game->block_list[2].hit.y = 304;
game->block_list[2].hit.vx = 0;
game->block_list[2].hit.vy = 0;
game->block_list[2].hit.sx = 16;
game->block_list[2].hit.sy = 16;
game->block_list[2].enable = 1;
game->block_list[2].loaded =0;
game->block_list[2].L = 1;
game->block_list[2].R = 1;
game->block_list[2].U = 0;
game->block_list[2].D = 0;
game->block_list[2].block_t = TYPE_A;
game->block_list[2].block_s = BLOCK_NORMAL;
game->block_list[2].vis.visible = 1;

game->block_list[3].hit.x = 272;
game->block_list[3].hit.y = 304;
game->block_list[3].hit.vx = 0;
game->block_list[3].hit.vy = 0;
game->block_list[3].hit.sx = 16;
game->block_list[3].hit.sy = 16;
game->block_list[3].enable = 1;
game->block_list[3].loaded = 0;
game->block_list[3].L = 0;
game->block_list[3].R = 1;
game->block_list[3].U = 0;
game->block_list[3].D = 0;
game->block_list[3].block_t = OBJ_M;
game->block_list[3].block_s = BLOCK_NORMAL;
game->block_list[3].vis.visible = 1;

game->block_list[4].hit.x = 224;
game->block_list[4].hit.y = 304;
game->block_list[4].hit.vx = 0;
```

```
game->block_list[4].hit.vy = 0;
game->block_list[4].hit.sx = 16;
game->block_list[4].hit.sy = 16;
game->block_list[4].enable = 1;
game->block_list[4].loaded = 0;
game->block_list[4].L = 0;
game->block_list[4].R = 0;
game->block_list[4].U = 0;
game->block_list[4].D = 0;
game->block_list[4].block_t = TYPE_A;
game->block_list[4].block_s = BLOCK_NORMAL;
game->block_list[4].vis.visible = 1;



game->block_list[5].hit.x = 560 + 48;
game->block_list[5].hit.y = 352;
game->block_list[5].hit.vx = 0;
game->block_list[5].hit.vy = 0;
game->block_list[5].hit.sx = 16;
game->block_list[5].hit.sy = 16;
game->block_list[5].enable = 1;
game->block_list[5].loaded = 0;
game->block_list[5].L = 0;
game->block_list[5].R = 0;
game->block_list[5].U = 0;
game->block_list[5].D = 0;
game->block_list[5].block_t = TYPE_B_1;
game->block_list[5].block_s = BLOCK_NORMAL;
game->block_list[5].vis.visible = 1;

game->block_list[6].hit.x = 576 + 48;
game->block_list[6].hit.y = 336;
game->block_list[6].hit.vx = 0;
game->block_list[6].hit.vy = 0;
game->block_list[6].hit.sx = 16;
game->block_list[6].hit.sy = 32;
game->block_list[6].enable = 1;
game->block_list[6].loaded = 0;
game->block_list[6].L = 0;
game->block_list[6].R = 0;
game->block_list[6].U = 0;
game->block_list[6].D = 0;
game->block_list[6].block_t = TYPE_B_2;
```

```
game->block_list[6].block_s = BLOCK_NORMAL;
game->block_list[6].vis.visible = 1;

game->block_list[7].hit.x = 592 + 48;
game->block_list[7].hit.y = 320;
game->block_list[7].hit.vx = 0;
game->block_list[7].hit.vy = 0;
game->block_list[7].hit.sx = 16;
game->block_list[7].hit.sy = 48;
game->block_list[7].enable = 1;
game->block_list[7].loaded =0;
game->block_list[7].L = 0;
game->block_list[7].R = 0;
game->block_list[7].U = 0;
game->block_list[7].D = 0;
game->block_list[7].block_t = TYPE_B_3;
game->block_list[7].block_s = BLOCK_NORMAL;
game->block_list[7].vis.visible = 1;

game->block_list[8].hit.x = 608 + 48;
game->block_list[8].hit.y = 304;
game->block_list[8].hit.vx = 0;
game->block_list[8].hit.vy = 0;
game->block_list[8].hit.sx = 16;
game->block_list[8].hit.sy = 64;
game->block_list[8].enable = 1;
game->block_list[8].loaded = 0;
game->block_list[8].L = 0;
game->block_list[8].R = 0;
game->block_list[8].U = 0;
game->block_list[8].D = 0;
game->block_list[8].block_t = TYPE_B_4;
game->block_list[8].block_s = BLOCK_NORMAL;
game->block_list[8].vis.visible = 1;

game->block_list[9].hit.x = 688 + 48;
game->block_list[9].hit.y = 256;
game->block_list[9].hit.vx = 0;
game->block_list[9].hit.vy = 0;
game->block_list[9].hit.sx = 128;
game->block_list[9].hit.sy = 16;
game->block_list[9].enable = 1;
game->block_list[9].loaded = 0;
game->block_list[9].L = 0;
```

```
game->block_list[9].R = 0;
game->block_list[9].U = 0;
game->block_list[9].D = 0;
game->block_list[9].block_t = TYPE_A_H_8;
game->block_list[9].block_s = BLOCK_NORMAL;
game->block_list[9].vis.visible = 1;

game->block_list[10].hit.x = 672 + 48;
game->block_list[10].hit.y = 176;
game->block_list[10].hit.vx = 0;
game->block_list[10].hit.vy = 0;
game->block_list[10].hit.sx = 16;
game->block_list[10].hit.sy = 16;
game->block_list[10].enable = 1;
game->block_list[10].loaded = 0;
game->block_list[10].L = 0;
game->block_list[10].R = 0;
game->block_list[10].U = 0;
game->block_list[10].D = 0;
game->block_list[10].block_t = OBJ_C;
game->block_list[10].block_s = BLOCK_NORMAL;
game->block_list[10].vis.visible = 1;

game->block_list[11].hit.x = 736 + 48;
game->block_list[11].hit.y = 176;
game->block_list[11].hit.vx = 0;
game->block_list[11].hit.vy = 0;
game->block_list[11].hit.sx = 16;
game->block_list[11].hit.sy = 16;
game->block_list[11].enable = 1;
game->block_list[11].loaded = 0;
game->block_list[11].L = 0;
game->block_list[11].R = 0;
game->block_list[11].U = 0;
game->block_list[11].D = 0;
game->block_list[11].block_t = OBJ_C;
game->block_list[11].block_s = BLOCK_NORMAL;
game->block_list[11].vis.visible = 1;

game->block_list[12].hit.x = 800 + 48;
game->block_list[12].hit.y = 176;
game->block_list[12].hit.vx = 0;
game->block_list[12].hit.vy = 0;
game->block_list[12].hit.sx = 16;
```

```
game->block_list[12].hit.sy = 16;
game->block_list[12].enable = 1;
game->block_list[12].loaded = 0;
game->block_list[12].L = 0;
game->block_list[12].R = 0;
game->block_list[12].U = 0;
game->block_list[12].D = 0;
game->block_list[12].block_t = OBJ_C;
game->block_list[12].block_s = BLOCK_NORMAL;
game->block_list[12].vis.visible = 1;

game->block_list[13].hit.x = 736 + 48;
game->block_list[13].hit.y = 304;
game->block_list[13].hit.vx = 0;
game->block_list[13].hit.vy = 0;
game->block_list[13].hit.sx = 16;
game->block_list[13].hit.sy = 16;
game->block_list[13].enable = 1;
game->block_list[13].loaded = 0;
game->block_list[13].L = 0;
game->block_list[13].R = 0;
game->block_list[13].U = 0;
game->block_list[13].D = 0;
game->block_list[13].block_t = OBJ_M;
game->block_list[13].block_s = BLOCK_NORMAL;
game->block_list[13].vis.visible = 1;

game->block_list[14].hit.x = 1104 + 48;
game->block_list[14].hit.y = 272;
game->block_list[14].hit.vx = 0;
game->block_list[14].hit.vy = 0;
game->block_list[14].hit.sx = 16;
game->block_list[14].hit.sy = 48;
game->block_list[14].enable = 1;
game->block_list[14].loaded = 0;
game->block_list[14].L = 0;
game->block_list[14].R = 0;
game->block_list[14].U = 0;
game->block_list[14].D = 0;
game->block_list[14].block_t = TYPE_B_3;
game->block_list[14].block_s = BLOCK_NORMAL;
game->block_list[14].vis.visible = 1;

game->block_list[15].hit.x = 1120 + 48;
```

```
game->block_list[15].hit.y = 304;
game->block_list[15].hit.vx = 0;
game->block_list[15].hit.vy = 0;
game->block_list[15].hit.sx = 16;
game->block_list[15].hit.sy = 16;
game->block_list[15].enable = 1;
game->block_list[15].loaded = 0;
game->block_list[15].L = 1;
game->block_list[15].R = 1;
game->block_list[15].U = 0;
game->block_list[15].D = 0;
game->block_list[15].block_t = TYPE_A;
game->block_list[15].block_s = BLOCK_NORMAL;
game->block_list[15].vis.visible = 1;

game->block_list[16].hit.x = 1136 + 48;
game->block_list[16].hit.y = 304;
game->block_list[16].hit.vx = 0;
game->block_list[16].hit.vy = 0;
game->block_list[16].hit.sx = 16;
game->block_list[16].hit.sy = 16;
game->block_list[16].enable = 1;
game->block_list[16].loaded = 0;
game->block_list[16].L = 1;
game->block_list[16].R = 1;
game->block_list[16].U = 0;
game->block_list[16].D = 0;
game->block_list[16].block_t = TYPE_A;
game->block_list[16].block_s = BLOCK_NORMAL;
game->block_list[16].vis.visible = 1;

game->block_list[17].hit.x = 1152 + 48;
game->block_list[17].hit.y = 272;
game->block_list[17].hit.vx = 0;
game->block_list[17].hit.vy = 0;
game->block_list[17].hit.sx = 16;
game->block_list[17].hit.sy = 48;
game->block_list[17].enable = 1;
game->block_list[17].loaded = 0;
game->block_list[17].L = 0;
game->block_list[17].R = 0;
game->block_list[17].U = 0;
game->block_list[17].D = 0;
game->block_list[17].block_t = TYPE_B_3;
```

```
game->block_list[17].block_s = BLOCK_NORMAL;
game->block_list[17].vis.visible = 1;
// Sort block with block x accending
int i, j;
block temp;
for (i = 0; i < BLOCK_NUM - 1; i++)
        for (j = 0; j < BLOCK_NUM - 1 - i; j++) {
                if (game->block_list[j].hit.x > game->block_list[j + 1].hit.x) {
                        temp = game->block_list[j];
              game->block_list[j] = game->block_list[j + 1];
              game->block_list[j + 1] = temp;
            }
          }
for (i = 0; i < BLOCK_NUM; i++)
        printf("%d: %f \n", i, game->block_list[i].hit.x);

// Coin
int c_i;
for (c_i = 0; c_i < COIN_NUM; c_i++){
        game->coin_list[c_i].hit.x = 0;
        game->coin_list[c_i].hit.y = 0;
        game->coin_list[c_i].hit.vx = 0;
        game->coin_list[c_i].hit.vy = 0;
        game->coin_list[c_i].hit.sx = 8;
        game->coin_list[c_i].hit.sy = 16;
        game->coin_list[c_i].enable = 0;
        game->coin_list[c_i].loaded = 0;
}

game->coin_list[0].hit.x = 324;
game->coin_list[0].hit.y = 352;
game->coin_list[0].hit.vx = 0;
game->coin_list[0].hit.vy = 0;
game->coin_list[0].hit.sx = 8;
game->coin_list[0].hit.sy = 16;
game->coin_list[0].enable = 1;
game->coin_list[0].loaded = 1;
game->coin_list[0].coin_s = COIN_NORMAL;

game->coin_list[1].hit.x = 964;
game->coin_list[1].hit.y = 352;
game->coin_list[1].hit.vx = 0;
game->coin_list[1].hit.vy = 0;
game->coin_list[1].hit.sx = 8;
```

```
game->coin_list[1].hit.sy = 16;
game->coin_list[1].enable = 1;
game->coin_list[1].loaded = 1;
game->coin_list[1].coin_s = COIN_NORMAL;

game->coin_list[2].hit.x = 324;
game->coin_list[2].hit.y = 288;
game->coin_list[2].hit.vx = 0;
game->coin_list[2].hit.vy = 0;
game->coin_list[2].hit.sx = 8;
game->coin_list[2].hit.sy = 16;
game->coin_list[2].enable = 1;
game->coin_list[2].loaded = 1;
game->coin_list[2].coin_s = COIN_NORMAL;

game->coin_list[3].hit.x = 656 + 48;
game->coin_list[3].hit.y = 352;
game->coin_list[3].hit.vx = 0;
game->coin_list[3].hit.vy = 0;
game->coin_list[3].hit.sx = 8;
game->coin_list[3].hit.sy = 16;
game->coin_list[3].enable = 1;
game->coin_list[3].loaded = 1;
game->coin_list[3].coin_s = COIN_NORMAL;

game->coin_list[4].hit.x = 704 + 48;
game->coin_list[4].hit.y = 352;
game->coin_list[4].hit.vx = 0;
game->coin_list[4].hit.vy = 0;
game->coin_list[4].hit.sx = 8;
game->coin_list[4].hit.sy = 16;
game->coin_list[4].enable = 1;
game->coin_list[4].loaded = 1;
game->coin_list[4].coin_s = COIN_NORMAL;

game->coin_list[5].hit.x = 1124 + 48;
game->coin_list[5].hit.y = 288;
game->coin_list[5].hit.vx = 0;
game->coin_list[5].hit.vy = 0;
game->coin_list[5].hit.sx = 8;
game->coin_list[5].hit.sy = 16;
game->coin_list[5].enable = 1;
game->coin_list[5].loaded = 1;
game->coin_list[5].coin_s = COIN_NORMAL;
```

```
game->coin_list[6].hit.x = 1140 + 48;
game->coin_list[6].hit.y = 288;
game->coin_list[6].hit.vx = 0;
game->coin_list[6].hit.vy = 0;
game->coin_list[6].hit.sx = 8;
game->coin_list[6].hit.sy = 16;
game->coin_list[6].enable = 1;
game->coin_list[6].loaded = 1;
game->coin_list[6].coin_s = COIN_NORMAL;

// Mush
int m_i;
for (m_i = 0; m_i < MUSH_NUM; m_i++){
        game->mush_list[m_i].hit.x = 0;
        game->mush_list[m_i].hit.y = 0;
        game->mush_list[m_i].hit.vx = 0;
        game->mush_list[m_i].hit.vy = 0;
        game->mush_list[m_i].hit.sx = 16;
        game->mush_list[m_i].hit.sy = 16;
        game->mush_list[m_i].enable = 0;
        game->mush_list[m_i].loaded = 0;
}


// Goomba
int g_i;
for (g_i = 0; g_i < GOOMBA_NUM; g_i++){
        game->goomba_list[g_i].hit.x = 0;
        game->goomba_list[g_i].hit.y = 0;
        game->goomba_list[g_i].hit.vx = 0;
        game->goomba_list[g_i].hit.vy = 0;
        game->goomba_list[g_i].hit.sx = 16;
        game->goomba_list[g_i].hit.sy = 16;
        game->goomba_list[g_i].enable = 0;
        game->goomba_list[g_i].loaded = 0;
}

game->goomba_list[0].hit.x = 324;
game->goomba_list[0].hit.y = 268;
game->goomba_list[0].hit.vx = 0;
game->goomba_list[0].hit.vy = 0;
game->goomba_list[0].hit.sx = 16;
game->goomba_list[0].hit.sy = 16;
```

```
game->goomba_list[0].enable = 1;
game->goomba_list[0].loaded = 1;
game->goomba_list[0].goomba_s = GOOMBA_NORMAL;
game->goomba_list[0].l_limit = 300;
game->goomba_list[0].r_limit = 600;

game->goomba_list[1].hit.x = 720 + 48;
game->goomba_list[1].hit.y = 208;
game->goomba_list[1].hit.vx = 0;
game->goomba_list[1].hit.vy = 0;
game->goomba_list[1].hit.sx = 16;
game->goomba_list[1].hit.sy = 16;
game->goomba_list[1].enable = 1;
game->goomba_list[1].loaded = 1;
game->goomba_list[1].goomba_s = GOOMBA_NORMAL;
game->goomba_list[1].l_limit = 704;
game->goomba_list[1].r_limit = 800;

game->goomba_list[2].hit.x = 1124 + 48;
game->goomba_list[2].hit.y = 272;
game->goomba_list[2].hit.vx = 0;
game->goomba_list[2].hit.vy = 0;
game->goomba_list[2].hit.sx = 16;
game->goomba_list[2].hit.sy = 16;
game->goomba_list[2].enable = 1;
game->goomba_list[2].loaded = 1;
game->goomba_list[2].goomba_s = GOOMBA_NORMAL;
game->goomba_list[2].l_limit = 1060;
game->goomba_list[2].r_limit = 1204;

// Tube
int t_i;
for (t_i = 0; t_i < TUBE_NUM; t_i++){
        game->tube_list[t_i].hit.x = 0;
        game->tube_list[t_i].hit.y = 0;
        game->tube_list[t_i].hit.vx = 0;
        game->tube_list[t_i].hit.vy = 0;
        game->tube_list[t_i].hit.sx = 32;
        game->tube_list[t_i].hit.sy = 160;
        game->tube_list[t_i].enable = 0;
        game->tube_list[t_i].loaded = 0;
}

game->tube_list[0].hit.x = 416;
```

```
	game->tube_list[0].hit.y = 320;
	game->tube_list[0].hit.vx = 0;
	game->tube_list[0].hit.vy = 0;
	game->tube_list[0].hit.sx = 32;
	game->tube_list[0].hit.sy = 160;
	game->tube_list[0].enable = 1;
	game->tube_list[0].loaded = 1;

	// Cloud
	for (t_i = 0; t_i < CLOUD_NUM; t_i++){
		game->cloud_list[t_i].hit.x = 0;
		game->cloud_list[t_i].hit.y = 0;
		game->cloud_list[t_i].hit.vx = 0;
		game->cloud_list[t_i].hit.vy = 0;
		game->cloud_list[t_i].hit.sx = 0;
		game->cloud_list[t_i].hit.sy = 0;
		game->cloud_list[t_i].enable = 0;
		game->cloud_list[t_i].loaded = 0;
	}

	game->cloud_list[0].hit.x = 240;
	game->cloud_list[0].hit.y = 176;
	game->cloud_list[0].hit.vx = 0;
	game->cloud_list[0].hit.vy = 0;
	game->cloud_list[0].hit.sx = 0;
	game->cloud_list[0].hit.sy = 0;
	game->cloud_list[0].enable = 1;
	game->cloud_list[0].loaded = 1;

	game->cloud_list[1].hit.x = 240 + CAMERA_SIZE + 60 + 48;
	game->cloud_list[1].hit.y = 144;
	game->cloud_list[1].hit.vx = 0;
	game->cloud_list[1].hit.vy = 0;
	game->cloud_list[1].hit.sx = 0;
	game->cloud_list[1].hit.sy = 0;
	game->cloud_list[1].enable = 1;
	game->cloud_list[1].loaded = 1;


}
```

Game_struct.h :

```
#ifndef _MARIO_GAME_STRUCT
#define _MARIO_GAME_STRUCT
```

```
#define BLOCK_NUM 45
#define MUSH_NUM 3
#define GOOMBA_NUM 10
#define CLOUD_NUM 7
#define GROUND_NUM 4
#define TUBE_NUM 5
#define COIN_NUM 25

#define GRAVITY (0.23)
#define MAX_SPEED_H (1.85)
#define MAX_SPEED_V (4.6)

// Mario data
#define WALK_ACC (0.09)
#define SHUT_ACC (0.12)
#define JUMP_INIT_V_SMALL (4.6)
#define JUMP_INIT_V_LARGE (5.6)
#define MAX_SPEED_V_JUMP (8.1)

// Mush DATA


// PPU CHILD LIMIT
#define BLOCK_VIS_LIMIT 9
#define COIN_VIS_LIMIT 4
#define GOOMBA_VIS_LIMIT 2
#define MUSH_VIS_LIMIT 1
#define TUBE_VIS_LIMIT 1
#define CLOUD_VIS_LIMIT 1

enum contact{NONE, LEFT, RIGHT, UP, DOWN};

typedef struct{
   float x, y, vx, vy; // location $ speed
   int sx, sy;         // size of the box
} hit_box;

typedef struct{
   unsigned int pattern_code;
   int visible, flip;
   int x, y;
   int extra_info;
```

```
} ppu_obj_info;

enum mario_control_state{MARIO_NORMAL, MARIO_ANIMATE};
enum mario_animate_state{HIT, ENLARGE, ANI_DEAD};
enum mario_game_state{MARIO_GAME_NORMAL, MARIO_GMAE_HIT};
enum mario_state{SMALL, LARGE, DEAD};

typedef struct {
    int enable;
        int x_block;
        int y_block;
    hit_box hit;
    ppu_obj_info vis;
    enum mario_control_state control_s;
    enum mario_game_state game_s;
    enum mario_state mario_s;
    enum mario_animate_state animate_s;
    float acc_x, acc_y; // Accelerate
    int animate_frame_counter;
} mario;

enum mush_state{MUSH_NORMAL, MUSH_ANIMATE};

typedef struct {
    int enable;
        int loaded;
        int x_block;
        int y_block;
    hit_box hit;
    ppu_obj_info vis;
    enum mush_state mush_s;
    float acc_x, acc_y; // Accelerate
    int animate_frame_counter;
} mush;

enum coin_state{COIN_NORMAL, COIN_ANIMATE};

typedef struct {
    int enable;
        int loaded;
    hit_box hit;
    ppu_obj_info vis;
    enum coin_state coin_s;
    int animate_frame_counter;
```

```
} coin;

enum goomba_state{GOOMBA_NORMAL, GOOMBA_ANIMATE};

typedef struct {
    int enable;
        int loaded;
        int x_block;
        int y_block;
    hit_box hit;
    ppu_obj_info vis;
    enum goomba_state goomba_s;
    float acc_x, acc_y; // Accelerate
    int r_limit, l_limit;
    int animate_frame_counter;
} goomba;

enum block_type{TYPE_A, TYPE_B_1, TYPE_B_2, TYPE_B_3, TYPE_B_4, TYPE_B_16,
TYPE_A_H_8, OBJ_C, OBJ_M, EMP};
enum block_state{BLOCK_NORMAL, BLOCK_ANIMATE};

typedef struct {
    int enable;
        int loaded;
        int extra_info;
        int L;
        int R;
        int U;
        int D;
    hit_box hit;
    ppu_obj_info vis;
    enum block_type block_t;
    enum block_state block_s;
    int animate_frame_counter;
} block;

typedef struct {
    hit_box hit;
} ground;

typedef struct {
        int enable;
        int loaded;
    hit_box hit;
```

```
    ppu_obj_info vis;
} tube;

typedef struct {
    int enable;
        int loaded;
        hit_box hit;
    ppu_obj_info vis;
} cloud;

#define LOAD_LIMIT (5*16)
#define CAMERA_SIZE (30*16)
enum game_state{GAME_START, GAME_NORMAL, GAME_END};
typedef struct {
        enum game_state game_s;
        int camera_pos;
        mario mario_0;
        ground ground_list[GROUND_NUM];
        block block_list[BLOCK_NUM];
        coin coin_list[COIN_NUM];
    mush mush_list[MUSH_NUM];
    goomba goomba_list[GOOMBA_NUM];
    tube tube_list[TUBE_NUM];
    cloud cloud_list[CLOUD_NUM];
} mario_game;



enum contact hitbox_contact(const hit_box *A, const hit_box *B);
void new_game(mario_game *game);

#endif
```

Hello.c:

```
/*
 * Userspace program that communicates with the vga_ball device driver
 * through ioctls
 *
 * Stephen A. Edwards
```

```
 * Columbia University
 */

#include <stdio.h>
#include "vga_ball.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>

int vga_ball_fd;

/* Read and print the background color */
void print_background_color() {
  vga_ball_arg_t vla;

  if (ioctl(vga_ball_fd, VGA_BALL_READ_BACKGROUND, &vla)) {
      perror("ioctl(VGA_BALL_READ_BACKGROUND) failed");
      return;
  }
  printf("%02x %02x %02x\n",
           vla.background.red, vla.background.green, vla.background.blue);
}

/* Set the background color */
void set_background_color(const vga_ball_color_t *c, unsigned char xl, unsigned char xh,
unsigned char yl, unsigned char yh)
{
  vga_ball_arg_t vla;
  vla.background = *c;
           vla.coordinate.xl = xl;
           vla.coordinate.xh = xh;
           vla.coordinate.yl = yl;
           vla.coordinate.yh = yh;
  if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &vla)) {
      perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
      return;
  }
}
```

```
void write_2_hw(int addr, int info)
{
        vga_ball_arg_t vla;
        vla.addr = addr;
        vla.info = info;
        if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &vla)) {
                perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
                return;
        }
}

int main()
{
  vga_ball_arg_t vla;
  int i;
        uint16_t clk_count = 0;
  static const char filename[] = "/dev/vga_ball";
        unsigned char xl, xh, yl, yh;
        uint16_t x_16, y_16;
        uint16_t x_v, y_v;
        int8_t x_sign, y_sign;
        int bus_info;
        int ping_pong = 0;
        int info_001 = 1;
        int info_010 = 2;
        int info_011 = 3;
        int info_100 = 4;
        int flip = 0;
        int flip_count = 400;
        int pp = 0;
        int pp_count = 20;
        int coin_pp_count = 8;
        int block_pp;
        int coin_pp = 0;
        int goomba_flip = 0;
        int goomba_flip_count = 6;
        int ground_shift = 0;
        int ground_count = 7;

        //=========================
        int margin_width = 5 * 16;

  static const vga_ball_color_t colors[] = {
```

```
    { 0xff, 0x00, 0x00 }, /* Red */
    { 0x00, 0xff, 0x00 }, /* Green */
    { 0x00, 0x00, 0xff }, /* Blue */
    { 0xff, 0xff, 0x00 }, /* Yellow */
    { 0x00, 0xff, 0xff }, /* Cyan */
    { 0xff, 0x00, 0xff }, /* Magenta */
    { 0x80, 0x80, 0x80 }, /* Gray */
    { 0x00, 0x00, 0x00 }, /* Black */
    { 0xff, 0xff, 0xff }  /* White */
  };

# define COLORS 9

          time_t t;
          srand((unsigned) time(&t));

  printf("VGA ball Userspace program started\n");

  if ( (vga_ball_fd = open(filename, O_RDWR)) == -1) {
    fprintf(stderr, "could not open %s\n", filename);
    return -1;
  }
          x_16 = (uint16_t)(rand()%200 + 100);
          y_16 = (uint16_t)(rand()%100 + 100);
          xh = (unsigned char)((x_16>>8) & 0xFF);
          xl = (unsigned char)(x_16 & 0xFF);
          yh = (unsigned char)((y_16>>8) & 0xFF);
          yl = (unsigned char)(y_16 & 0xFF);
          x_v = 5 + rand()%5;
          y_v = 5 + rand()%5;

          vga_ball_color_t back = { 0x00, 0x00, 0xff };
          clk_count = 0;
          x_sign = 1;
          y_sign = 1;
          while (1){

                    if (x_16 >= 450-45) x_sign = -1;
                    else if(x_16 <= 45) x_sign = 1;

                    if (y_16 >= 450-(45/2)) y_sign = -1;
                    else if(y_16 <= 45/2) y_sign = 1;

                    if(clk_count%x_v == 0){
```

```
                x_16 = x_sign == 1? x_16 + 1 : x_16 - 1;
            }
            if(clk_count%y_v == 0){
                y_16 = y_sign == 1? y_16 + 1 : y_16 - 1;
            }
            xh = (unsigned char)((x_16>>8) & 0xFF);
            xl = (unsigned char)(x_16 & 0xFF);
            yh = (unsigned char)((y_16>>8) & 0xFF);
            yl = (unsigned char)(y_16 & 0xFF);
            // bus_info = (x_16 & 0x3FF) << 10;
            // bus_info += (y_16 & 0x3FF);
            // write_2_hw(0, bus_info);

            if(clk_count%flip_count == 0){
                flip = flip? 0:1;
            }
            //Mario ==========================
            bus_info = (1 << 26);
            write_2_hw(0, (int)(bus_info + (1 << 17) + (info_001 << 14) + (ping_pong << 13)
+ (1 << 12) + (flip << 11) + (pp & 0x1F)));
            write_2_hw(0, (int)(bus_info + (1 << 17) + (info_010 << 14) + (ping_pong << 13)
+ ((margin_width + x_16) & 0x3FF)));
            write_2_hw(0, (int)(bus_info + (1 << 17) + (info_011 << 14) + (ping_pong << 13)
+ (y_16 & 0x3FF)));
            // write_2_hw(0, (int)(bus_info + (1 << 17) + (info_100 << 14) + (ping_pong <<
13) + (0 & 0x3FF)));
            //Block ==========================
            for(int i = 0; i < 9; i++){
                block_pp = i + 8;
                write_2_hw(0, (int)((2 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_001
<< 14) + (ping_pong << 13) + (1 << 12) + (flip << 11) + (block_pp & 0x1F)));
                write_2_hw(0, (int)((2 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_010
<< 14) + (ping_pong << 13) + (((i%4) * 64 + margin_width) & 0x3FF)));
                write_2_hw(0, (int)((2 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_011
<< 14) + (ping_pong << 13) + (((int)(i/4)*64) & 0x3FF)));
                // write_2_hw(0, (int)((2 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_100
<< 14) + (ping_pong << 13) + (8 & 0x3FF)));
            }
            //Coin ==========================
            for(int i = 0; i < 4; i++){
                write_2_hw(0, (int)((3 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_001
<< 14) + (ping_pong << 13) + (1 << 12) + (flip << 11) + (coin_pp & 0x1F)));
                write_2_hw(0, (int)((3 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_010
<< 14) + (ping_pong << 13) + ((margin_width + 350 + (i%4) * 32) & 0x3FF)));
```

```
                        write_2_hw(0, (int)((3 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_011
<< 14) + (ping_pong << 13) + (((int)(i/2)*64) & 0x3FF)));
                        // write_2_hw(0, (int)((3 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_100
<< 14) + (ping_pong << 13) + (0 & 0x3FF)));
                }

                //Cloud ========================
                for(int i = 0; i < 1; i++){
                        write_2_hw(0, (int)((14 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_001
<< 14) + (ping_pong << 13) + (1 << 12) + (flip << 11) + (0 & 0x1F)));
                        write_2_hw(0, (int)((14 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_010
<< 14) + (ping_pong << 13) + ((margin_width + 270 + (i%4) * 64) & 0x3FF)));
                        write_2_hw(0, (int)((14 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_011
<< 14) + (ping_pong << 13) + (((int)(i/4)*64 + 250) & 0x3FF)));
                        // write_2_hw(0, (int)((3 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_100
<< 14) + (ping_pong << 13) + (0 & 0x3FF)));
                }

                //Mush ========================
                for(int i = 0; i < 2; i++){
                        write_2_hw(0, (int)((9 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_001
<< 14) + (ping_pong << 13) + (1 << 12) + (flip << 11) + (i & 0x1F)));
                        write_2_hw(0, (int)((9 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_010
<< 14) + (ping_pong << 13) + ((margin_width + 270 + (i%4) * 32) & 0x3FF)));
                        write_2_hw(0, (int)((9 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_011
<< 14) + (ping_pong << 13) + (((int)(i/4)*64 + 180) & 0x3FF)));
                        // write_2_hw(0, (int)((3 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_100
<< 14) + (ping_pong << 13) + (0 & 0x3FF)));
                }

                //Tube ========================
                for(int i = 0; i < 1; i++){
                        write_2_hw(0, (int)((10 << 26) + (((i*2)&0x1F) << 21) + (1 << 17) +
(info_001 << 14) + (ping_pong << 13) + (1 << 12) + (flip << 11) + (0 & 0x1F)));
                        write_2_hw(0, (int)((10 << 26) + (((i*2)&0x1F) << 21) + (1 << 17) +
(info_010 << 14) + (ping_pong << 13) + ((margin_width + 20 + (i%4) * 64) & 0x3FF)));
                        write_2_hw(0, (int)((10 << 26) + (((i*2)&0x1F) << 21) + (1 << 17) +
(info_011 << 14) + (ping_pong << 13) + (((int)(i/4)*64 + 250) & 0x3FF)));
                        ////////////////////////////////////
                        write_2_hw(0, (int)((10 << 26) + (((i*2 + 1)&0x1F) << 21) + (1 << 17) +
(info_001 << 14) + (ping_pong << 13) + (1 << 12) + (flip << 11) + (1 & 0x1F)));
                        write_2_hw(0, (int)((10 << 26) + (((i*2 + 1)&0x1F) << 21) + (1 << 17) +
(info_010 << 14) + (ping_pong << 13) + ((margin_width + 20 + (i%4) * 64) & 0x3FF)));
```

```
                write_2_hw(0, (int)((10 << 26) + (((i*2 + 1)&0x1F) << 21) + (1 << 17) +
(info_011 << 14) + (ping_pong << 13) + (((int)(i/4)*64 + 250) & 0x3FF)));
                // write_2_hw(0, (int)((3 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_100
<< 14) + (ping_pong << 13) + (0 & 0x3FF)));
            }

            //Goomba ========================
            for(int i = 0; i < 2; i++){
                write_2_hw(0, (int)((5 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_001
<< 14) + (ping_pong << 13) + (1 << 12) + (goomba_flip << 11) + (flip & 0x1F)));
                write_2_hw(0, (int)((5 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_010
<< 14) + (ping_pong << 13) + ((margin_width + 350 + (i%4) * 32) & 0x3FF)));
                write_2_hw(0, (int)((5 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_011
<< 14) + (ping_pong << 13) + (((int)(i/4)*64 + 180) & 0x3FF)));
                // write_2_hw(0, (int)((3 << 26) + ((i&0x1F) << 21) + (1 << 17) + (info_100
<< 14) + (ping_pong << 13) + (0 & 0x3FF)));
            }

            //Ground ========================
                write_2_hw(0, (int)((15 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_001
<< 14) + (ping_pong << 13) + (1 << 12) + (0 << 11) + (0 & 0x1F)));
                write_2_hw(0, (int)((15 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_010
<< 14) + (ping_pong << 13) + ((15 - (ground_shift%16)) & 0x3FF)));
                write_2_hw(0, (int)((15 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_011
<< 14) + (ping_pong << 13) + (400 & 0x3FF)));
                write_2_hw(0, (int)((15 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_100
<< 14) + (ping_pong << 13) + (450 & 0x3FF)));
            //Flush ========================
            write_2_hw(0, (int)(bus_info + (0xf << 17) + (ping_pong << 13)));
            ping_pong = ping_pong? 0 : 1;

            if(clk_count%pp_count == 0){
                pp = (pp >= 18)? 0 : (pp+1);
            }
            if(clk_count%coin_pp_count == 0){
                coin_pp = (coin_pp >= 3)? 0 : (coin_pp+1);
            }
            if(clk_count%goomba_flip_count == 0){
                goomba_flip = goomba_flip? 0 : 1;
            }
            if(clk_count%ground_count == 0){
                ground_shift  = (ground_shift >= 479)? 0 : ground_shift + 1;
            }
            clk_count++;
```

```
                usleep(25000);
        }

  printf("VGA BALL Userspace program terminating\n");
  return 0;
}
```

SuperMario_64.c:

```
/*
 * Game Logic
 * First Edied on April 15 2023 by Zhiyuan Liu
*/

#include <stdio.h>
#include <math.h>
#include "vga_ball.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdint.h>
#include <time.h>
#include "usbkeyboard.h"
#include <pthread.h>
#include "game_struct.h"
#include "game_animation.h"

#define PPU_ADDR 0

// Ground space
#define GROUND_0_L 448
#define GROUND_0_R 527
#define GROUND_1_L 1472
#define GROUND_1_R 1503
#define GROUND_2_L 2144
#define GROUND_2_R 2175
// Keyboard // joystic input
enum key_input{KEY_NONE, KEY_JUMP, KEY_LEFT, KEY_RIGHT, KEY_NEWGAME,
KEY_END};
```

```
enum key_input current_key;
// Sound Effect
#define SOUND_JUMP 0
#define SOUND_COIN 0
#define SOUND_BLOCK 0
#define SOUND_DEAD 0
#define SOUND_NONE 5
// Avalon bus file ind
int vga_ball_fd;

int info_001 = 1;
int info_010 = 2;
int info_011 = 3;
int info_100 = 4;

int block_r = 0;
int block_l = 0;
int frame_counter = 0;

int sound_new = 0;
int sound_ind = 0;


void write_2_hw(int addr, int info)
{
	vga_ball_arg_t vla;
	vla.addr = addr;
	vla.info = info;
	if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &vla)) {
		perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
		return;
	}
}

void flush_mario(const mario *mario_0, int ping_pong){
	int visible = mario_0->vis.visible;
	int flip = mario_0->vis.flip;
	int x = mario_0->vis.x;
	int y = mario_0->vis.y;
	int pp = mario_0->vis.pattern_code;
	write_2_hw(0, (int)((1 << 26) + (1 << 17) + (info_001 << 14) + (ping_pong << 13) +
(visible << 12) + (flip << 11) + (pp & 0x1F)));
	write_2_hw(0, (int)((1 << 26) + (1 << 17) + (info_010 << 14) + (ping_pong << 13) + (x &
0x3FF)));
```

```
	write_2_hw(0, (int)((1 << 26) + (1 << 17) + (info_011 << 14) + (ping_pong << 13) + (y &
0x3FF)));
}
void flush_mush(const mush *mush_0, int ping_pong){
	int visible = mush_0->vis.visible;
	int flip = mush_0->vis.flip;
	int x = mush_0->vis.x;
	int y = mush_0->vis.y;
	int pp = mush_0->vis.pattern_code;
	write_2_hw(0, (int)((9 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_001 << 14) +
(ping_pong << 13) + (1 << 12) + (0 << 11) + (pp & 0x1F)));
	write_2_hw(0, (int)((9 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_010 << 14) +
(ping_pong << 13) + (x & 0x3FF)));
	write_2_hw(0, (int)((9 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_011 << 14) +
(ping_pong << 13) + (y & 0x3FF)));
}
void flush_goomba(const goomba *goomba_0, int ind, int ping_pong){
	int visible = goomba_0->vis.visible;
	int flip = goomba_0->vis.flip;
	int x = goomba_0->vis.x;
	int y = goomba_0->vis.y;
	int pp = goomba_0->vis.pattern_code;
	write_2_hw(0, (int)((5 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_001 << 14) +
(ping_pong << 13) + (1 << 12) + (flip << 11) + (pp & 0x1F)));
	write_2_hw(0, (int)((5 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_010 << 14) +
(ping_pong << 13) + (x & 0x3FF)));
	write_2_hw(0, (int)((5 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_011 << 14) +
(ping_pong << 13) + (y & 0x3FF)));
}
void flush_block(const block *block_0, int ind, int ping_pong){
	int visible = block_0->vis.visible;
	int flip = block_0->vis.flip;
	int x = block_0->vis.x;
	int y = block_0->vis.y;
	int pp = block_0->vis.pattern_code;
	write_2_hw(0, (int)((2 << 26) + ((ind&0x1F) << 21) + (1 << 17) + (info_001 << 14) +
(ping_pong << 13) + (1 << 12) + (0 << 11) + (pp & 0x1F)));
	write_2_hw(0, (int)((2 << 26) + ((ind&0x1F) << 21) + (1 << 17) + (info_010 << 14) +
(ping_pong << 13) + (x & 0x3FF)));
	write_2_hw(0, (int)((2 << 26) + ((ind&0x1F) << 21) + (1 << 17) + (info_011 << 14) +
(ping_pong << 13) + (y & 0x3FF)));
}
// Flush the ground info into PPU
void flush_ground(int camera_pos, int ping_pong){
```

```
	int ground_r[3] = {GROUND_0_R, GROUND_1_R, GROUND_2_R};
	int ground_l[3] = {GROUND_0_L, GROUND_1_L, GROUND_2_L};
	int ground_screen_l = 0;
	int ground_screen_r = 0;
	for (int i = 0; i < GROUND_NUM; i++){
		if ((ground_l[i]<=camera_pos + CAMERA_SIZE + LOAD_LIMIT && ground_l[i] >=
camera_pos) ||
			(ground_r[i]<=camera_pos + CAMERA_SIZE + LOAD_LIMIT &&
ground_r[i] >= camera_pos)){
			ground_screen_r = (ground_r[i]-camera_pos <= CAMERA_SIZE +
LOAD_LIMIT)? ground_r[i]-camera_pos : CAMERA_SIZE + LOAD_LIMIT;
			ground_screen_l = (ground_l[i] >= camera_pos)? ground_l[i]-camera_pos
: 0;
			break;
		}
	}

	write_2_hw(0, (int)((15 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_001 << 14) +
(ping_pong << 13) + (1 << 12) + (0 << 11) + (0 & 0x1F)));
	write_2_hw(0, (int)((15 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_010 << 14) +
(ping_pong << 13) + ((15 - (camera_pos%16)) & 0x3FF)));
	write_2_hw(0, (int)((15 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_011 << 14) +
(ping_pong << 13) + (ground_screen_l & 0x3FF)));
	write_2_hw(0, (int)((15 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_100 << 14) +
(ping_pong << 13) + (ground_screen_r & 0x3FF)));
}
void flush_tube(const tube *tube_0, int ping_pong){
	int visible = tube_0->vis.visible;
	int flip = tube_0->vis.flip;
	int x = tube_0->vis.x;
	int y = tube_0->vis.y;

	write_2_hw(0, (int)((10 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_001 << 14) +
(ping_pong << 13) + (1 << 12) + (0 << 11) + (ANI_TUBE_H & 0x1F)));
	write_2_hw(0, (int)((10 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_010 << 14) +
(ping_pong << 13) + (x & 0x3FF)));
	write_2_hw(0, (int)((10 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_011 << 14) +
(ping_pong << 13) + (y & 0x3FF)));

	write_2_hw(0, (int)((10 << 26) + ((1&0x1F) << 21) + (1 << 17) + (info_001 << 14) +
(ping_pong << 13) + (1 << 12) + (0 << 11) + (ANI_TUBE_B & 0x1F)));
	write_2_hw(0, (int)((10 << 26) + ((1&0x1F) << 21) + (1 << 17) + (info_010 << 14) +
(ping_pong << 13) + (x & 0x3FF)));
```

```
	write_2_hw(0, (int)((10 << 26) + ((1&0x1F) << 21) + (1 << 17) + (info_011 << 14) +
(ping_pong << 13) + (y & 0x3FF)));
}
void flush_cloud(const cloud *cloud_0, int ping_pong){
	int visible = cloud_0->vis.visible;
	int flip = cloud_0->vis.flip;
	int x = cloud_0->vis.x;
	int y = cloud_0->vis.y;

	write_2_hw(0, (int)((14 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_001 << 14) +
(ping_pong << 13) + (1 << 12) + (0 << 11) + (ANI_CLOUD_NORMAL & 0x1F)));
	write_2_hw(0, (int)((14 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_010 << 14) +
(ping_pong << 13) + (x & 0x3FF)));
	write_2_hw(0, (int)((14 << 26) + ((0&0x1F) << 21) + (1 << 17) + (info_011 << 14) +
(ping_pong << 13) + (y & 0x3FF)));

}
void flush_coin(const coin *coin_0, int ind, int ping_pong){
	int visible = coin_0->vis.visible;
	int flip = coin_0->vis.flip;
	int x = coin_0->vis.x;
	int y = coin_0->vis.y;
	int pp = coin_0->vis.pattern_code;
	write_2_hw(0, (int)((3 << 26) + ((ind&0x1F) << 21) + (1 << 17) + (info_001 << 14) +
(ping_pong << 13) + (1 << 12) + (0 << 11) + (pp & 0x1F)));
	write_2_hw(0, (int)((3 << 26) + ((ind&0x1F) << 21) + (1 << 17) + (info_010 << 14) +
(ping_pong << 13) + (x & 0x3FF)));
	write_2_hw(0, (int)((3 << 26) + ((ind&0x1F) << 21) + (1 << 17) + (info_011 << 14) +
(ping_pong << 13) + (y & 0x3FF)));
}
void flush_ping_pong_frame(const mario_game *game_0, int ping_pong){

	int block_count = 0;
	int coin_count = 0;
	int mush_count = 0;
	int goomba_count = 0;
	int tube_count = 0;
	int cloud_count = 0;
	// Flush Mario
	mario_animation(game_0, &(game_0->mario_0), frame_counter);
	flush_mario(&(game_0->mario_0), ping_pong);

	// Flush Ground
	flush_ground(game_0->camera_pos, ping_pong);
```

```
        // Flush Block
        // ******LATER ADD A LIMITED NUMBER OF BLOCK
        for(int i = block_l; (i < block_r) && (block_count < BLOCK_VIS_LIMIT); i++){
                if(game_0->block_list[i].vis.visible == 1 && game_0->block_list[i].enable == 1){
                        block_animation(game_0, &(game_0->block_list[i]), frame_counter);
                        flush_block(&(game_0->block_list[i]), block_count, ping_pong);
                        block_count += 1;
                }
        }
        // Flush Coin
        for(int i = 0; (i < COIN_NUM) && (coin_count < COIN_VIS_LIMIT); i++){
                if(game_0->coin_list[i].enable == 1 && game_0->coin_list[i].hit.x >= game_0->
camera_pos
                && game_0->coin_list[i].hit.x <= game_0->camera_pos + CAMERA_SIZE +
LOAD_LIMIT){
                        coin_animation(game_0, &(game_0->coin_list[i]), frame_counter);
                        flush_coin(&(game_0->coin_list[i]), coin_count, ping_pong);
                        coin_count += 1;
                }
        }

        // Flush Mush
        for(int i = 0; (i < MUSH_NUM) && (mush_count < MUSH_VIS_LIMIT); i++){
                if(game_0->mush_list[i].enable == 1 && game_0->mush_list[i].hit.x >= game_0->
camera_pos
                && game_0->mush_list[i].hit.x <= game_0->camera_pos + CAMERA_SIZE +
LOAD_LIMIT){
                        mush_animation(game_0, &(game_0->mush_list[i]), frame_counter);
                        flush_mush(&(game_0->mush_list[i]), ping_pong);
                        mush_count += 1;
                }
        }

        // Flush Goomba
        for(int i = 0; (i < GOOMBA_NUM) && (goomba_count < GOOMBA_VIS_LIMIT); i++){
                if(game_0->goomba_list[i].enable == 1 && game_0->goomba_list[i].hit.x >=
game_0-> camera_pos
                && game_0->goomba_list[i].hit.x <= game_0->camera_pos + CAMERA_SIZE +
LOAD_LIMIT){
                        goomba_animation(game_0, &(game_0->goomba_list[i]), frame_counter);
                        flush_goomba(&(game_0->goomba_list[i]), goomba_count, ping_pong);
                        goomba_count += 1;
                }
```

```
        }

        // Flush Tube
        for(int i = 0; (i < TUBE_NUM) && (tube_count < TUBE_VIS_LIMIT); i++){
                if(game_0->tube_list[i].enable == 1 && game_0->tube_list[i].hit.x >= game_0->
camera_pos
                && game_0->tube_list[i].hit.x <= game_0->camera_pos + CAMERA_SIZE +
LOAD_LIMIT){
                        tube_animation(game_0, &(game_0->tube_list[i]), frame_counter);
                        flush_tube(&(game_0->tube_list[i]), ping_pong);
                        tube_count += 1;
                }
        }

        // Flush Cloud
        for(int i = 0; (i <  CLOUD_NUM) && (cloud_count < CLOUD_VIS_LIMIT); i++){
                if(game_0->cloud_list[i].enable == 1 && game_0->cloud_list[i].hit.x >= game_0->
camera_pos
                && game_0->cloud_list[i].hit.x <= game_0->camera_pos + CAMERA_SIZE +
LOAD_LIMIT){
                        cloud_animation(game_0, &(game_0->cloud_list[i]), frame_counter);
                        flush_cloud(&(game_0->cloud_list[i]), ping_pong);
                        cloud_count += 1;
                }
        }


        // Flush the current frame
        write_2_hw(0, (int)((1 << 26) + (0xf << 17) + (ping_pong << 13)));
        frame_counter = (frame_counter >= FRAME_LIMIT) ? 0 : frame_counter + 1;

        // Flush the sound effect
        if (sound_new == 1){
                write_2_hw(4, (int)(SOUND_NONE));
                sound_new = 0;
        }
        else{
                write_2_hw(4, (int)(sound_ind));
        }

}


//Keyboard
```

```
// input from device
	libusb_context *ctx = NULL; // a libusb session
	libusb_device **devs;	// pointer to pointer of device, used to retrieve a list of devices
	int r;			// for return values
	ssize_t cnt;		// holding number of devices in list
	struct libusb_device_handle *mouse;   // a mouse device handle


// Threads================
pthread_t input_thread;
pthread_t sound_thread;
void *input_thread_f(void *);
void *sound_thread_f(void *);


// Block ind updater==========
void block_ind_update(block *block_list, int camera_pos){
	int i;
	for (i = block_r; i < BLOCK_NUM; i++)
		if (block_list[i].hit.x > camera_pos + CAMERA_SIZE + LOAD_LIMIT)
			break;
	block_r = i;
	for (i = block_l; (i < block_r) && (i < BLOCK_NUM); i++){
		if (block_list[i].hit.x + block_list[i].hit.sx + 1 >= camera_pos + LOAD_LIMIT)
			break;
		block_list[i].loaded = 0;
	}
	block_l = i;
	for (i = block_l; i < block_r; i++)
		block_list[i].loaded = 1;
}

int main(){

	//Set up=======================
	int err, col;
	static const char filename[] = "/dev/vga_ball";
	int ping_pong = 0;
	block_r = 0;
	block_l = 0;

	// Game OBJ
	mario_game game_0;
```

```
	/* Open the keyboard */
	// if ( (keyboard = openkeyboard(&endpoint_address)) == NULL ) {
	//		fprintf(stderr, "Did not find a keyboard\n");
	//		exit(1);
	// }

	r = libusb_init(&ctx);	// initialize a library session
	if (r < 0)
	{
	printf("%s  %d\n", "Init Error", r); // there was an error
	return 1;
	}
	libusb_set_debug(ctx, 3);		// set verbosity level to 3, as suggested in the
documentation
	cnt = libusb_get_device_list(ctx, &devs); // get the list of devices
	if (cnt < 0)
	{
	printf("%s\n", "Get Device Error"); // there was an error
	}
	mouse = libusb_open_device_with_vid_pid(ctx, 0x0079, 0x0011);
	if (mouse == NULL)
	{
		printf("%s\n", "Cannot open device");
		libusb_free_device_list(devs, 1); // free the list, unref the devices in it
		libusb_exit(ctx);		// close the session
		return 0;
	}
	else
	{
		printf("%s\n", "Device opened");
		libusb_free_device_list(devs, 1); // free the list, unref the devices in it
		if (libusb_kernel_driver_active(mouse, 0) == 1)
		{ // find out if kernel driver is attached
			printf("%s\n", "Kernel Driver Active");
			if (libusb_detach_kernel_driver(mouse, 0) == 0) // detach it
		   printf("%s\n", "Kernel Driver Detached!");
		}
		r = libusb_claim_interface(mouse, 0); // claim interface 0 (the first) of device
(mine had just 1)
		if (r < 0)
		{
			printf("%s\n", "Cannot Claim Interface");
			return 1;
		}
```

```
	}
	printf("%s\n", "Claimed Interface");

	// Open Avalon bus
	if ( (vga_ball_fd = open(filename, O_RDWR)) == -1) {
		fprintf(stderr, "could not open %s\n", filename);
		return -1;
	}

	////////////////////////////////////
	/* Start the network thread */
	pthread_create(&input_thread, NULL, input_thread_f, NULL);
	//pthread_create(&longpress_thread, NULL, longpress_thread_f, NULL);


	// Start a new game
	new_game(&game_0);
	// loop to find all in-screen element first
	// ================================
	while (1){

		//  new game if Mario is dead
		if (game_0.mario_0.mario_s == DEAD || current_key == KEY_NEWGAME)
{new_game(&game_0); block_r = 0; block_l = 0;}

		// loop to include all the avaliable block / entities / mario
		block_ind_update(&(game_0.block_list[0]), game_0.camera_pos);
		// using LR ind / xy for entity
			//entity, tube, cloud, all included now

		//================================

		// apply acceleration to all entity
		game_0.mario_0.acc_y = GRAVITY;
		if(game_0.mario_0.control_s == MARIO_NORMAL){
			if(game_0.mario_0.y_block == 1){
				switch (current_key){
					case KEY_LEFT:
						game_0.mario_0.acc_x = -WALK_ACC;
						game_0.mario_0.vis.flip = 1;
						break;
					case KEY_RIGHT:
						game_0.mario_0.acc_x = WALK_ACC;
						game_0.mario_0.vis.flip = 0;
```

```
                                        break;
                                    case KEY_JUMP:
                                        game_0.mario_0.hit.vy =
(game_0.mario_0.mario_s == SMALL)? -JUMP_INIT_V_SMALL : -JUMP_INIT_V_LARGE;
                                        game_0.mario_0.acc_x = 0;
                                        game_0.mario_0.hit.vx *= 0.8;
                                        sound_ind = SOUND_JUMP;
                                        sound_new = 1;
                                        break;
                                    // Shut down
                                    default:
                                        game_0.mario_0.hit.vx =
(fabs(game_0.mario_0.hit.vx) > 2 * SHUT_ACC)?

(game_0.mario_0.hit.vx > 0)? game_0.mario_0.hit.vx - SHUT_ACC: game_0.mario_0.hit.vx +
SHUT_ACC
                                                                                    : 0;
                                        game_0.mario_0.acc_x = 0;
                                }
                            }
                            //apply acceleration / volicy / input() jump to mario
                            //base on the grounded / block x/ block y state from last frame
                            game_0.mario_0.hit.vy = (game_0.mario_0.hit.vy +
game_0.mario_0.acc_y > MAX_SPEED_V) ? MAX_SPEED_V :
                                                        (game_0.mario_0.hit.vy +
game_0.mario_0.acc_y < -MAX_SPEED_V_JUMP) ? -MAX_SPEED_V_JUMP:
                                                        game_0.mario_0.hit.vy +
game_0.mario_0.acc_y;
                            game_0.mario_0.hit.vx = (game_0.mario_0.hit.vx +
game_0.mario_0.acc_x > MAX_SPEED_H) ? MAX_SPEED_H :
                                                        (game_0.mario_0.hit.vx +
game_0.mario_0.acc_x < -MAX_SPEED_H) ? -MAX_SPEED_H:
                                                        game_0.mario_0.hit.vx +
game_0.mario_0.acc_x;
                        }

                        // Mush==============================

                        for (int i = 0; i < MUSH_NUM; i++){
                            if (game_0.mush_list[i].enable == 1 && game_0.mush_list[i].mush_s ==
MUSH_NORMAL){
                                    if (game_0.mush_list[i].hit.y > 500)
game_0.mush_list[i].enable = 0;
                                    else{
```

```
                                        game_0.mush_list[i].acc_y = GRAVITY;

                                        if(game_0.mush_list[i].x_block != 0)
game_0.mush_list[i].vis.extra_info = (game_0.mush_list[i].x_block == 1)?

                                                                -1 : 1;
                                        // Fall?
                                        if (game_0.mush_list[i].y_block == 1)
game_0.mush_list[i].hit.vx = MAX_SPEED_H * 0.5 * game_0.mush_list[i].vis.extra_info;
                                        else game_0.mush_list[i].hit.vx = 0;

                                        game_0.mush_list[i].hit.vy =
(game_0.mush_list[i].hit.vy + game_0.mush_list[i].acc_y > MAX_SPEED_V) ? MAX_SPEED_V :

game_0.mush_list[i].hit.vy + game_0.mush_list[i].acc_y;
                                }
                        }

                }
                //================================

                // Goomba==============================

                for (int i = 0; i < GOOMBA_NUM; i++){
                        if (game_0.goomba_list[i].enable == 1 &&
game_0.goomba_list[i].goomba_s == GOOMBA_NORMAL){
                                if (game_0.goomba_list[i].hit.y > 500)
game_0.goomba_list[i].enable = 0;
                                else{
                                        game_0.goomba_list[i].acc_y = GRAVITY;

                                        if(game_0.goomba_list[i].x_block != 0 ||
game_0.goomba_list[i].hit.x > game_0.goomba_list[i].r_limit || game_0.goomba_list[i].hit.x <
game_0.goomba_list[i].l_limit) game_0.goomba_list[i].vis.extra_info =
(game_0.goomba_list[i].hit.x > game_0.goomba_list[i].r_limit || game_0.goomba_list[i].x_block
== 1)?

                                                                -1 : 1;
                                        // Fall?
                                        if (game_0.goomba_list[i].y_block == 1)
game_0.goomba_list[i].hit.vx = MAX_SPEED_H * 0.5 * game_0.goomba_list[i].vis.extra_info;
                                        else game_0.goomba_list[i].hit.vx = 0;
```

```
                                                    game_0.goomba_list[i].hit.vy =
(game_0.goomba_list[i].hit.vy + game_0.goomba_list[i].acc_y > MAX_SPEED_V) ?
MAX_SPEED_V :

game_0.goomba_list[i].hit.vy + game_0.goomba_list[i].acc_y;
                                                }
                        }

                }
                //================================

                // Apply hit check to all entity
                // update state / hit / animation
                enum contact result;
                enum contact result2;
                // Coin
                for (int i = 0; i < COIN_NUM; i++){
                        if(game_0.coin_list[i].enable == 1 && game_0.coin_list[i].hit.x >=
game_0.camera_pos
                        && game_0.coin_list[i].hit.x <= game_0.camera_pos + CAMERA_SIZE +
LOAD_LIMIT
                        && game_0.coin_list[i].coin_s == COIN_NORMAL){
                                result = hitbox_contact(&(game_0.coin_list[i].hit),
&(game_0.mario_0.hit));
                                if (result != NONE){
                                        game_0.coin_list[i].coin_s = COIN_ANIMATE;
                                        game_0.coin_list[i].animate_frame_counter =
frame_counter;
                                        sound_ind = SOUND_COIN;
                                        sound_new = 1;
                                }
                        }
                }

                // Mush
                for (int i = 0; i < MUSH_NUM; i++){
                        if(game_0.mush_list[i].enable == 1 && game_0.mush_list[i].hit.x >=
game_0.camera_pos
                        && game_0.mush_list[i].hit.x <= game_0.camera_pos + CAMERA_SIZE
+ LOAD_LIMIT
                        && game_0.mush_list[i].mush_s == MUSH_NORMAL){
                                result = hitbox_contact(&(game_0.mush_list[i].hit),
&(game_0.mario_0.hit));
                                if (result != NONE){
```

```
                                        game_0.mush_list[i].enable = 0;
                                        game_0.mario_0.control_s = MARIO_ANIMATE;
                                        game_0.mario_0.animate_s = ENLARGE;
                                        game_0.mario_0.animate_frame_counter =
frame_counter;
                                }
                        }
                }

                // Goomba
                for (int i = 0; i < GOOMBA_NUM; i++){
                        if(game_0.goomba_list[i].enable == 1 && game_0.goomba_list[i].hit.x >=
game_0.camera_pos
                        && game_0.goomba_list[i].hit.x <= game_0.camera_pos +
CAMERA_SIZE + LOAD_LIMIT - 8
                        && game_0.goomba_list[i].goomba_s == GOOMBA_NORMAL){
                                result = hitbox_contact(&(game_0.goomba_list[i].hit),
&(game_0.mario_0.hit));
                                result2 = hitbox_contact(&(game_0.mario_0.hit),
&(game_0.goomba_list[i].hit));
                                if (result != NONE || result2 != NONE){
                                        // Kill Goomba
                                        if (result == UP || result2 == DOWN){
                                                game_0.goomba_list[i].goomba_s =
GOOMBA_ANIMATE;
                                                game_0.goomba_list[i].animate_frame_counter =
frame_counter;
                                                // Jump again
                                                game_0.mario_0.hit.vy =
(game_0.mario_0.mario_s == SMALL)? -JUMP_INIT_V_SMALL : -JUMP_INIT_V_LARGE;
                                        }
                                        // Damage to Mario
                                        else if (game_0.mario_0.game_s ==
MARIO_GAME_NORMAL){
                                                // Dead
                                                if (game_0.mario_0.mario_s == SMALL){
                                                        game_0.mario_0.control_s =
MARIO_ANIMATE;
                                                        game_0.mario_0.animate_s = ANI_DEAD;
                                                        game_0.mario_0.animate_frame_counter =
frame_counter;
                                                        sound_ind = SOUND_DEAD;
                                                        sound_new = 1;
                                                }
```

```
                                        // Return to SMALL
                                        else{
                                                game_0.mario_0.control_s =
MARIO_ANIMATE;
                                                game_0.mario_0.animate_s = HIT;
                                                game_0.mario_0.animate_frame_counter =
frame_counter;
                                        }
                                }
                        }
                }
        }

        //================================

        // do the animation


        //======================
        //**** clear all the hit / block state first *******
        if(game_0.mario_0.control_s == MARIO_NORMAL){
                game_0.mario_0.x_block = 0; game_0.mario_0.y_block = 0;
                // apply hit check on the ground
                for (int i = 0; i < GROUND_NUM; i++){
                        result = hitbox_contact(&(game_0.mario_0.hit),
&(game_0.ground_list[i].hit));
                        if (result == LEFT){
                                game_0.mario_0.x_block = -1;
                                game_0.mario_0.hit.vx = 0;
                                game_0.mario_0.hit.x = game_0.ground_list[i].hit.x +
game_0.ground_list[i].hit.sx;
                        }
                        else if (result == RIGHT){
                                game_0.mario_0.x_block = 1;
                                game_0.mario_0.hit.vx = 0;
                                game_0.mario_0.hit.x = game_0.ground_list[i].hit.x -
game_0.mario_0.hit.sx;
                        }
                        else if (result == UP){
                                game_0.mario_0.y_block = -1;
                                game_0.mario_0.hit.vy = 0;
                                game_0.mario_0.hit.y = game_0.ground_list[i].hit.y +
game_0.ground_list[i].hit.sy;
                        }
```

```
                        else if (result == DOWN){
                                game_0.mario_0.y_block = 1;
                                game_0.mario_0.hit.vy = 0;
                                game_0.mario_0.hit.y = game_0.ground_list[i].hit.y -
game_0.mario_0.hit.sy;
                        }
                }

                // apply hit check on the Tube
                for (int i = 0; i < TUBE_NUM; i++){
                        if (game_0.tube_list[i].enable == 1){
                                result = hitbox_contact(&(game_0.mario_0.hit),
&(game_0.tube_list[i].hit));
                                if (result == LEFT){
                                        game_0.mario_0.x_block = -1;
                                        game_0.mario_0.hit.vx = 0;
                                        game_0.mario_0.hit.x = game_0.tube_list[i].hit.x +
game_0.tube_list[i].hit.sx;
                                }
                                else if (result == RIGHT){
                                        game_0.mario_0.x_block = 1;
                                        game_0.mario_0.hit.vx = 0;
                                        game_0.mario_0.hit.x = game_0.tube_list[i].hit.x -
game_0.mario_0.hit.sx;
                                }
                                else if (result == UP){
                                        game_0.mario_0.y_block = -1;
                                        game_0.mario_0.hit.vy = 0;
                                        game_0.mario_0.hit.y = game_0.tube_list[i].hit.y +
game_0.tube_list[i].hit.sy;
                                }
                                else if (result == DOWN){
                                        game_0.mario_0.y_block = 1;
                                        game_0.mario_0.hit.vy = 0;
                                        game_0.mario_0.hit.y = game_0.tube_list[i].hit.y -
game_0.mario_0.hit.sy;
                                }
                        }
                }

                //Hit the block
                for(int i = block_l; i < block_r; i++){
                        if (game_0.block_list[i].enable == 1){
                                // NEED TO CHECK THE SIZE *********
```

```
                                        if (game_0.mario_0.hit.sy > game_0.block_list[i].hit.sy){
                                                result = hitbox_contact(&(game_0.block_list[i].hit),
&(game_0.mario_0.hit));
                                                switch (result){
                                                        case LEFT: result = RIGHT; break;
                                                        case RIGHT: result = LEFT; break;
                                                        default: result =
hitbox_contact(&(game_0.mario_0.hit), &(game_0.block_list[i].hit));
                                                }
                                        }
                                        else result = hitbox_contact(&(game_0.mario_0.hit),
&(game_0.block_list[i].hit));

                                        if (result == LEFT && game_0.block_list[i].R == 0){
                                                game_0.mario_0.x_block = -1;
                                                game_0.mario_0.hit.vx = 0;
                                                game_0.mario_0.hit.x = game_0.block_list[i].hit.x +
game_0.block_list[i].hit.sx;
                                        }
                                        else if (result == RIGHT && game_0.block_list[i].L == 0){
                                                game_0.mario_0.x_block = 1;
                                                game_0.mario_0.hit.vx = 0;
                                                game_0.mario_0.hit.x = game_0.block_list[i].hit.x -
game_0.mario_0.hit.sx;
                                        }
                                        // Head on the Block, do animation
                                        else if (result == UP  && game_0.block_list[i].D == 0){
                                                game_0.mario_0.y_block = -1;
                                                game_0.mario_0.hit.vy = game_0.mario_0.hit.vy *
-0.5;
                                                game_0.mario_0.hit.y = game_0.block_list[i].hit.y +
game_0.block_list[i].hit.sy;
                                                // Breaking the block ******* if mario game_s is
LARGE
                                                if (game_0.block_list[i].block_t == TYPE_A &&
game_0.block_list[i].block_s == BLOCK_NORMAL && game_0.mario_0.mario_s == LARGE){
                                                        for(int j = block_l; j < block_r; j++){
                                                                if (game_0.block_list[j].enable == 1){
                                                                        // For block on LEFT, enable
R
                                                                        if (game_0.block_list[j].hit.x
== game_0.block_list[i].hit.x - 16) game_0.block_list[j].R = 0;
                                                                        // For block on RIGHT,
enable L
```

```
                                                    else if
(game_0.block_list[j].hit.x == game_0.block_list[i].hit.x + 16) game_0.block_list[j].L = 0;
                                                    // For block on UP, enable D
                                                    else if
(game_0.block_list[j].hit.y == game_0.block_list[i].hit.y - 16) game_0.block_list[j].D = 0;
                                                }
                                            }
                                            game_0.block_list[i].block_s =
BLOCK_ANIMATE;
                                            game_0.block_list[i].animate_frame_counter
= frame_counter;
                                            sound_ind = SOUND_BLOCK;
                                            sound_new = 1;
                                        }
                                        // Hitting an coin block
                                        if (game_0.block_list[i].block_t == OBJ_C &&
game_0.block_list[i].block_s == BLOCK_NORMAL){
                                            game_0.block_list[i].block_s =
BLOCK_ANIMATE;
                                            game_0.block_list[i].animate_frame_counter
= frame_counter;
                                            // generate an extra item
                                            game_0.coin_list[COIN_NUM-1].hit.x =
game_0.block_list[i].hit.x + 4;
                                            game_0.coin_list[COIN_NUM-1].hit.y =
game_0.block_list[i].hit.y;
                                            game_0.coin_list[COIN_NUM-1].enable = 1;
                                            game_0.coin_list[COIN_NUM-1].loaded = 1;
                                            game_0.coin_list[COIN_NUM-1].coin_s =
COIN_ANIMATE;

game_0.coin_list[COIN_NUM-1].animate_frame_counter = frame_counter;
                                        }
                                        // Hitting an mush block
                                        if (game_0.block_list[i].block_t == OBJ_M &&
game_0.block_list[i].block_s == BLOCK_NORMAL){
                                            game_0.block_list[i].block_s =
BLOCK_ANIMATE;
                                            game_0.block_list[i].animate_frame_counter
= frame_counter;
                                            // generate an extra item
                                            game_0.mush_list[MUSH_NUM-1].hit.x =
game_0.block_list[i].hit.x;
```

```
                                                    game_0.mush_list[MUSH_NUM-1].hit.y =
game_0.block_list[i].hit.y;
                                                    game_0.mush_list[MUSH_NUM-1].enable =
1;
                                                    game_0.mush_list[MUSH_NUM-1].loaded =
1;
                                                    game_0.mush_list[MUSH_NUM-1].mush_s
= MUSH_ANIMATE;

game_0.mush_list[MUSH_NUM-1].animate_frame_counter = frame_counter;

game_0.mush_list[MUSH_NUM-1].vis.extra_info = 1;
                                                }
                                        }
                                        else if (result == DOWN && game_0.block_list[i].U == 0){
                                                game_0.mario_0.y_block = 1;
                                                game_0.mario_0.hit.vy = 0;
                                                game_0.mario_0.hit.y = game_0.block_list[i].hit.y -
game_0.mario_0.hit.sy;
                                        }
                                }
                        }
                        // update position
                        game_0.mario_0.hit.x += game_0.mario_0.hit.vx; game_0.mario_0.hit.y
+= game_0.mario_0.hit.vy;
                        // On the edge:
                        if (game_0.mario_0.hit.x <= game_0.camera_pos + LOAD_LIMIT) {
                                game_0.mario_0.hit.x = game_0.camera_pos + LOAD_LIMIT;
                                game_0.mario_0.hit.vx = (game_0.mario_0.hit.vx < 0)? 0 :
game_0.mario_0.hit.vx;
                                }
                        if (game_0.mario_0.hit.y >= 384) {
                                game_0.mario_0.control_s = MARIO_ANIMATE;
                                game_0.mario_0.animate_s = ANI_DEAD;
                                game_0.mario_0.animate_frame_counter = frame_counter;
                                sound_ind = SOUND_DEAD;
                                sound_new = 1;
                        }
                }

                // MUSH ================================
                for (int j = 0; j < MUSH_NUM; j++){
                        if (game_0.mush_list[j].enable == 1 && game_0.mush_list[j].mush_s ==
MUSH_NORMAL){
```

```
                        game_0.mush_list[j].x_block = 0; game_0.mush_list[j].y_block = 0;
                        for (int i = 0; i < GROUND_NUM; i++){
                                result = hitbox_contact(&(game_0.mush_list[j].hit),
&(game_0.ground_list[i].hit));
                                if (result == LEFT){
                                        game_0.mush_list[j].x_block = -1;
                                        game_0.mush_list[j].hit.vx = 0;
                                        game_0.mush_list[j].hit.x =
game_0.ground_list[i].hit.x + game_0.ground_list[i].hit.sx;
                                }
                                else if (result == RIGHT){
                                        game_0.mush_list[j].x_block = 1;
                                        game_0.mush_list[j].hit.vx = 0;
                                        game_0.mush_list[j].hit.x =
game_0.ground_list[i].hit.x - game_0.mush_list[j].hit.sx;
                                }
                                else if (result == DOWN){
                                        game_0.mush_list[j].y_block = 1;
                                        game_0.mush_list[j].hit.vy = 0;
                                        game_0.mush_list[j].hit.y =
game_0.ground_list[i].hit.y - game_0.mush_list[j].hit.sy;
                                }
                        }

                        for (int i = 0; i < TUBE_NUM; i++){
                                if (game_0.tube_list[i].enable == 1){
                                        result = hitbox_contact(&(game_0.mush_list[j].hit),
&(game_0.tube_list[i].hit));
                                        if (result == LEFT){
                                                game_0.mush_list[j].x_block = -1;
                                                game_0.mush_list[j].hit.vx = 0;
                                                game_0.mush_list[j].hit.x =
game_0.tube_list[i].hit.x + game_0.tube_list[i].hit.sx;
                                        }
                                        else if (result == RIGHT){
                                                game_0.mush_list[j].x_block = 1;
                                                game_0.mush_list[j].hit.vx = 0;
                                                game_0.mush_list[j].hit.x =
game_0.tube_list[i].hit.x - game_0.mush_list[j].hit.sx;
                                        }
                                        else if (result == DOWN){
                                                game_0.mush_list[j].y_block = 1;
                                                game_0.mush_list[j].hit.vy = 0;
```

```
                                                game_0.mush_list[j].hit.y =
game_0.tube_list[i].hit.y - game_0.mush_list[j].hit.sy;
                                        }
                                }
                        }

                        for(int i = block_l; i < block_r; i++){
                                if (game_0.block_list[i].enable == 1){
                                        result = hitbox_contact(&(game_0.mush_list[j].hit),
&(game_0.block_list[i].hit));
                                        if (result == LEFT && game_0.block_list[i].R == 0){
                                                game_0.mush_list[j].x_block = -1;
                                                game_0.mush_list[j].hit.vx = 0;
                                                game_0.mush_list[j].hit.x =
game_0.block_list[i].hit.x + game_0.block_list[i].hit.sx;
                                        }
                                        else if (result == RIGHT && game_0.block_list[i].L
== 0){
                                                game_0.mush_list[j].x_block = 1;
                                                game_0.mush_list[j].hit.vx = 0;
                                                game_0.mush_list[j].hit.x =
game_0.block_list[i].hit.x - game_0.mush_list[j].hit.sx;
                                        }
                                        else if (result == DOWN && game_0.block_list[i].U
== 0){
                                                game_0.mush_list[j].y_block = 1;
                                                game_0.mush_list[j].hit.vy = 0;
                                                game_0.mush_list[j].hit.y =
game_0.block_list[i].hit.y - game_0.mush_list[j].hit.sy;
                                        }
                                }
                        }
                        // update position
                        game_0.mush_list[j].hit.x += game_0.mush_list[j].hit.vx;
game_0.mush_list[j].hit.y += game_0.mush_list[j].hit.vy;
                }
        }
        //
GOOMBA=====================================================
        for (int j = 0; j < GOOMBA_NUM; j++){
                if (game_0.goomba_list[j].enable == 1 &&
game_0.goomba_list[j].goomba_s == GOOMBA_NORMAL &&
                        game_0.goomba_list[j].hit.x >= game_0.camera_pos &&
```

```
                        game_0.goomba_list[j].hit.x <= game_0.camera_pos +
CAMERA_SIZE + LOAD_LIMIT){
                        game_0.goomba_list[j].x_block = 0;
game_0.goomba_list[j].y_block = 0;
                        for (int i = 0; i < GROUND_NUM; i++){
                                result = hitbox_contact(&(game_0.goomba_list[j].hit),
&(game_0.ground_list[i].hit));
                                if (result == LEFT){
                                        game_0.goomba_list[j].x_block = -1;
                                        game_0.goomba_list[j].hit.vx = 0;
                                        game_0.goomba_list[j].hit.x =
game_0.ground_list[i].hit.x + game_0.ground_list[i].hit.sx;
                                }
                                else if (result == RIGHT){
                                        game_0.goomba_list[j].x_block = 1;
                                        game_0.goomba_list[j].hit.vx = 0;
                                        game_0.goomba_list[j].hit.x =
game_0.ground_list[i].hit.x - game_0.goomba_list[j].hit.sx;
                                }
                                else if (result == DOWN){
                                        game_0.goomba_list[j].y_block = 1;
                                        game_0.goomba_list[j].hit.vy = 0;
                                        game_0.goomba_list[j].hit.y =
game_0.ground_list[i].hit.y - game_0.goomba_list[j].hit.sy;
                                }
                        }

                        for (int i = 0; i < TUBE_NUM; i++){
                                if (game_0.tube_list[i].enable == 1){
                                        result =
hitbox_contact(&(game_0.goomba_list[j].hit), &(game_0.tube_list[i].hit));
                                        if (result == LEFT){
                                                game_0.goomba_list[j].x_block = -1;
                                                game_0.goomba_list[j].hit.vx = 0;
                                                game_0.goomba_list[j].hit.x =
game_0.tube_list[i].hit.x + game_0.tube_list[i].hit.sx;
                                        }
                                        else if (result == RIGHT){
                                                game_0.goomba_list[j].x_block = 1;
                                                game_0.goomba_list[j].hit.vx = 0;
                                                game_0.goomba_list[j].hit.x =
game_0.tube_list[i].hit.x - game_0.goomba_list[j].hit.sx;
                                        }
                                        else if (result == DOWN){
```

```
                                        game_0.goomba_list[j].y_block = 1;
                                        game_0.goomba_list[j].hit.vy = 0;
                                        game_0.goomba_list[j].hit.y =
game_0.tube_list[i].hit.y - game_0.goomba_list[j].hit.sy;
                                    }
                                }
                            }

                            for(int i = block_l; i < block_r; i++){
                                if (game_0.block_list[i].enable == 1){
                                    // NEED TO CHECK THE SIZE *********
                                    if (game_0.goomba_list[j].hit.sy >
game_0.block_list[i].hit.sy){
                                        result =
hitbox_contact(&(game_0.block_list[i].hit), &(game_0.goomba_list[j].hit));
                                        switch (result){
                                            case LEFT: result = RIGHT; break;
                                            case RIGHT: result = LEFT; break;
                                            default: result =
hitbox_contact(&(game_0.goomba_list[j].hit), &(game_0.block_list[i].hit));
                                        }
                                    }
                                    else result =
hitbox_contact(&(game_0.goomba_list[j].hit), &(game_0.block_list[i].hit));
                                    if (result == LEFT && game_0.block_list[i].R == 0){
                                        game_0.goomba_list[j].x_block = -1;
                                        game_0.goomba_list[j].hit.vx = 0;
                                        game_0.goomba_list[j].hit.x =
game_0.block_list[i].hit.x + game_0.block_list[i].hit.sx;
                                    }
                                    else if (result == RIGHT && game_0.block_list[i].L
== 0){
                                        game_0.goomba_list[j].x_block = 1;
                                        game_0.goomba_list[j].hit.vx = 0;
                                        game_0.goomba_list[j].hit.x =
game_0.block_list[i].hit.x - game_0.goomba_list[j].hit.sx;
                                    }
                                    else if (result == DOWN && game_0.block_list[i].U
== 0){
                                        game_0.goomba_list[j].y_block = 1;
                                        game_0.goomba_list[j].hit.vy = 0;
                                        game_0.goomba_list[j].hit.y =
game_0.block_list[i].hit.y - game_0.goomba_list[j].hit.sy;
                                    }
```

```
                                        }
                                    }
                                    // update position
                                    game_0.goomba_list[j].hit.x += game_0.goomba_list[j].hit.vx;
game_0.goomba_list[j].hit.y += game_0.goomba_list[j].hit.vy;
                                }
                        }

                        // update block x block y state (and update x, y position)
                        // modify the velocity of all entity
                        // update position
                        //============================
                        // update the camera position
                        if(((int)game_0.mario_0.hit.x) > game_0.camera_pos + CAMERA_SIZE/2 +
LOAD_LIMIT){
                                game_0.camera_pos = (((int)game_0.mario_0.hit.x) - CAMERA_SIZE/2 -
LOAD_LIMIT > 0)?
                                                                ((int)game_0.mario_0.hit.x) -
CAMERA_SIZE/2 - LOAD_LIMIT : 0;
                        }
                        // generate all the ppu info
                        // flush the frame into ppu
                        //test=======================
                        flush_ping_pong_frame(&game_0, ping_pong);
                        ping_pong = (ping_pong == 0)? 1 : 0;
                        usleep(25000);
                }

                pthread_cancel(input_thread);
                pthread_join(input_thread, NULL);

                return 0;
}

void *input_thread_f(void *ignored)
{
                unsigned char buff[64];
   int size = 8;
   libusb_interrupt_transfer(mouse, 0x81, buff, 0x0008, &size, 0);
                for (;;) {
                        // libusb_interrupt_transfer(keyboard, endpoint_address,
                        //                         (unsigned char *) &packet, sizeof(packet),
                        //                         &transferred, 0);
                        // if (transferred == sizeof(packet)) {
```

```
		//		if (packet.keycode[0] == 0x0){current_key = KEY_NONE;
printf("NONE\n");}
		//		else
		//			for(int i = 0; i < 3; i++){
		//				if (packet.keycode[i] == 0x2c) {current_key = KEY_JUMP;
printf("JUMP\n");}
		//				else if (packet.keycode[i] == 0x50) {current_key =
KEY_LEFT; printf("LEFT\n");}
		//				else if (packet.keycode[i] == 0x4F) {current_key =
KEY_RIGHT; printf("RIGHT\n");}
		//			}
		//		if (packet.keycode[0] == 0x29) { /* ESC pressed? */
		//			break;
		//		}
		// }
		size = 8;
		libusb_interrupt_transfer(mouse, 0x81, buff, 0x0008, &size, 0);
		if (size == 0x0008){

			if (buff[5] == 47) {
				// A:	127 127 0 128 128 47
				current_key = KEY_JUMP; // printf("JUMP\n");
			}
			else if (buff[3] == 0) {
				// 127 127 0 128 128 15
				// left
				current_key = KEY_LEFT; // printf("LEFT\n");
			}
			else if (buff[3] == 255) {
				// right
				// 127 127 255 128 128 15
				current_key = KEY_RIGHT; // printf("RIGHT\n");
			}
			else if (buff[6] == 32) {
				// restart
				// 127 127 127 127 127 15 32
				current_key = KEY_NEWGAME; // printf("KEY_NEWGAME\n");

			}
			else{
				current_key = KEY_NONE; // printf("NONE\n");
			}
		}
	}
```

```
        return NULL;
}
```

Usbkeyboard.c:

```
#include "usbkeyboard.h"

#include <stdio.h>
#include <stdlib.h>

/* References on libusb 1.0 and the USB HID/keyboard protocol
 *
 * http://libusb.org
 * http://www.dreamincode.net/forums/topic/148707-introduction-to-using-libusb-10/
 * http://www.usb.org/developers/devclass_docs/HID1_11.pdf
 * http://www.usb.org/developers/devclass_docs/Hut1_11.pdf
 */

/*
 * Find and return a USB keyboard device or NULL if not found
 * The argument con
 *
 */
struct libusb_device_handle *openkeyboard(uint8_t *endpoint_address) {
  libusb_device **devs;
  struct libusb_device_handle *keyboard = NULL;
  struct libusb_device_descriptor desc;
  ssize_t num_devs, d;
  uint8_t i, k;

  /* Start the library */
  if ( libusb_init(NULL) < 0 ) {
    fprintf(stderr, "Error: libusb_init failed\n");
    exit(1);
  }

  /* Enumerate all the attached USB devices */
  if ( (num_devs = libusb_get_device_list(NULL, &devs)) < 0 ) {
    fprintf(stderr, "Error: libusb_get_device_list failed\n");
    exit(1);
  }

  /* Look at each device, remembering the first HID device that speaks
```

```
    the keyboard protocol */

  for (d = 0 ; d < num_devs ; d++) {
    libusb_device *dev = devs[d];
    if ( libusb_get_device_descriptor(dev, &desc) < 0 ) {
      fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");
      exit(1);
    }

    if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {
      struct libusb_config_descriptor *config;
      libusb_get_config_descriptor(dev, 0, &config);
      for (i = 0 ; i < config->bNumInterfaces ; i++)
           for ( k = 0 ; k < config->interface[i].num_altsetting ; k++ ) {
             const struct libusb_interface_descriptor *inter =
               config->interface[i].altsetting + k ;
             if ( inter->bInterfaceClass == LIBUSB_CLASS_HID &&
                  inter->bInterfaceProtocol == USB_HID_KEYBOARD_PROTOCOL) {
               int r;
               if ((r = libusb_open(dev, &keyboard)) != 0) {
                 fprintf(stderr, "Error: libusb_open failed: %d\n", r);
                 exit(1);
               }
               if (libusb_kernel_driver_active(keyboard,i))
                 libusb_detach_kernel_driver(keyboard, i);
               libusb_set_auto_detach_kernel_driver(keyboard, i);
               if ((r = libusb_claim_interface(keyboard, i)) != 0) {
                 fprintf(stderr, "Error: libusb_claim_interface failed: %d\n", r);
                 exit(1);
               }
               *endpoint_address = inter->endpoint[0].bEndpointAddress;
               goto found;
             }
           }
    }
  }

 found:
  libusb_free_device_list(devs, 1);

  return keyboard;
}
```

Usbkeyboard.h:

```
#ifndef _USBKEYBOARD_H
#define _USBKEYBOARD_H

#include <libusb-1.0/libusb.h>

#define USB_HID_KEYBOARD_PROTOCOL 1

/* Modifier bits */
#define USB_LCTRL  (1 << 0)
#define USB_LSHIFT (1 << 1)
#define USB_LALT   (1 << 2)
#define USB_LGUI   (1 << 3)
#define USB_RCTRL  (1 << 4)
#define USB_RSHIFT (1 << 5)
#define USB_RALT   (1 << 6)
#define USB_RGUI   (1 << 7)

struct usb_keyboard_packet {
  uint8_t modifiers;
  uint8_t reserved;
  uint8_t keycode[6];
};

/* Find and open a USB keyboard device.  Argument should point to
   space to store an endpoint address.  Returns NULL if no keyboard
   device was found. */
extern struct libusb_device_handle *openkeyboard(uint8_t *);

#endif
```

Vga_ball.c:

```
/* * Device driver for the VGA video generator
 *
 * A Platform device implemented using the misc subsystem
 *
 * Stephen A. Edwards
 * Columbia University
 *
```

```
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 *               drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod vga_ball.ko
 *
 * Check code style with
 * checkpatch.pl --file --no-tree vga_ball.c
 */

#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "vga_ball.h"

#define DRIVER_NAME "vga_ball"

/* Device registers */
#define BG_RED(x) (x)
#define BG_GREEN(x) ((x)+1)
#define BG_BLUE(x) ((x)+2)
#define BG_BALL_XL(x) ((x)+3)
#define BG_BALL_XH(x) ((x)+4)
#define BG_BALL_YL(x) ((x)+5)
#define BG_BALL_YH(x) ((x)+6)

/*
```

```
 * Information about our device
 */
struct vga_ball_dev {
    struct resource res; /* Resource: our registers */
    void __iomem *virtbase; /* Where registers can be accessed in memory
*/
        vga_ball_color_t background;
    vga_ball_coordinate_t coordinate;
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */
static void write_background(vga_ball_color_t *background)
{
    iowrite8(background->red, BG_RED(dev.virtbase) );
    iowrite8(background->green, BG_GREEN(dev.virtbase) );
    iowrite8(background->blue, BG_BLUE(dev.virtbase) );
    dev.background = *background;
}

static void write_ball(vga_ball_coordinate_t *coordinate)
{
    iowrite8(coordinate->xl, BG_BALL_XL(dev.virtbase) );
    iowrite8(coordinate->xh, BG_BALL_XH(dev.virtbase) );
    iowrite8(coordinate->yl, BG_BALL_YL(dev.virtbase) );
    iowrite8(coordinate->yh, BG_BALL_YH(dev.virtbase) );
    dev.coordinate = *coordinate;
}

static void write_hw(int addr, int info)
{
    iowrite32(info, dev.virtbase + addr);
}

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
```

```
 */
static long vga_ball_ioctl(struct file *f, unsigned int cmd, unsigned long
arg)
{
    vga_ball_arg_t vla;

    switch (cmd) {
    case VGA_BALL_WRITE_BACKGROUND:
        if (copy_from_user(&vla, (vga_ball_arg_t *) arg,
                   sizeof(vga_ball_arg_t)))
            return -EACCES;
        write_hw(vla.addr, vla.info);
        break;

    case VGA_BALL_READ_BACKGROUND:
        vla.background = dev.background;
        if (copy_to_user((vga_ball_arg_t *) arg, &vla,
                 sizeof(vga_ball_arg_t)))
            return -EACCES;
        break;

    default:
        return -EINVAL;
    }

    return 0;
}

/* The operations our device knows how to do */
static const struct file_operations vga_ball_fops = {
    .owner      = THIS_MODULE,
    .unlocked_ioctl = vga_ball_ioctl,
};

/* Information about our device for the "misc" framework -- like a char
dev */
static struct miscdevice vga_ball_misc_device = {
    .minor      = MISC_DYNAMIC_MINOR,
    .name       = DRIVER_NAME,
    .fops       = &vga_ball_fops,
```

```
};

/*
 * Initialization code: get resources (registers) and display
 * a welcome message
 */
static int __init vga_ball_probe(struct platform_device *pdev)
{
        vga_ball_color_t beige = { 0xf9, 0xe4, 0xb7 };
    int ret;

    /* Register ourselves as a misc device: creates /dev/vga_ball */
    ret = misc_register(&vga_ball_misc_device);

    /* Get the address of our registers from the device tree */
    ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
    if (ret) {
        ret = -ENOENT;
        goto out_deregister;
    }

    /* Make sure we can use these registers */
    if (request_mem_region(dev.res.start, resource_size(&dev.res),
                           DRIVER_NAME) == NULL) {
        ret = -EBUSY;
        goto out_deregister;
    }

    /* Arrange access to our registers */
    dev.virtbase = of_iomap(pdev->dev.of_node, 0);
    if (dev.virtbase == NULL) {
        ret = -ENOMEM;
        goto out_release_mem_region;
    }

    /* Set an initial color */
        write_background(&beige);

    return 0;
```

```
out_release_mem_region:
    release_mem_region(dev.res.start, resource_size(&dev.res));
out_deregister:
    misc_deregister(&vga_ball_misc_device);
    return ret;
}

/* Clean-up code: release resources */
static int vga_ball_remove(struct platform_device *pdev)
{
    iounmap(dev.virtbase);
    release_mem_region(dev.res.start, resource_size(&dev.res));
    misc_deregister(&vga_ball_misc_device);
    return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id vga_ball_of_match[] = {
    { .compatible = "csee4840,vga_ball-1.0" },
    {},
};
MODULE_DEVICE_TABLE(of, vga_ball_of_match);
#endif

/* Information for registering ourselves as a "platform" driver */
static struct platform_driver vga_ball_driver = {
    .driver = {
        .name   = DRIVER_NAME,
        .owner  = THIS_MODULE,
        .of_match_table = of_match_ptr(vga_ball_of_match),
    },
    .remove = __exit_p(vga_ball_remove),
};

/* Called when the module is loaded: set things up */
static int __init vga_ball_init(void)
{
    pr_info(DRIVER_NAME ": init\n");
    return platform_driver_probe(&vga_ball_driver, vga_ball_probe);
```

```
}

/* Calball when the module is unloaded: release resources */
static void __exit vga_ball_exit(void)
{
    platform_driver_unregister(&vga_ball_driver);
    pr_info(DRIVER_NAME ": exit\n");
}

module_init(vga_ball_init);
module_exit(vga_ball_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("VGA ball driver");
```

Vga_ball.h:

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>

typedef struct {
    unsigned char red, green, blue;
} vga_ball_color_t;

typedef struct {
    unsigned char xl, xh, yl, yh;
} vga_ball_coordinate_t;

typedef struct {
  vga_ball_color_t background;
    vga_ball_coordinate_t coordinate;
    int addr;
    int info;
} vga_ball_arg_t;
```

```
#define VGA_BALL_MAGIC 'q'

/* ioctls and their arguments */
#define VGA_BALL_WRITE_BACKGROUND _IOW(VGA_BALL_MAGIC, 1, vga_ball_arg_t
*)
#define VGA_BALL_READ_BACKGROUND  _IOR(VGA_BALL_MAGIC, 2, vga_ball_arg_t
*)

#endif
```

Wav to Mif.ipynb:

```
import wave

print("Enter File Name: ")
file_name = input()
im = wave.open(file_name, 'rb')
print(im.getnchannels())
print(im.getsampwidth())
n = im.getnframes()
print("Frame #:")
print(n)
out = "DEPTH = " +str(1000) +";\nWIDTH = 16;\nADDRESS_RADIX =
HEX;\nDATA_RADIX = HEX;\nCONTENT\nBEGIN\n\n"
mem = im.readframes(n)
print(len(mem))
for i in range(1000):
  out += (str(hex(i))[2:]).upper() + " : " +
"0x{:02X}".format(mem[2*i])[2:6] + "0x{:02X}".format(mem[2*i+1])[2:6] +
";\n"

out += "\nEND;"
saved_file_name = input()
s_f = open(saved_file_name, "w")
s_f.write(out)
s_f.close()
pass
```