HuarongDao Inspired Block Game

Haobo Liu (hl3645), Nina Hsu (hh2961), Rui Chu (rc3414), Jingwei Zhang (jz3555), Jiusheng Zhang (jz3444)

1 Introduction

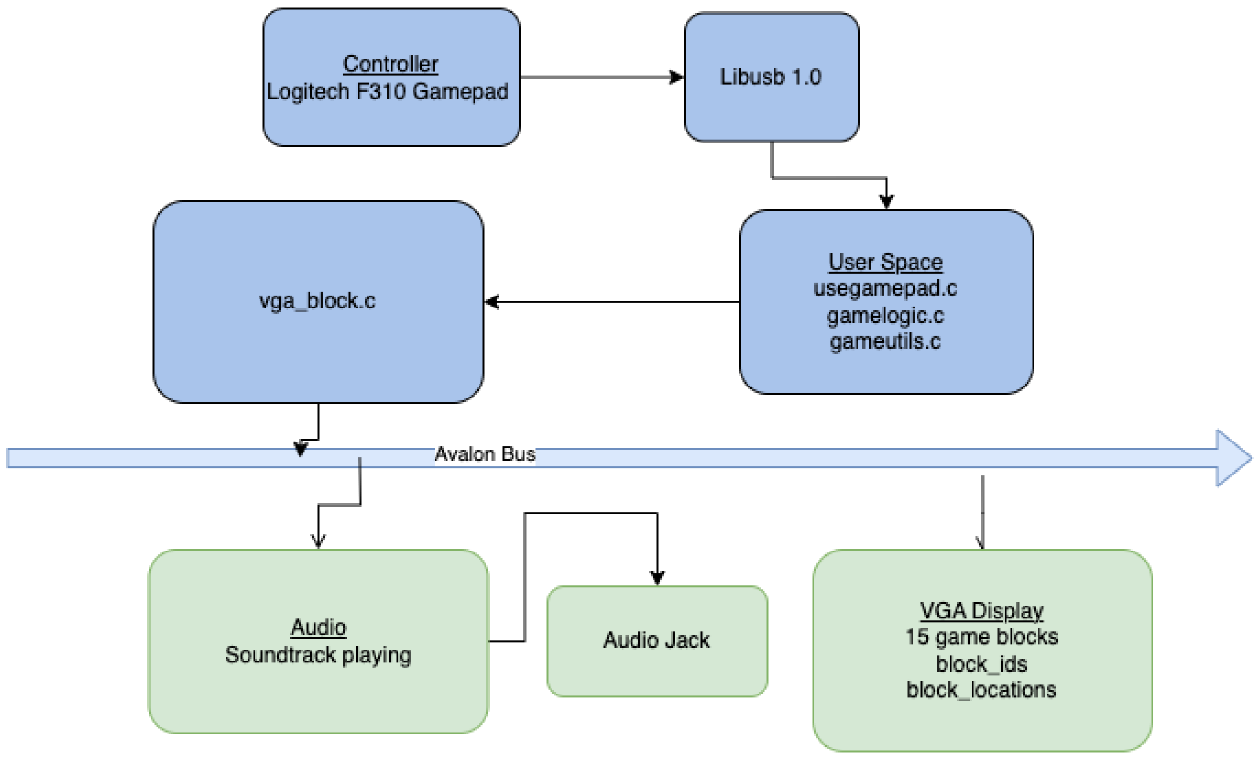
In our final project for the Embedded System course, our group created a block-moving game on a DE1-SoC board using Logitech F310 gamepad as controlling inputs, as well as a VGA display to show our game graphics and speakers to play a background soundtrack along the game playing experience. The simple logic of this game is inspired by both HuarongDao and some other block-moving games. As the player, he/she will be displayed with a 4x4 game space with 15 numbered (1-15) blocks. And the numbers will have scrambled placements on the space. The task for the player is, by moving one block at a time, to re-order the blocks into the correct numerical order.

For displaying the graphic on the VGA screen, we implemented a hardware system verilog design to register each game block as a hardware component, in which each block will be displayed on the screen based on the location information stored in memory. Also, the. The background soundtrack is implemented in System Verilog completely, players will hear the soundtrack the moment they start the game.

Other than those hardware designs, the rest of the system is handled by software in C. First, the Logitech F310 gamepad was connected and registered as a USB HID device using the usblib 1.0 library. After interpreting the packets from Direct-Input mode from the gamepad, we were able to control the game logic using the gamepad. Additionally, the game logic is fully in C code. It will accept input from the gamepad and change the hardware components. Lastly, we also have a C file in charge of any controlling action on the game blocks.

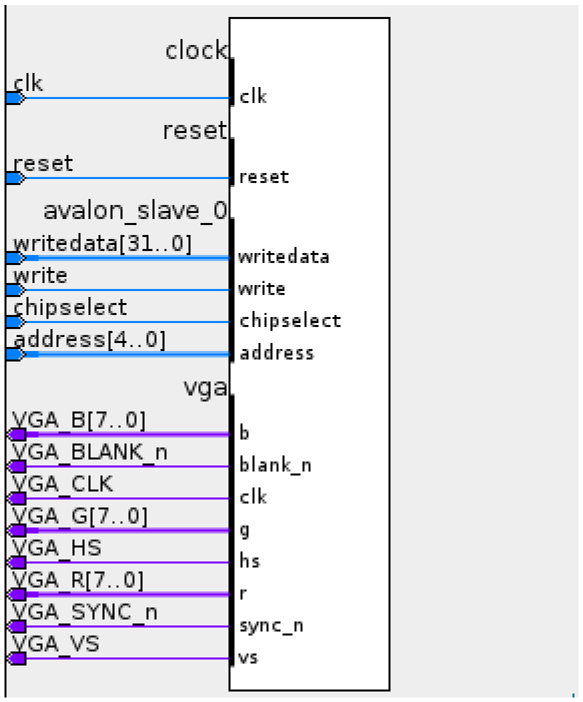
In this report, we will first cover the system Architecture overview to give the whole system design a review. Then, we will cover the hardware design on both the VGA display and soundtrack. Next, we will cover the software design in our design, which will be on both the control drivers and game logic. We will then show a series of screenshots of our project. All the designs and code will be included in the Code Appendix for viewing. Additionally, you can download the zip files in (GitHub link) to try to load this design.

2 System Overview



3 Hardware

Hardware Interface



Writedata:32-bit data received from software

Address:determines the locations the writedata need to be stored to

Signal x, y control the location of corresponding blocks.

Cursor_x, cursor_y control the location of the cursor.

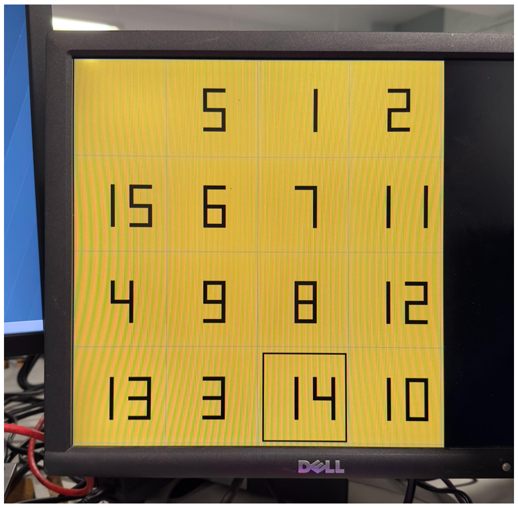
Signal selected indicate whether the block at current cursor is selected.

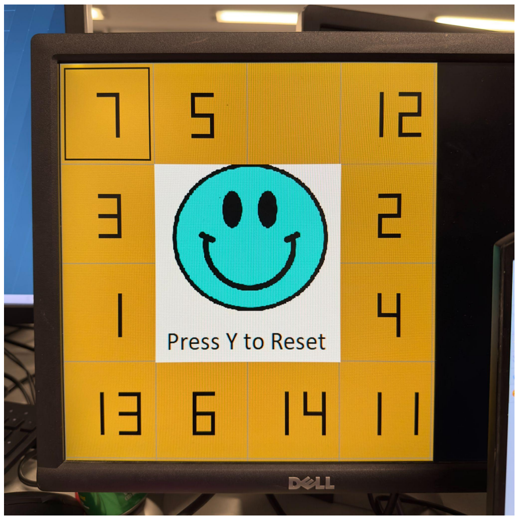
Signal win decides whether to display the smile face.

Address	Writedata (4 bits per cell)							
0	x1	y1	x2	y2	x3	y3	x4	y4
1	x5	y5	x6	y6	x7	y7	x8	y8
2	x9	y9	x10	y10	x11	y11	x12	y12
3	x13	y13	x14	y14	x15	y15	cursor_x	cursor_y

4	unused	selected	win

VGA Display

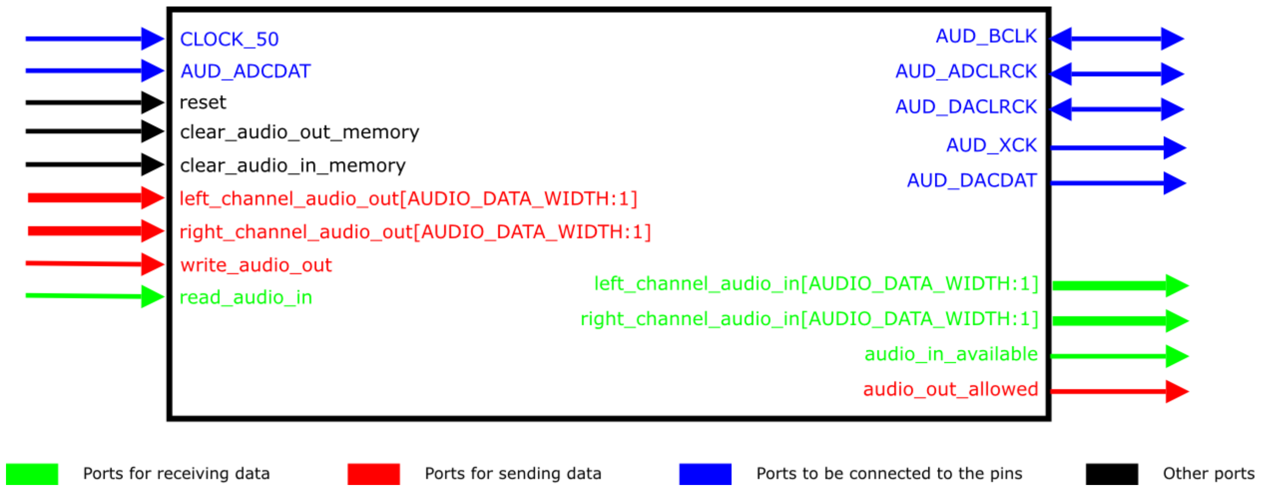


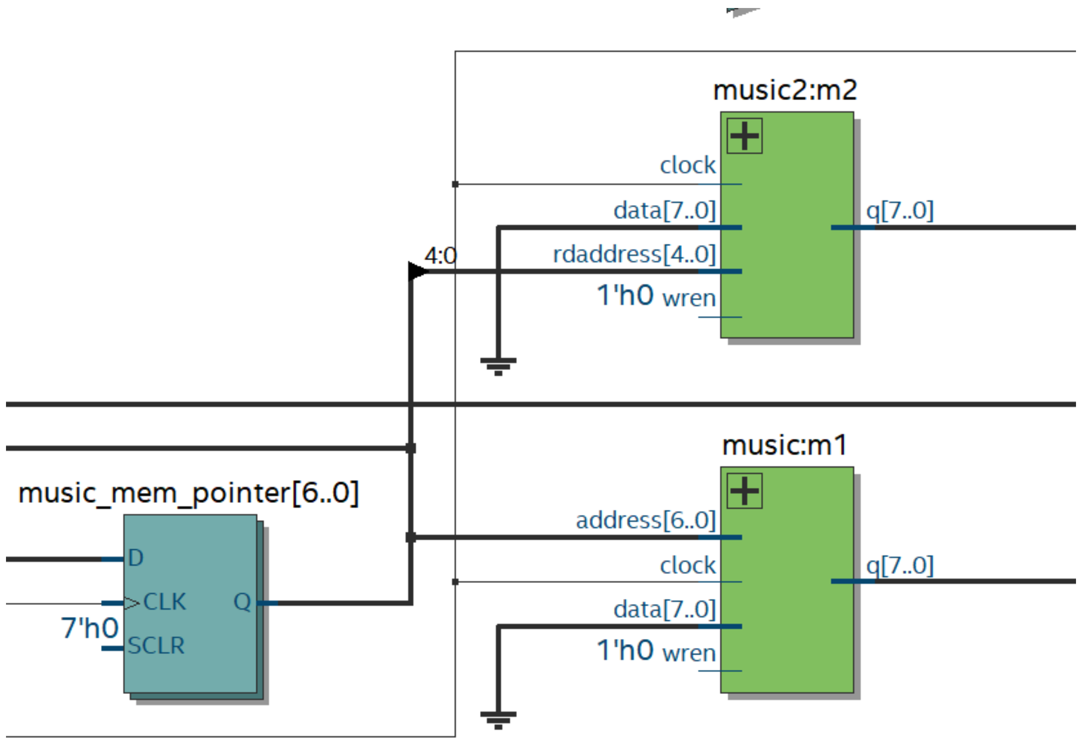


Digital numbers are simple, so we hard coded the pixel configuration of every number in the hardware. The black box represents the cursor. Similar to digital numbers, its pixel configuration was also hard coded in hardware. The smile face is a 240x240 px image, which is loaded into memory during the initialization process. One pixel is read from memory every cycle.

Soundtrack playing

For the audio part, we used an Audio Controller module, which can accept Clock, left/right channel input, and allow_input signals. The tune of the sound is controlled by changing the frequency of input. The frequency number sequences is the data of music which is stored in 2 M10K memory modules as 2 different music selections. When the music is playing, a slower counter advances the memory address and sends the frequency control data to the audio controller.





4 Software

In this section, we will discuss the software design of this project. It will cover the game logic design, hardware interaction, and gamepad driver.

Gamelogic

In the *gamelogic.c* file, this is the file containing all of our game logic for this project. This is our core program that will take in gamepad input and update the VGA display based on the game states. The main states our program keeps track of are:

- Game states
    - Blocks location (in the 4x4 grid)
    - Steps remaining
    - Win
- Cursor location
    - Cursor x/y (in the 4x4 grid)
- Gamepad inputs
    - Actions from player (up/down/left/right, select/un-select)

Game States: For the game state, it will record all the needed information about each block state in the game, which can be used to communicate with the hardware to determine the location of each displayed VGA block. When starting the game, it will initialize a 4x4 board with a random order of 15 blocks and 1 empty block. In the struct for the game state, it will record this information about each block and its relative location in the 4x4 grid.

Code for customized Struct for our gamelogic

```
typedef struct {
   unsigned char x;
   unsigned char y;
} location_t;

typedef struct {
 unsigned char block_id;
 int type; // 0: Empty, 1: Normal Block, 2: Target Block
 location_t location;
} block_t;

typedef struct {
 location_t cursor_loc;
 location_t selected; // set to 4,4 if none gets selected
 unsigned char win;
 block_t blocks[15];
} vga_ball_arg_t;
```

Code for Gamelogic Array

```
block_t arr[4][4] = {
       {block1, block2, block3, block4},
       {block5, block6, block7, block8},
       {block9, block10, block11, block12},
       {block13, block14, block0, block15}};
   for (int i = 0; i < 4; i++)
                   {
                       for (int j = 0; j < 4; j++)
                           {
                               arr[i][j].location.x = j;
                               arr[i][j].location.y = i;
                           }
                   }
```

Cursor Locations: Additionally, we also record the player's cursor location in the game, which reflects which block they are currently on. The cursor will also have a visual effect on the display, which will use information from this code.

Gamepad Inputs: Based on the intercepted packets from the gamepad driver, the game logic file will use the input to change game states, which will reflect on the VGA display if the state changes.

Code for gamepad driver in gamelogic.c file

```
int r = libusb_interrupt_transfer(gamepad, endpoint_address, data, sizeof(data),
&actual_length, 0);

       if (r == 0 && actual_length == sizeof(data))
       {// code for intercepting gamepad input and conduct game logic}
```

With the game logic file recording the game states and communicating them with the hardware, the player can do the following actions in our project game:

1. Connect the gamepad to the FPGA board and start the game

2. Start the game with random order of the blocks
3. Move the cursor to select the block to interact with
4. Press A to select the current block
5. Move blocks (with empty space nearby) with gamepad input
6. Press A again to deselect from the current block
7. Repeat steps 3-6 till
8. Win: blocks are sorted in order
9. Reset: Press Y to reset the game

Gamepad Driver

The main peripheral of this project is a Logitech F310 Gamepad, which is used to provide a way for the player to interact with the game through controller pressing. For connecting the gamepad with our program, we use libusb 1.0 to identify the gamepad as a USB HID device, and we will use the intercepted packets as user input actions.

While implementing the driver for the gamepad, there were some roadblocks we had to overcome to make sure the controller was usable:

1. The controller (Logitech F310) has two different modes when plugging into the board, X-Input, and Direct-Input. According to the analysis using WireShark, those two modes can send different styles of packets to the board. After researching what kind of input will be best suited for FPGA board connection, we found out, by using Direct-Input mode, the board can identify the gamepad as a USB device, which is less raw to handle.
2. Although as a USB device, it is not the keyboard type we used to use in our lab implementation. Therefore, we need to find out the correct way to connect the gamepad and receive its packets. After looking into the documentation of libusb1.0, we found out there is a *USB_HID_GAMEPAD_PROTOCOL* we can use to do the above works.
3. After that, the input from the gamepad is in its raw format of data. We need to find useful data from the raw packets that represent certain actions on the gamepad. Also, we need to understand how the gamepad sends packets. After connecting the gamepad to a test driver, we were successfully able to find the correct data to use in the game logic file to represent the actions on the gamepad.

Code snippet from gamepad.c

```c
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.idVendor == LOGITECH_VENDOR_ID && desc.idProduct ==
LOGITECH_F310_PRODUCT_ID) {
            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 == USB_HID_INTERFACE_CLASS &&
inter->bInterfaceProtocol == USB_HID_GAMEPAD_PROTOCOL) {
                        int r;
                        if ((r = libusb_open(dev, &gamepad)) != 0) {
                            fprintf(stderr, "Error: libusb_open failed: %d\n", r);
                            exit(1);
                        }
                        if (libusb_kernel_driver_active(gamepad, i))
                            libusb_detach_kernel_driver(gamepad, i);
                        libusb_set_auto_detach_kernel_driver(gamepad, 1);
                        if ((r = libusb_claim_interface(gamepad, i)) != 0) {
                            fprintf(stderr, "Error: libusb_claim_interface failed:
%d\n", r);

                            exit(1);
                        }
                        *endpoint_address = inter->endpoint[0].bEndpointAddress;
                        goto found;
                    }
                }
            }
        }
    }
```

After setup, we only used partial input from the gamepad, which is enough for our game logic implementation:

- Arrow buttons: Up/Down/Left/Right for moving the cursor/block
- Action buttons: A/Y for different game interaction

Note: in the game logic file, the game loop will ignore any other packet input that is not one of the above actions.

vga_ball.c

This file helps to aid the interface between hardware components with our software design. The main purpose of this file gives the necessary communication methods for software in C to access hardware information on our game blocks.

Code for VLA struct

```
typedef struct {
 location_t cursor_loc;
 location_t selected; // set to 4,4 if none gets selected
 unsigned char win;
 block_t blocks[15];
} vga_ball_arg_t;
```

```
struct vga_ball_dev {
   struct resource res; /* Resource: our registers */
   void __iomem *virtbase; /* Where registers can be accessed in memory */
   location_t locations[16];
} dev;

static void writelocation(location_t loc1, location_t loc2,
               location_t loc3, location_t loc4,
               unsigned char offset)
{
   unsigned long data;
   data = ((loc1.x << 28) & 0xF0000000) | ((loc1.y << 24) & 0x0F000000)
       |((loc2.x << 20) & 0x00F00000) | ((loc2.y << 16) & 0x000F0000)
       | ((loc3.x << 12) & 0x0000F000) | ((loc3.y  << 8) & 0x00000F00)
```

```
      | ((loc4.x << 4) & 0x000000F0) | ((loc4.y & 0x0000000F));
  iowrite32(data, LOC(dev.virtbase, offset));
}
```

5 Discussion and Learning

In our project design, we implemented a essentially solve-15 game in a FPGA board situation. The design was simple, but we were able to combine things we learned from classes and labs into this project.

The roadblock encountered in this project was also able to help us understand the design in embedded systems further.

- First, connecting a USB gamepad requires finding the compatible controller to the FPGA board, as well as finding the correct lib to use so the board can identify the USB plugin as a HID device. Also, since most online resources on libusb 1.0 are about how to use it for keyboard input, we need to deal with the raw packets from the gamepad ourselves. After using WireShark to analyze the output, we were able to find the data we needed to use for our gamelogic.
- Second, since our project requires the display of pictures and numerical data, we need to find a way to load our resources into the hardware for compiling. The way we came up with were two parts: for the block number, we wrote the correct display of each number (1-15) in our systemverilog file; then, for the picture we need to display, we use tool to transfer the 240x240 png file into .mif file, which can be latter used to create new memory black in Quartus.
- Third, for the gamelogic implementation, we need to understand how can we interface with the block components in hardware. The method we came up with was to pass the gamestate to the block driver file, and in the driver, it will break down the gamestate into addresses to retrieve/update. Thus, our graphic display was able to show on VGA based on the current gamestate. By restricting the player actions inside the gamelogic, we were able to avoid any hardware faults.

There are also many future improvements we can do to make this a better game to play with:

- The game interface is still in a rough shape since we did not use Sprite to display layers of graphics. If time allowed, we could make a more sophisticated game UI so the playing will be smooth
- The soundtrack playing is currently isolated from the game playing, our hardware was capable to play the sound, all we need to add was another sound driver file. By doing so, we can add more game sound to the project.
- For the block display, we displayed everything on a single layer. Since our gamelogic is simple enough, there seems to have no bug/lag due to memory accessing issue. However, for future more complicated UI design, we need to make sure our memory access has no conflict.

6 Team Contribution

- Haobo Liu (hl3645): Gamelogic design and implementation, Project documentation writing (slides, reports)
- Nina Hsu (hh2961): GameLogic design and implementation
- Jingwei Zhang (jz3555): Soundtrack playing design, graphic display implementation
- Jiusheng Zhang (jz3444):Soundtrack playing design, graphic display implementation
- Rui Chu (rc3414): Gamelogic design and implementation, graphic display implementation

7 Code Appendix

SV file:

Music.v

```
// megafunction wizard: %RAM: 1-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: music.v
// Megafunction Name(s):
//          altsyncram
//
// Simulation Library Files(s):
//          altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 22.1std.1 Build 917 02/14/2023 SC Lite Edition
// ************************************************************



//Copyright (C) 2023  Intel Corporation. All rights reserved.
//Your use of Intel Corporation's design tools, logic functions
//and other software and tools, and any partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License Agreement,
//the Intel FPGA IP License Agreement, or other applicable license
//agreement, including, without limitation, that your use is for
//the sole purpose of programming logic devices manufactured by
//Intel and sold by Intel or its authorized distributors.  Please
//refer to the applicable agreement for further details, at
//https://fpgasoftware.intel.com/eula.
```

```
// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module music (
   address,
   clock,
   data,
   wren,
   q);

   input   [6:0]  address;
   input     clock;
   input   [7:0]  data;
   input     wren;
   output  [7:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
   tri1     clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

   wire [7:0] sub_wire0;
   wire [7:0] q = sub_wire0[7:0];

   altsyncram  altsyncram_component (
               .address_a (address),
               .clock0 (clock),
               .data_a (data),
               .wren_a (wren),
               .q_a (sub_wire0),
               .aclr0 (1'b0),
               .aclr1 (1'b0),
               .address_b (1'b1),
               .addressstall_a (1'b0),
               .addressstall_b (1'b0),
               .byteena_a (1'b1),
               .byteena_b (1'b1),
               .clock1 (1'b1),
               .clocken0 (1'b1),
               .clocken1 (1'b1),
```

```verilog
               .clocken2 (1'b1),
               .clocken3 (1'b1),
               .data_b (1'b1),
               .eccstatus (),
               .q_b (),
               .rden_a (1'b1),
               .rden_b (1'b1),
               .wren_b (1'b0));
   defparam
       altsyncram_component.clock_enable_input_a = "BYPASS",
       altsyncram_component.clock_enable_output_a = "BYPASS",
       altsyncram_component.init_file = "../music.mif",
       altsyncram_component.intended_device_family = "Cyclone V",
       altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
       altsyncram_component.lpm_type = "altsyncram",
       altsyncram_component.numwords_a = 128,
       altsyncram_component.operation_mode = "SINGLE_PORT",
       altsyncram_component.outdata_aclr_a = "NONE",
       altsyncram_component.outdata_reg_a = "CLOCK0",
       altsyncram_component.power_up_uninitialized = "FALSE",
       altsyncram_component.ram_block_type = "M10K",
       altsyncram_component.read_during_write_mode_port_a = "DONT_CARE",
       altsyncram_component.widthad_a = 7,
       altsyncram_component.width_a = 8,
       altsyncram_component.width_byteena_a = 1;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrData NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: DataBusSeparated NUMERIC "1"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "../music.mif"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "128"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "2"
// Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "2"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegData NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "1"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "1"
// Retrieval info: PRIVATE: WRCONTROL_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "7"
// Retrieval info: PRIVATE: WidthData NUMERIC "8"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING "../music.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "128"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "SINGLE_PORT"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "CLOCK0"
// Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
// Retrieval info: CONSTANT: RAM_BLOCK_TYPE STRING "M10K"
// Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_PORT_A STRING "DONT_CARE"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "7"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "8"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 7 0 INPUT NODEFVAL "address[6..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: data 0 0 8 0 INPUT NODEFVAL "data[7..0]"
// Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL "q[7..0]"
// Retrieval info: USED_PORT: wren 0 0 0 0 INPUT NODEFVAL "wren"
// Retrieval info: CONNECT: @address_a 0 0 7 0 address 0 0 7 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: @data_a 0 0 8 0 data 0 0 8 0
// Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
// Retrieval info: CONNECT: q 0 0 8 0 @q_a 0 0 8 0
// Retrieval info: GEN_FILE: TYPE_NORMAL music.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL music.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL music.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL music.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL music_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL music_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf
```

Face.v

```
// megafunction wizard: %ROM: 1-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: face.v
// Megafunction Name(s):
//          altsyncram
//
// Simulation Library Files(s):
//          altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 21.1.0 Build 842 10/21/2021 SJ Lite Edition
// ************************************************************



//Copyright (C) 2021  Intel Corporation. All rights reserved.
//Your use of Intel Corporation's design tools, logic functions
//and other software and tools, and any partner logic
```

```
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License Agreement,
//the Intel FPGA IP License Agreement, or other applicable license
//agreement, including, without limitation, that your use is for
//the sole purpose of programming logic devices manufactured by
//Intel and sold by Intel or its authorized distributors.  Please
//refer to the applicable agreement for further details, at
//https://fpgasoftware.intel.com/eula.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module face (
   address,
   clock,
   q);

   input   [15:0]  address;
   input     clock;
   output  [2:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
   tri1      clock;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

   wire [2:0] sub_wire0;
   wire [2:0] q = sub_wire0[2:0];

   altsyncram  altsyncram_component (
               .address_a (address),
               .clock0 (clock),
               .q_a (sub_wire0),
               .aclr0 (1'b0),
               .aclr1 (1'b0),
               .address_b (1'b1),
               .addressstall_a (1'b0),
               .addressstall_b (1'b0),
               .byteena_a (1'b1),
               .byteena_b (1'b1),
               .clock1 (1'b1),
               .clocken0 (1'b1),
               .clocken1 (1'b1),
               .clocken2 (1'b1),
               .clocken3 (1'b1),
               .data_a ({3{1'b1}}),
               .data_b (1'b1),
               .eccstatus (),
               .q_b (),
               .rden_a (1'b1),
               .rden_b (1'b1),
               .wren_a (1'b0),
               .wren_b (1'b0));
   defparam
       altsyncram_component.address_aclr_a = "NONE",
       altsyncram_component.clock_enable_input_a = "BYPASS",
       altsyncram_component.clock_enable_output_a = "BYPASS",
       altsyncram_component.init_file = "face.mif",
       altsyncram_component.intended_device_family = "Cyclone V",
       altsyncram_component.lpm_hint = "ENABLE_RUNTIME_MOD=NO",
       altsyncram_component.lpm_type = "altsyncram",
       altsyncram_component.numwords_a = 57600,
       altsyncram_component.operation_mode = "ROM",
       altsyncram_component.outdata_aclr_a = "NONE",
       altsyncram_component.outdata_reg_a = "CLOCK0",
       altsyncram_component.ram_block_type = "M10K",
       altsyncram_component.widthad_a = 16,
       altsyncram_component.width_a = 3,
       altsyncram_component.width_byteena_a = 1;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: AclrAddr NUMERIC "0"
// Retrieval info: PRIVATE: AclrByte NUMERIC "0"
// Retrieval info: PRIVATE: AclrOutput NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: Clken NUMERIC "0"
// Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
// Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_A"
// Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
// Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
// Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
// Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
// Retrieval info: PRIVATE: MIFfilename STRING "face.mif"
// Retrieval info: PRIVATE: NUMWORDS_A NUMERIC "57600"
// Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "2"
// Retrieval info: PRIVATE: RegAddr NUMERIC "1"
// Retrieval info: PRIVATE: RegOutput NUMERIC "1"
// Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
// Retrieval info: PRIVATE: SingleClock NUMERIC "1"
// Retrieval info: PRIVATE: UseDQRAM NUMERIC "0"
// Retrieval info: PRIVATE: WidthAddr NUMERIC "16"
// Retrieval info: PRIVATE: WidthData NUMERIC "3"
// Retrieval info: PRIVATE: rden NUMERIC "0"
// Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
// Retrieval info: CONSTANT: ADDRESS_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_A STRING "BYPASS"
// Retrieval info: CONSTANT: INIT_FILE STRING "face.mif"
// Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
// Retrieval info: CONSTANT: LPM_HINT STRING "ENABLE_RUNTIME_MOD=NO"
// Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
// Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "57600"
// Retrieval info: CONSTANT: OPERATION_MODE STRING "ROM"
// Retrieval info: CONSTANT: OUTDATA_ACLR_A STRING "NONE"
// Retrieval info: CONSTANT: OUTDATA_REG_A STRING "CLOCK0"
// Retrieval info: CONSTANT: RAM_BLOCK_TYPE STRING "M10K"
// Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "16"
// Retrieval info: CONSTANT: WIDTH_A NUMERIC "3"
// Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
// Retrieval info: USED_PORT: address 0 0 16 0 INPUT NODEFVAL "address[15..0]"
// Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
// Retrieval info: USED_PORT: q 0 0 3 0 OUTPUT NODEFVAL "q[2..0]"
// Retrieval info: CONNECT: @address_a 0 0 16 0 address 0 0 16 0
// Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
// Retrieval info: CONNECT: q 0 0 3 0 @q_a 0 0 3 0
// Retrieval info: GEN_FILE: TYPE_NORMAL face.v TRUE
// Retrieval info: GEN_FILE: TYPE_NORMAL face.inc FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL face.cmp FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL face.bsf FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL face_inst.v FALSE
// Retrieval info: GEN_FILE: TYPE_NORMAL face_bb.v TRUE
// Retrieval info: LIB_FILE: altera_mf
```

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 [4:0]  address,

   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 [7:0]      background_r, background_g, background_b;

 logic [9:0]     x1, y1;
 logic [9:0]     x2, y2;
 logic [9:0]     x3, y3;
 logic [9:0]     x4, y4;
 logic [9:0]     x5, y5;
 logic [9:0]     x6, y6;
 logic [9:0]     x7, y7;
 logic [9:0]     x8, y8;
 logic [9:0]     x9, y9;
 logic [9:0]     x10, y10;
 logic [9:0]     x11, y11;
 logic [9:0]     x12, y12;
 logic [9:0]     x13, y13;
 logic [9:0]     x14, y14;
 logic [9:0]     x15, y15;
 logic [9:0]     cursor_x, cursor_y;
 logic [9:0]     width;
 logic [9:0]     selected_x, selected_y;
 logic           win;
 vga_counters counters(.clk50(clk), .*);
 logic [23:0]     face_out;
 face f1(hcount[10:1] - 119 + (vcount - 119) * 240, clk, face_out);

 always_ff @(posedge clk)
   if (reset) begin
 width = 10'h5;
 background_r <= 8'h0;
 background_g <= 8'h0;
 background_b <= 8'h0;
 x1 <= 10'h0;
 y1 <= 10'h0;
 x2 <= 10'h78; // 120
 y2 <= 10'h0;
 x3 <= 10'hf0; // 240
 y3 <= 10'h0;
 x4 <= 10'h0;
 y4 <= 10'h78; // 120
 x5 <= 10'h78;
 y5 <= 10'h78;
 x6 <= 10'hf0;
 y6 <= 10'h78;
 x7 <= 10'h168; // 360
 y7 <= 10'h78;
```

```
x8 <= 10'h0;
y8 <= 10'hf0;
x9 <= 10'h78;
y9 <= 10'hf0;
x10 <= 10'hf0;
y10 <= 10'hf0;
x11 <= 10'h168;
y11 <= 10'hf0;
x12 <= 10'h00;
y12 <= 10'h168;
x13 <= 10'h78;
y13 <= 10'h168;
x14 <= 10'hf0;
y14 <= 10'h168;
x15 <= 10'h168;
y15 <= 10'h168;
cursor_x <= 10'h0;
cursor_y <= 10'h0;
selected_x = 10'h4;
selected_y = 10'h4;
win <= 1'b0;
  end else if (chipselect && write)
    case (address)
5'h00 : begin
  x1 <= {1'b0, writedata[31:28] * 120};
  y1 <= {1'b0, writedata[27:24] * 120};
  x2 <= {1'b0, writedata[23:20] * 120};
  y2 <= {1'b0, writedata[19:16] * 120};
   x3 <= {1'b0, writedata[15:12] * 120};
  y3 <= {1'b0, writedata[11:8] * 120};
  x4 <= {1'b0, writedata[7:4] * 120};
  y4 <= {1'b0, writedata[3:0] * 120};
end
5'h01 : begin
  x5 <= {1'b0, writedata[31:28] * 120};
  y5 <= {1'b0, writedata[27:24] * 120};
  x6 <= {1'b0, writedata[23:20] * 120};
  y6 <= {1'b0, writedata[19:16] * 120};

  x7 <= {1'b0, writedata[15:12] * 120};
  y7 <= {1'b0, writedata[11:8] * 120};
  x8 <= {1'b0, writedata[7:4] * 120};
```

```
   y8 <= {1'b0, writedata[3:0] * 120};
 end
 5'h02 : begin
   x9 <= {1'b0, writedata[31:28] * 120};
   y9 <= {1'b0, writedata[27:24] * 120};
   x10 <= {1'b0, writedata[23:20] * 120};
   y10 <= {1'b0, writedata[19:16] * 120};
    x11 <= {1'b0, writedata[15:12] * 120};
   y11 <= {1'b0, writedata[11:8] * 120};
   x12 <= {1'b0, writedata[7:4] * 120};
   y12 <= {1'b0, writedata[3:0] * 120};
 end
 3'h03 : begin
   x13 <= {1'b0, writedata[31:28] * 120};
   y13 <= {1'b0, writedata[27:24] * 120};
   x14 <= {1'b0, writedata[23:20] * 120};
   y14 <= {1'b0, writedata[19:16] * 120};

   x15 <= {1'b0, writedata[15:12] * 120};
   y15 <= {1'b0, writedata[11:8] * 120};
   cursor_x <= {1'b0, writedata[7:4] * 120};
   cursor_y <= {1'b0, writedata[3:0] * 120};
 end
 3'h04 : begin
   selected_x <= {1'b0, writedata[15:12]};
   selected_y <= {1'b0, writedata[11:8]};
   win <= writedata[0];
 end
     endcase

 always_comb begin
     {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
     if (VGA_BLANK_n )
 if (hcount[10:1] >= 0 && hcount[10:1] < 480) begin
   if (hcount[10:1] >= x1 + 71 && hcount[10:1] < x1 + 71 + width && vcount >= y1 + 35
&& vcount < y1 + 83 + width || // 1
       hcount[10:1] >= x2 + 47 && hcount[10:1] < x2 + 71 + width &&
       (vcount >= y2 + 35 && vcount < y2 + 35 + width ||
       vcount >= y2 + 59 && vcount < y2 + 59 + width ||
       vcount >= y2 + 83 && vcount < y2 + 83 + width) ||
       hcount[10:1] >= x2 + 71 && hcount[10:1] < x2 + 71 + width && vcount >= y2 + 35
&& vcount < y2 + 59 + width||
       hcount[10:1] >= x2 + 47 && hcount[10:1] < x2 + 47 + width && vcount >= y2 + 59
&& vcount < y2 + 83 + width|| // 2
       hcount[10:1] >= x3 + 47 && hcount[10:1] < x3 + 71 + width &&
       (vcount >= y3 + 35 && vcount < y3 + 35 + width ||
       vcount >= y3 + 59 && vcount < y3 + 59 + width ||
       vcount >= y3 + 83 && vcount < y3 + 83 + width) ||
       hcount[10:1] >= x3 + 71 && hcount[10:1] < x3 + 71 + width && vcount >= y3 + 35
&& vcount < y3 + 83 + width|| // 3
       hcount[10:1] >= x4 + 47 && hcount[10:1] < x4 + 71 + width && vcount >= y4 + 59
&& vcount < y4 + 59 + width ||
       hcount[10:1] >= x4 + 47 && hcount[10:1] < x4 + 47 + width && vcount >= y4 + 35
&& vcount < y4 + 59 + width||
       hcount[10:1] >= x4 + 71 && hcount[10:1] < x4 + 71 + width && vcount >= y4 + 35
&& vcount < y4 + 83 + width|| // 4
       hcount[10:1] >= x5 + 47 && hcount[10:1] < x5 + 71 + width &&
       (vcount >= y5 + 35 && vcount < y5 + 35 + width ||
       vcount >= y5 + 59 && vcount < y5 + 59 + width ||
       vcount >= y5 + 83 && vcount < y5 + 83 + width) ||
       hcount[10:1] >= x5 + 71 && hcount[10:1] < x5 + 71 + width && vcount >= y5 + 59
&& vcount < y5 + 83 + width ||
       hcount[10:1] >= x5 + 47 && hcount[10:1] < x5 + 47 + width && vcount >= y5 + 35
&& vcount < y5 + 59 + width || // 5
       hcount[10:1] >= x6 + 47 && hcount[10:1] < x6 + 71 + width &&
       (vcount >= y6 + 35 && vcount < y6 + 35 + width ||
       vcount >= y6 + 59 && vcount < y6 + 59 + width ||
       vcount >= y6 + 83 && vcount < y6 + 83 + width) ||
       hcount[10:1] >= x6 + 71 && hcount[10:1] < x6 + 71 + width && vcount >= y6 + 59
&& vcount < y6 + 83 + width ||
       hcount[10:1] >= x6 + 47 && hcount[10:1] < x6 + 47 + width && vcount >= y6 + 35
&& vcount < y6 + 83 + width || // 6
       hcount[10:1] >= x7 + 47 && hcount[10:1] < x7 + 71 + width && vcount >= y7 + 35
&& vcount < y7 + 35 + width ||
       hcount[10:1] >= x7 + 71 && hcount[10:1] < x7 + 71 + width && vcount >= y7 + 35
&& vcount < y7 + 83 + width || // 7
       hcount[10:1] >= x8 + 47 && hcount[10:1] < x8 + 71 &&
       (vcount >= y8 + 35 && vcount < y8 + 35 + width ||
       vcount >= y8 + 59 && vcount < y8 + 59 + width ||
       vcount >= y8 + 83 && vcount < y8 + 83 + width) ||
       hcount[10:1] >= x8 + 47 && hcount[10:1] < x8 + 47 + width && vcount >= y8 + 35
&& vcount < y8 + 83 + width ||
       hcount[10:1] >= x8 + 71 && hcount[10:1] < x8 + 71 + width && vcount >= y8 + 35
&& vcount < y8 + 83 + width || // 8
```

```
       hcount[10:1] >= x9 + 47 && hcount[10:1] < x9 + 71 + width &&
       (vcount >= y9 + 35 && vcount < y9 + 35 + width ||
       vcount >= y9 + 59 && vcount < y9 + 59 + width ||
       vcount >= y9 + 83 && vcount < y9 + 83 + width) ||
       hcount[10:1] >= x9 + 47 && hcount[10:1] < x9 + 47 + width && vcount >= y9 + 35
&& vcount < y9 + 59 + width ||
       hcount[10:1] >= x9 + 71 && hcount[10:1] < x9 + 71 + width && vcount >= y9 + 35
&& vcount < y9 + 83 + width || // 9
       hcount[10:1] >= x10 + 47 && hcount[10:1] < x10 + 47 + width && vcount >= y10 +
35 && vcount < y10 + 83 + width ||
       hcount[10:1] >= x10 + 71 && hcount[10:1] < x10 + 95 + width &&
       (vcount >= y10 + 35 && vcount < y10 + 35 + width ||
       vcount >= y10 + 83 && vcount < y10 + 83 + width) ||
       hcount[10:1] >= x10 + 71 && hcount[10:1] < x10 + 71 + width && vcount >= y10 +
35 && vcount < y10 + 83 + width ||
       hcount[10:1] >= x10 + 95 && hcount[10:1] < x10 + 95 + width && vcount >= y10 +
35 && vcount < y10 + 83 + width || // 10
       hcount[10:1] >= x11 + 47 && hcount[10:1] < x11 + 47 + width && vcount >= y11 +
35 && vcount < y11 + 83 + width ||
       hcount[10:1] >= x11 + 95 && hcount[10:1] < x11 + 95 + width && vcount >= y11 +
35 && vcount < y11 + 83 + width || // 11
       hcount[10:1] >= x12 + 47 && hcount[10:1] < x12 + 47 + width && vcount >= y12 +
35 && vcount < y12 + 83 + width ||
       hcount[10:1] >= x12 + 71 && hcount[10:1] < x12 + 95 + width &&
       (vcount >= y12 + 35 && vcount < y12 + 35 + width ||
       vcount >= y12 + 59 && vcount < y12 + 59 + width ||
       vcount >= y12 + 83 && vcount < y12 + 83 + width) ||
       hcount[10:1] >= x12 + 95 && hcount[10:1] < x12 + 95 + width && vcount >= y12 +
35 && vcount < y12 + 59 + width ||
       hcount[10:1] >= x12 + 71 && hcount[10:1] < x12 + 71 + width && vcount >= y12 +
59 && vcount < y12 + 83 + width || // 12
       hcount[10:1] >= x13 + 47 && hcount[10:1] < x13 + 47 + width && vcount >= y13 +
35 && vcount < y13 + 83 + width ||
       hcount[10:1] >= x13 + 71 && hcount[10:1] < x13 + 95 &&
       (vcount >= y13 + 35 && vcount < y13 + 35 + width ||
       vcount >= y13 + 59 && vcount < y13 + 59 + width ||
       vcount >= y13 + 83 && vcount < y13 + 83 + width) ||
       hcount[10:1] >= x13 + 95 && hcount[10:1] < x13 + 95 + width && vcount >= y13 +
35 && vcount < y13 + 83 + width || // 13
       hcount[10:1] >= x14 + 47 && hcount[10:1] < x14 + 47 + width && vcount >= y14 +
35 && vcount < y14 + 83 + width ||
       hcount[10:1] >= x14 + 71 && hcount[10:1] < x14 + 95 + width && vcount >= y14 +
59 && vcount < y14 + 59 + width ||
       hcount[10:1] >= x14 + 71 && hcount[10:1] < x14 + 71 + width && vcount >= y14 +
35 && vcount < y14 + 59 + width ||
       hcount[10:1] >= x14 + 95 && hcount[10:1] < x14 + 95 + width && vcount >= y14 +
35 && vcount < y14 + 83 + width || // 14
       hcount[10:1] >= x15 + 47 && hcount[10:1] < x15 + 47 + width && vcount >= y15 +
35 && vcount < y15 + 83 + width ||
       hcount[10:1] >= x15 + 71 && hcount[10:1] < x15 + 95 + width &&
       (vcount >= y15 + 35 && vcount < y15 + 35 + width ||
       vcount >= y15 + 59 && vcount < y15 + 59 + width ||
       vcount >= y15 + 83 && vcount < y15 + 83 + width) ||
       hcount[10:1] >= x15 + 95 && hcount[10:1] < x15 + 95 + width && vcount >= y15 +
59 && vcount < y15 + 83 + width ||
       hcount[10:1] >= x15 + 71 && hcount[10:1] < x15 + 71 + width && vcount >= y15 +
35 && vcount < y15 + 59 + width // 15
   )
     {VGA_R, VGA_G, VGA_B} = {background_r, background_g, background_b};
   else if (hcount[10:1] % 119 == 0 || vcount % 119 == 0)
     {VGA_R, VGA_G, VGA_B} =
             {8'h80, 8'h80, 8'h80};
   else
     {VGA_R, VGA_G, VGA_B} = {8'hbf, 8'h8f, 8'h00};
 end
 else
   {VGA_R, VGA_G, VGA_B} =
           {background_r, background_g, background_b};
 /* cursor */
 if (hcount[10:1] >= cursor_x + 5 && hcount[10:1] < cursor_x + 115 && vcount >=
cursor_y + 5 && vcount < cursor_y + 7)
   {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
 if (hcount[10:1] >= cursor_x + 5 && hcount[10:1] < cursor_x + 7 && vcount >= cursor_y
+ 5 && vcount < cursor_y + 115)
   {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
 if (hcount[10:1] >= cursor_x + 113 && hcount[10:1] < cursor_x + 115 && vcount >=
cursor_y + 5 && vcount < cursor_y + 115)
   {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
 if (hcount[10:1] >= cursor_x + 5 && hcount[10:1] < cursor_x + 115 && vcount >=
cursor_y + 113 && vcount < cursor_y + 115)
   {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
 if (selected_x < 4 && selected_y < 4) begin
```

```
   if (hcount[10:1] >= cursor_x + 7 && hcount[10:1] < cursor_x + 113 && vcount >= 
cursor_y + 7 && vcount < cursor_y + 9)
     {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'h00};
   if (hcount[10:1] >= cursor_x + 7 && hcount[10:1] < cursor_x + 9 && vcount >= 
cursor_y + 7 && vcount < cursor_y + 113)
     {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'h00};
   if (hcount[10:1] >= cursor_x + 111 && hcount[10:1] < cursor_x + 113 && vcount >= 
cursor_y + 7 && vcount < cursor_y + 113)
     {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'h00};
   if (hcount[10:1] >= cursor_x + 7 && hcount[10:1] < cursor_x + 113 && vcount >= 
cursor_y + 111 && vcount < cursor_y + 113)
     {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'h00};
 end
 if (win == 1) begin
   if (hcount[10:1] >= 119 && hcount[10:1] < 359 && vcount >= 119 && vcount < 359) 
begin
     {VGA_R, VGA_G, VGA_B} = face_out;
     /*
     if (face_out[2] == 1)
       VGA_R <= 8'hff;
     else
       VGA_R <= 8'h00;
     if (face_out[1] == 1)
       VGA_G <= 8'hff;
     else
       VGA_G <= 8'h00;
     if (face_out[0] == 1)
       VGA_B <= 8'hff;
     else
       VGA_B <= 8'h00;
       */
   end
 end
 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
```

<u>C file:</u>

Usegamepad.h

```
#ifndef USBGAMEPAD_H
#define USBGAMEPAD_H

#include <stdint.h>
#include <libusb-1.0/libusb.h>
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>

#define USB_HID_INTERFACE_CLASS 0x03
#define USB_HID_GAMEPAD_PROTOCOL 0x00
#define LOGITECH_VENDOR_ID 0x046d
```

```
#define LOGITECH_F310_PRODUCT_ID 0xc216

struct libusb_device_handle *open_gamepad(uint8_t *endpoint_address) {
   libusb_device **devs;
   struct libusb_device_handle *gamepad = NULL;
   struct libusb_device_descriptor desc;
   ssize_t num_devs, d;
   uint8_t i, k;

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

   if ((num_devs = libusb_get_device_list(NULL, &devs)) < 0) {
       fprintf(stderr, "Error: libusb_get_device_list failed\n");
       exit(1);
   }

   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.idVendor == LOGITECH_VENDOR_ID && desc.idProduct ==
LOGITECH_F310_PRODUCT_ID) {
           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 == USB_HID_INTERFACE_CLASS &&
inter->bInterfaceProtocol == USB_HID_GAMEPAD_PROTOCOL) {
                       int r;
                       if ((r = libusb_open(dev, &gamepad)) != 0) {
                           fprintf(stderr, "Error: libusb_open failed: %d\n", r);
                           exit(1);
                       }
                       if (libusb_kernel_driver_active(gamepad, i))
```

```
                            libusb_detach_kernel_driver(gamepad, i);
                        libusb_set_auto_detach_kernel_driver(gamepad, 1);
                        if ((r = libusb_claim_interface(gamepad, 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 gamepad;
}

#endif // USBGAMEPAD_H
```

Vga_ball.c

```
#ifndef USBGAMEPAD_H
#define USBGAMEPAD_H

#include <stdint.h>
#include <libusb-1.0/libusb.h>
#include <stdio.h>
#include <stdlib.h>
#include <arpa/inet.h>

#define USB_HID_INTERFACE_CLASS 0x03
#define USB_HID_GAMEPAD_PROTOCOL 0x00
#define LOGITECH_VENDOR_ID 0x046d
#define LOGITECH_F310_PRODUCT_ID 0xc216

struct libusb_device_handle *open_gamepad(uint8_t *endpoint_address) {
    libusb_device **devs;
   struct libusb_device_handle *gamepad = NULL;
   struct libusb_device_descriptor desc;
   ssize_t num_devs, d;
   uint8_t i, k;

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

   if ((num_devs = libusb_get_device_list(NULL, &devs)) < 0) {
       fprintf(stderr, "Error: libusb_get_device_list failed\n");
       exit(1);
   }

   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.idVendor == LOGITECH_VENDOR_ID && desc.idProduct ==
LOGITECH_F310_PRODUCT_ID) {
           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 == USB_HID_INTERFACE_CLASS &&
inter->bInterfaceProtocol == USB_HID_GAMEPAD_PROTOCOL) {
                       int r;
                       if ((r = libusb_open(dev, &gamepad)) != 0) {
                           fprintf(stderr, "Error: libusb_open failed: %d\n", r);
                           exit(1);
                       }
                       if (libusb_kernel_driver_active(gamepad, i))
                           libusb_detach_kernel_driver(gamepad, i);
                       libusb_set_auto_detach_kernel_driver(gamepad, 1);
                       if ((r = libusb_claim_interface(gamepad, 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 gamepad;
}

#endif // USBGAMEPAD_H
```

Gamelogic.h

```
#ifndef _HUARONGDAO_H
#define _HUARONGDAO_H

#include <linux/ioctl.h>

typedef struct {
   unsigned char x;
   unsigned char y;
} location_t;

typedef struct {
 unsigned char block_id;
 int type; // 0: Empty, 1: Normal Block, 2: Target Block
 location_t location;
} block_t;

typedef struct {
 location_t cursor_loc;
 location_t selected; // set to 4,4 if none gets selected
 unsigned char win;
 block_t blocks[15];
} 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
```

Gamelogic.c

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>
#include "usbgamepad.h"
#include <stdint.h>
#include <stdbool.h>
#include "gamelogic.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
#include <unistd.h>

struct libusb_device_handle *gamepad;
uint8_t endpoint_address;

block_t create_block(int id)
{
   block_t b;
   b.block_id = id;
   b.type = 1;
   return b;
};
vga_ball_arg_t vla;
int vga_ball_fd;
int init;
char filename[] = "/dev/vga_ball";
unsigned char win;
void write2hw(vga_ball_arg_t vla)
{
   if (!init)
   {
       if ((vga_ball_fd = open(filename, O_RDWR)) == -1)
       {
           fprintf(stderr, "could not open %s\n", filename);
           return;
       }
       init = 1;
   }

   if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &vla))
   {
       perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
       return;
   }
}

bool cursor_moving(block_t arr[4][4], location_t location, int direction);
bool block_moving(block_t arr[4][4], location_t location, int direction);
void block_to_string(block_t arr[4][4], location_t cursor_location);
void write_hw(block_t arr[4][4],
             location_t location, location_t selected_loc);
bool arrays_are_equal(block_t arr1[4][4], block_t arr2[4][4])
{
   for (int i = 0; i < 4; i++)
   {
       for (int j = 0; j < 4; j++)
       {
           if (arr1[i][j].block_id != arr2[i][j].block_id)
           {
               return false;
           }
       }
   }
   return true;
}
```

```c
int main()
{
   int err, col;

   struct sockaddr_in serv_addr;

   int transferred;
   // char keystate[12];
   char msg[128];
   memset(msg, 0, 128);

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

   // init() the game state
   // object: move the block to top right corner

   // level one: 15 blocks,target one block under on the right corner

   // we will have a blocks.c file to deal with block moving solely
   // for game logic, please implement in this file!

   // declare variables
   bool selected_block;
   int selected_block_id;
   location_t cursor_location, selected_loc;
   block_t block0;
   block_t block1;
   block_t block2;
   block_t block3;
   block_t block4;
   block_t block5;
   block_t block6;
   block_t block7;
   block_t block8;
   block_t block9;
   block_t block10;
   block_t block11;
   block_t block12;
   block_t block13;
   block_t block14;
   block_t block15;
   win = 0;
   // initialize variables
   selected_block = false;
   selected_block_id = 0;
   cursor_location.x = 0;
   cursor_location.y = 0;
   selected_loc.x = 4;
   selected_loc.y = 4;
   block0 = create_block(0);
   block1 = create_block(1);
   block2 = create_block(2);
   block3 = create_block(3);
   block4 = create_block(4);
   block5 = create_block(5);
   block6 = create_block(6);
   block7 = create_block(7);
   block8 = create_block(8);
   block9 = create_block(9);
   block10 = create_block(10);
   block11 = create_block(11);
   block12 = create_block(12);
   block13 = create_block(13);
   block14 = create_block(14);
   block15 = create_block(15);
   block0.type = 0;
   block1.type = 2;
   // initialize block array


   // block_t init_arr[16] = {block0, block1, block2, block3, block4, block5, block6,
block7, block8, block9, block10, block11, block12, block13, block14, block15};
   // set winning array
   block_t arr[4][4] = {
       {block1, block2, block3, block4},
       {block5, block6, block7, block8},
       {block9, block10, block11, block12},
       {block13, block14, block0, block15}};
   for (int i = 0; i < 4; i++)
```

```
                    {
                        for (int j = 0; j < 4; j++)
                            {
                                arr[i][j].location.x = j;
                                arr[i][j].location.y = i;
                            }
                    }




   block_t winning_arr[4][4] = {
        {block1, block2, block3, block4},
        {block5, block6, block7, block8},
        {block9, block10, block11, block12},
        {block13, block14, block15, block0}};

   // random shuffle block array

   // produce a set of array
   //




   // DISPLAY THE CURRENT GAMESPACE

   write_hw(arr, cursor_location, selected_loc);

   bool game_end = 0;

   /* Look for and handle keypresses */
   for (;;)
   {
        uint8_t data[8];
        int actual_length;
        int r = libusb_interrupt_transfer(gamepad, endpoint_address, data,
sizeof(data), &actual_length, 0);

        if (r == 0 && actual_length == sizeof(data))
        {
            // Get pressed key
            uint8_t buttons = data[4];
```

```c
        // Sleep for 50 ms to avoid flooding the console with messages
        // usleep(500);

        // read game states:
        // int

        // game logic to handle block interaction based on the gamepad usage

        // left 6, right 2, up 0, down 4, A 40, B 72, X 24, Y 136

        if (buttons == 6 && game_end == 0)
        {
            // printf("this is the left arrow button\n");
            if (selected_block == false)
            {
                // TODO
                // cannot move to empty space
                selected_loc.x = 4;
                selected_loc.y = 4;
                printf("move the cursor to the left\n");
                if (cursor_moving(arr, cursor_location, 1))
                {
                    cursor_location.x -= 1;
                    vla.cursor_loc.x = cursor_location.x;
                    write_hw(arr, cursor_location, selected_loc);
                    block_to_string(arr, cursor_location);
                }
            }
            if (selected_block == true)
            {
                // TODO
                // need to verify the blocks movability
                write_hw(arr, cursor_location, cursor_location);
                // printf("move the block to the left\n");
                if (block_moving(arr, cursor_location, 1))
                {
                    block_t blk1 = arr[cursor_location.y][cursor_location.x];
                    block_t blk2 = arr[cursor_location.y][cursor_location.x - 1];
                    blk1.location.x--;
                    arr[cursor_location.y][cursor_location.x] = blk2;
                    arr[cursor_location.y][cursor_location.x - 1] = blk1;
                    cursor_location.x -= 1;

                    write_hw(arr, cursor_location, cursor_location);

                    block_to_string(arr, cursor_location);
                    if (arrays_are_equal(arr, winning_arr))
                    {
                        game_end =1;
                        win =1;
                        write_hw(arr, cursor_location, cursor_location);
                        // break;
                    }
                }
            }
        }
        else if (buttons == 2 && game_end == 0)
        {
            printf("this is the right arrow button\n");

            if (selected_block == false)
            {
                // TODO
                // cannot move to empty space
                selected_loc.x = 4;
                selected_loc.y = 4;
                // printf("move the cursor to the right\n");
                if (cursor_moving(arr, cursor_location, 3))
                {
                    cursor_location.x += 1;
                    vla.cursor_loc.x = cursor_location.x;
                    write_hw(arr, cursor_location, selected_loc);
                    block_to_string(arr, cursor_location);
                }
            }
            if (selected_block == true)
            {
                // TODO
                // need to verify the blocks movability
                write_hw(arr, cursor_location, cursor_location);
                printf("move the block to the right\n");
                if (block_moving(arr, cursor_location, 3))
                {
                    block_t blk1 = arr[cursor_location.y][cursor_location.x];
                    block_t blk2 = arr[cursor_location.y][cursor_location.x + 1];
                    blk1.location.x++;
        arr[cursor_location.y][cursor_location.x] = blk2;
                    arr[cursor_location.y][cursor_location.x + 1] = blk1;
        cursor_location.x += 1;

                    write_hw(arr, cursor_location, cursor_location);
                    block_to_string(arr, cursor_location);
                    if (arrays_are_equal(arr, winning_arr))
                        {
                            game_end =1;
                            win =1;
                            write_hw(arr, cursor_location, cursor_location);
                        }
                }
            }
        }
        else if (buttons == 0 && game_end == 0)
        {
            printf("this is the up arrow button\n");
            if (selected_block == false)
            {
                // TODO
                // cannot move to empty space

                // printf("move the cursor to the up\n");
                if (cursor_moving(arr, cursor_location, 2))
                {
                    cursor_location.y -= 1;
                    vla.cursor_loc.y = cursor_location.y;
                    write_hw(arr, cursor_location, selected_loc);
                    block_to_string(arr, cursor_location);
                }
            }
            if (selected_block == true)
            {
                // TODO
                // need to verify the blocks movability

                // printf("move the block to the up\n");
                write_hw(arr, cursor_location, cursor_location);
```

```
                if (block_moving(arr, cursor_location, 2))
                {
                    block_t blk1 = arr[cursor_location.y][cursor_location.x];
                    block_t blk2 = arr[cursor_location.y - 1][cursor_location.x];
                    blk1.location.y--;
                    arr[cursor_location.y][cursor_location.x] = blk2;
                    arr[cursor_location.y - 1][cursor_location.x] = blk1;

                    cursor_location.y -= 1;

                    write_hw(arr, cursor_location, cursor_location);
                    block_to_string(arr, cursor_location);
                    if (arrays_are_equal(arr, winning_arr))
                    {
                        game_end =1;
                        win =1;
                        write_hw(arr, cursor_location, cursor_location);

                    }
                }
            }
        }
        else if (buttons == 4 && game_end == 0)
        {
            printf("this is the down arrow button\n");
            if (selected_block == false)
            {
                // TODO
                // cannot move to empty space
                write_hw(arr, cursor_location, selected_loc);
                // printf("move the cursor to the down\n");
                if (cursor_moving(arr, cursor_location, 4))
                {
                    cursor_location.y += 1;
                    vla.cursor_loc.y = cursor_location.y;
                    write_hw(arr, cursor_location, selected_loc);
                    block_to_string(arr, cursor_location);
                }
            }
            if (selected_block == true)
            {
                // TODO
                // need to verify the blocks movability
                write_hw(arr, cursor_location, cursor_location);
                // printf("move the block to the down\n");
                if (block_moving(arr, cursor_location, 4))
                {
                    block_t blk1 = arr[cursor_location.y][cursor_location.x];
                    block_t blk2 = arr[cursor_location.y + 1][cursor_location.x];
                    blk1.location.y++;
        arr[cursor_location.y][cursor_location.x] = blk2;
                    arr[cursor_location.y + 1][cursor_location.x] = blk1;
        cursor_location.y += 1;

                    write_hw(arr, cursor_location, cursor_location);
                    block_to_string(arr, cursor_location);
                    if (arrays_are_equal(arr, winning_arr))
                    {
                        game_end =1;
                        win =1;
                        write_hw(arr, cursor_location, cursor_location);
                    }
                }
            }
        }
        else if (buttons == 40 && game_end == 0)
        {
            // printf("this is the A button\n");
            if (selected_block == false)
            {
                selected_block = true;
                write_hw(arr, cursor_location, cursor_location);
                printf("select the current block\n");
                block_to_string(arr, cursor_location);
            }
            else if (selected_block == true)
            {
                selected_block = false;
                write_hw(arr, cursor_location, selected_loc);
                printf("un-select the block\n");
                block_to_string(arr, cursor_location);
            }
        }
        else if (buttons == 72)
```

```
            {
                printf("this is the B button\n");
            }
            else if (buttons == 24)
            {
                block_to_string(arr, cursor_location);
            }
            else if (buttons == 136)

            {
                printf("this is the Y button\n");
                win = 0;

                game_end = 0;

                size_t i;
                srand(time(NULL));
                block_t init_arr[16] = {block0, block1, block2, block3, block4, block5,
block6, block7, block8, block9, block10, block11, block12, block13, block14, block15};
                for (i = 0; i < 16; i++)
                    {
                        size_t j = i + rand() / (RAND_MAX / (16 - i) + 1);

                        block_t t = init_arr[j];
                        init_arr[j] = init_arr[i];
                        init_arr[i] = t;
                    }
                for (int i = 0; i < 4; i++)
                    {
                        for (int j = 0; j < 4; j++)
                            {
                                arr[i][j] = init_arr[i * 4 + j];
                                arr[i][j].location.x = j;
                                arr[i][j].location.y = i;
                            }
                    }
                selected_block = false;
                write_hw(arr, cursor_location, selected_loc);



            }
        }
```

```
    }

    return 0;
}

block_t *blockat(block_t blocks[4][4], int id)
{
    int i, j;
    block_t *b;

    for (i = 0; i < 4; i++)
    {
        for (j = 0; j < 4; j++)
        {
            if (blocks[i][j].block_id == id)
            {
                printf("1\n");
                b = &blocks[i][j];
                return b;
            }
        }
    }
    return b;
}

void write_hw(block_t arr[4][4], location_t location, location_t selected_loc)
{
    int i, id;
    block_t *block;
    for (id = 1; id < 16; id++)
    {
        block = blockat(arr, id);
        vla.blocks[id - 1].block_id = id;
        vla.blocks[id - 1].location.x = block->location.x;
        vla.blocks[id - 1].location.y = block->location.y;
    }
    vla.selected.x = selected_loc.x;
    vla.selected.y = selected_loc.y;
    vla.win = win;
    vla.cursor_loc.x = location.x;
    vla.cursor_loc.y = location.y;
```

```
   for (i = 0; i < 15; i++)
   {
       printf("block [%hu]-x: %hu, y: %hu\n",
              vla.blocks[i].block_id, vla.blocks[i].location.x,
vla.blocks[i].location.y);
   }
   printf("cursor (%hu, %hu)\n", vla.cursor_loc.x, vla.cursor_loc.y);
   write2hw(vla);
}

bool cursor_moving(block_t arr[4][4], location_t location, int direction)
{
   if (direction == 1)
   { // left
       if (location.x == 0)
       {
           return false;
       }
       else if (arr[location.y][location.x - 1].type == 0)
       {
           return false;
       }
       else
       {
           return true;
       }
   }
   else if (direction == 2)
   { // up
       if (location.y == 0)
       {
           return false;
       }
       else if (arr[location.y - 1][location.x].type == 0)
       {
           return false;
       }
       else
       {
           return true;
       }
   }
```

```
    else if (direction == 3)
    { // right
        if (location.x == 3)
        {
            return false;
        }
        else if (arr[location.y][location.x + 1].type == 0)
        {
            return false;
        }
        else
        {
            return true;
        }
    }
    else
    { // down
        if (location.y == 3)
        {
            return false;
        }
        else if (arr[location.y + 1][location.x].type == 0)
        {
            return false;
        }
        else
        {
            return true;
        }
    }
}

bool block_moving(block_t arr[4][4], location_t location, int direction)
{
    if (direction == 1)
    { // left
        if (location.x == 0)
        {
            return false;
        }
        else if (arr[location.y][location.x - 1].type != 0)
        {
```

```
        return false;
    }
    else
    {
    return true;
    }
}
else if (direction == 2)
{ // up
    if (location.y == 0)
    {
        return false;
    }
    else if (arr[location.y - 1][location.x].type != 0)
    {
        return false;
    }
    else
    {
        return true;
    }
}
else if (direction == 3)
{ // right
    if (location.x == 3)
    {
        return false;
    }
    else if (arr[location.y][location.x + 1].type != 0)
    {
        return false;
    }
    else
    {
        return true;
    }
}
else
{ // down
    if (location.y == 3)
    {
        return false;
```

```c
       }
       else if (arr[location.y + 1][location.x].type != 0)
       {
           return false;
       }
       else
       {
         return true;
       }
   }
}

void block_to_string(block_t arr[4][4], location_t cursor_location)
{
   printf("cursor: %d, %d\n", cursor_location.x, cursor_location.y);
   for (int i = 0; i < 4; i++)
   {
       for (int j = 0; j < 4; j++)
       {
           if (arr[i][j].type == 0)
           {
               printf(" \t");
           }
           else
           {
               if (cursor_location.x == j && cursor_location.y == i)
               {
                   printf("%d*\t", arr[i][j].block_id);
               }
               else
               {
                   printf("%d\t", arr[i][j].block_id);
               }
           }
           // else if (arr[i][j].type == 1) {
           //     printf("%d\t", arr[i][j].block_id);
           //     // printf("X");
           // }
       }
       printf("\n");
   }
}
```

```
// return new array
// void shuffle_and_fill(block_t *init_arr, size_t len, block_t arr[][4])
// {
//     size_t i;
//     srand(time(NULL));
//     for (i = 0; i < len; i++)
//         {
//             size_t j = i + rand() / (RAND_MAX / (len - i) + 1);

//             block_t t = init_arr[j];
//             init_arr[j] = init_arr[i];
//             init_arr[i] = t;
//         }

//     for (int i = 0; i < 4; i++)
//         {
//             for (int j = 0; j < 4; j++)
//                 {
//                     arr[i][j] = init_arr[i * 4 + j];
//                     arr[i][j].location.x = j;
//                     arr[i][j].location.y = i;
//                 }
//         }
// }
```