CSEE 4840: Embedded Systems

Final Report: Donkey Kong

May 5, 2024

Ines Khouider (ik2512)

Ania Róża Krzyżańska (ark2219)

Sean Stothers (sps2308)

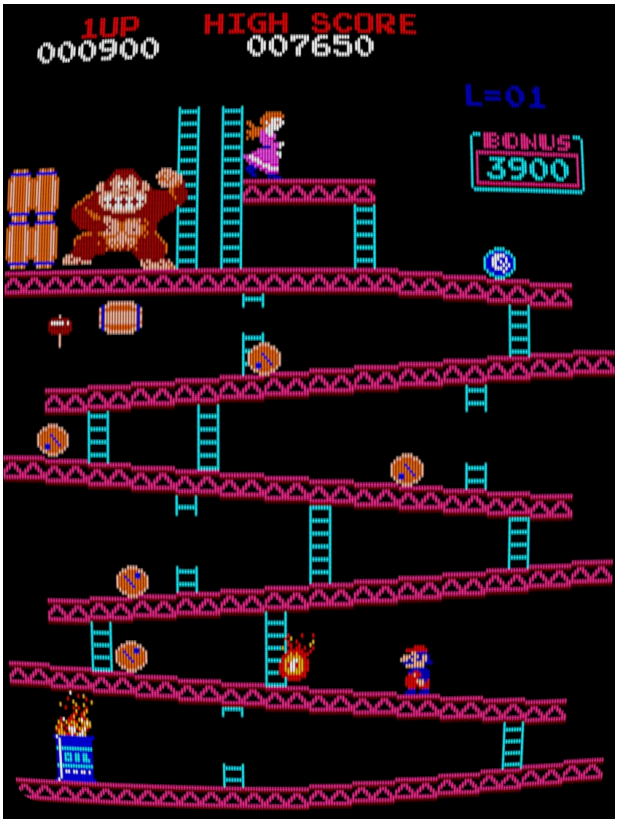


Table of Contents

I. Introduction

i) Game Overview

Our project is a basic replication of the first level of the popular 1981 Nintendo game, Donkey Kong. Using an NES controller connected to the FPGA via USB, the player can control Mario to avoid barrels rolling down ramps as they seek to reach Paulina at the top of the screen. The player can move Mario around the ramps, fall down off ramps, climb up ladders and jump over barrels. If the player is hit by a barrel, they die and the game automatically resets. If the objective is achieved and Mario reaches the top of the screen where Paulina is standing, Mario does a victory dance and the game likewise resets.

ii) Systems Block Diagram

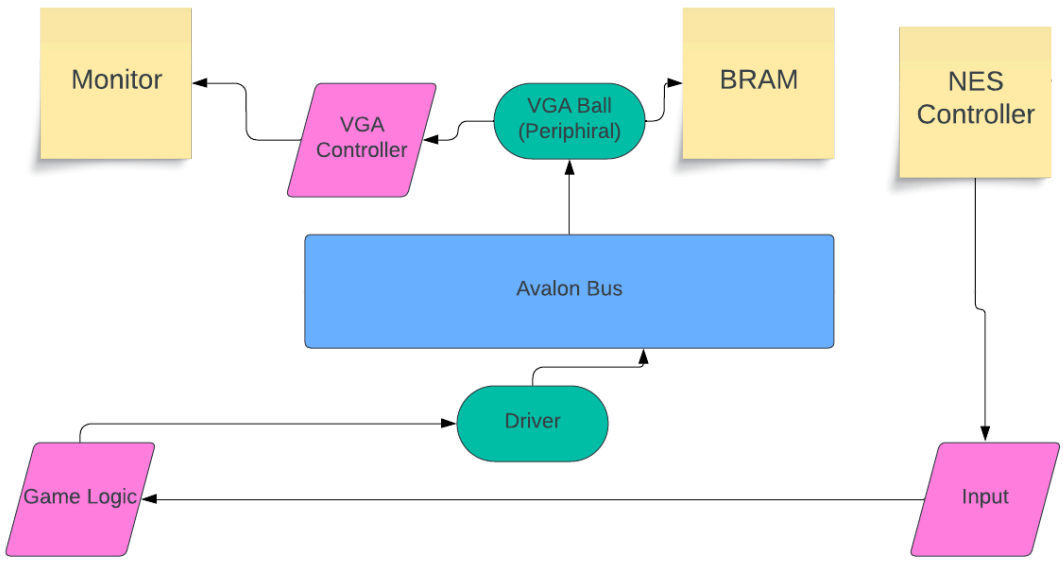


Figure 1. General System Architecture Diagram

Players use a USB-connected NES controller to pass in input which is read in using the controller protocol. The software uses this input to adjust the address and state of the mario sprite accordingly while also generating the motion of the barrels independent of the controller input. Data is written into registers and corresponding values are read out of the BRAM and sent to the monitor to be displayed. This then communicates with the FPGA through a device driver which sends the data through the 32-bit Avalon bus and to the VGA module (which we call VGA Ball).

In terms of software specially, all of the game logic is contained in a file called reset_and_barrel_spawn.c, including resetting the game, generating barrels, and limiting the bounds of Mario's motion. The usbkeyboard.c file implements the controller protocol and libusb library allowing us to access and read in input from the NES controller. Our vga_ball.c file writes the data into the registers and sends data to the FPGA hardware vag_ball.sv module via the Avalon bus.

In terms of hardware, our peripherals include the NES controller receiving input and a monitor ultimately displaying the intended graphics. The vga_ball.sv file interacts with Block RAM on the FPGA sending the appropriate address information to access the appropriate sprites to display at each coordinate on the screen pixel by pixel.

II. Hardware

i) Graphics–Background Tiles

Preloaded sprites will be stored in the RAM to be retrieved by their corresponding addresses. The background ramps and ladders as well as Paulina and Donkey Kong will be tiles instead of sprites.

Tile Block Diagram

For the background tiles we use hcount and vcount as a cursor traversing the entire display screen to determine the location at which we needed to display each pixel (described more in the Sprites section). This is first passed in for a RAM that is used to determine the current tile at which the "cursor" is located. Then based on the corresponding tile (which we had mapped out using a tile map) we again use hcount and count to determine the exact location of the cursor within the tile (i.e. the exact pixel we need to display). This second RAM outputs the corresponding 4 bit color code that then gets passed into the color table (this gets a bit more complicated if a sprite is located on the tile as well but, again, see the sprite graphics section for more details). The color table ultimately outputs a 24 bit RGB value that is then displayed for that pixel on the screen.

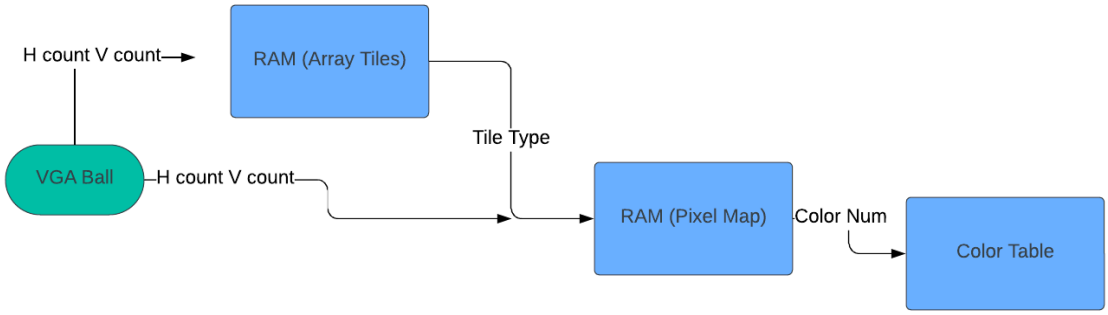


Figure 1. Background Tile block diagram

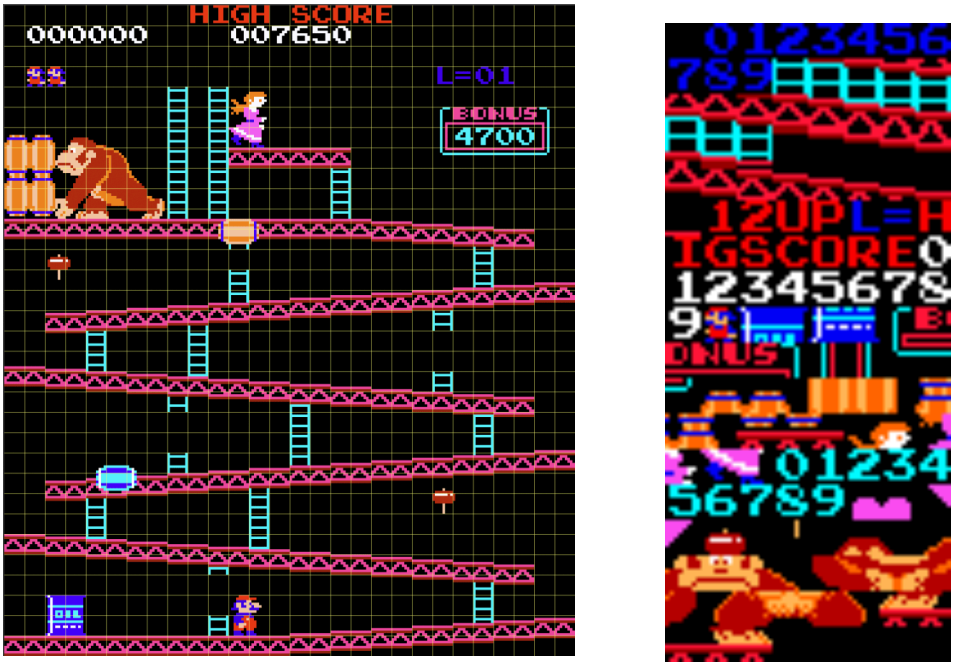


Figure 2. We used the above grid (left) when initially determining the tile layout. The tiles were stored in an array (right) and accessed accordingly,

ii) Graphics–Sprites

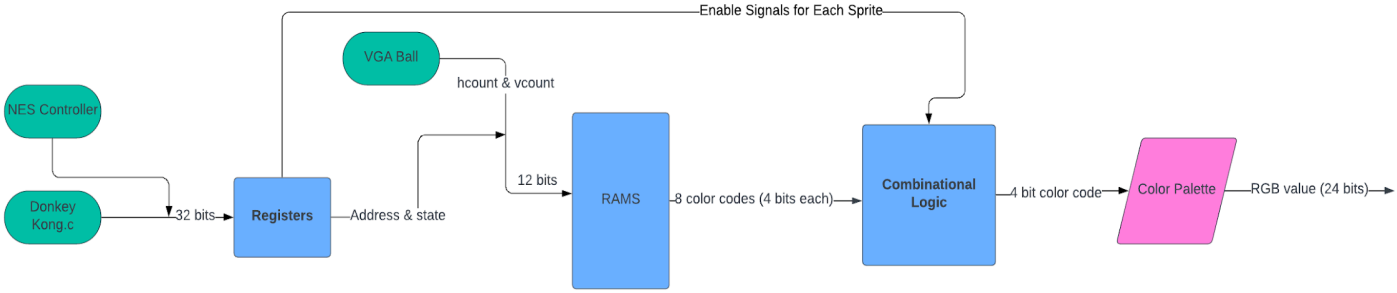


Figure 3. Sprite Block Diagram

Within the reset_and_barrel_spawn.c software, game logic or, in the case of the Mario sprite, controller input determine the desired coordinates at which we want the sprites to display. Since the barrels move without controller input, the game logic also determines if we want each barrel sprite to be visible, and each barrel has a dedicated bit in the state register that is high if the barrel should be displayed and low otherwise. The state register also indicates what particular sprite (e.g. Mario standing vs. Mario

jumping) should be displayed. This information is then sent to our vga module through an input/output control function.

After writing to the appropriate registers, the hardware can access the necessary sprite information. More specifically, the coordinate data from the registers is compared to hcount and vcount, signals indicating the x and y coordinates on the display screen that effectively act as a moving cursor traversing the entire length and width of the display screen and incrementing the coordinate values they represent accordingly. Each positive edge of a 50MHz clock cycle during this incrementation occurs, hcount and vcount are compared to the coordinate values stored in the registers for each sprite. If hcount is greater than the coordinate from a register and less than that coordinate plus the width of the sprite, and vcount is also in the appropriate range, the sprite has the potential to be displayed (i.e the enable signal for that sprite is set high).

First, the “address’ within the sprite (i.e. what pixel hcount and vcount are on) is determined using both hcount and vcount and the coordinates from the register. This address is then summed with additional bits indicating which sprite to display and is passed to the appropriate sprite RAM and used to select the corresponding pixel within the desired sprite. The resulting output is the 4-bit color code for this pixel. This value is then passed in as one of the inputs of a combinational logic block that goes through each of the sprites in order of priority (mario, then the barrels, then the background) and sets the value that will be passed into a color palette to the RAM output for a sprite only if that sprite’s enable signal is high and it’s color code is not “1111” which we used to represent a “hidden” sprite. The sprite color palette then takes whatever color code it’s passed in and outputs the appropriate RBG value which is then displayed on the monitor.

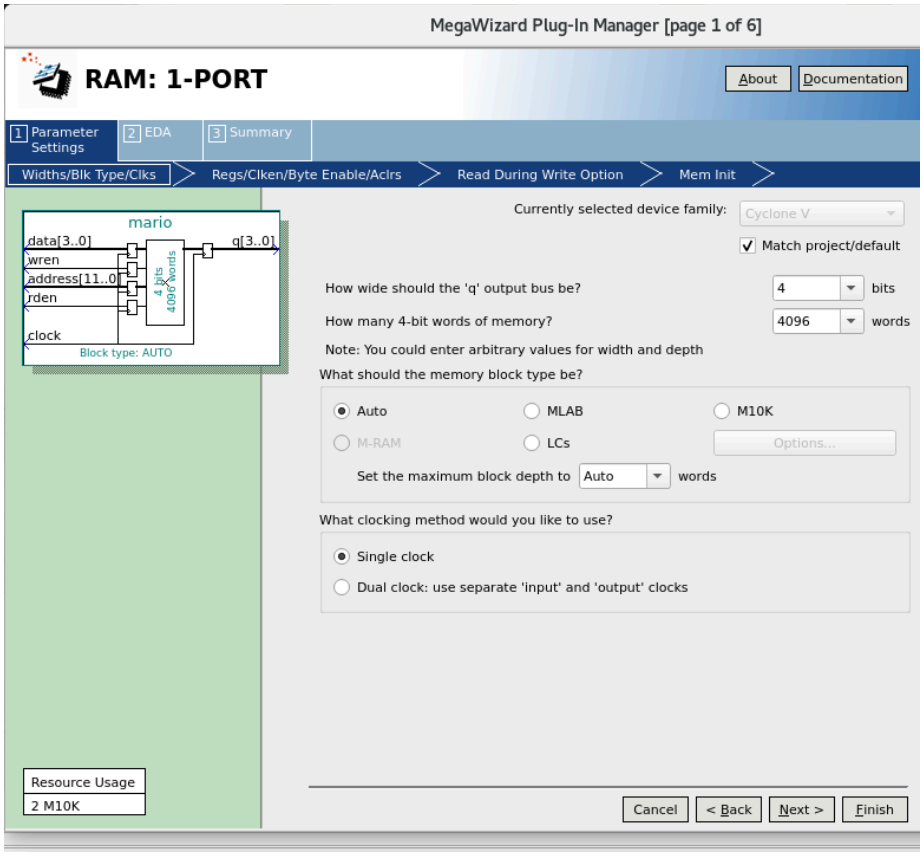
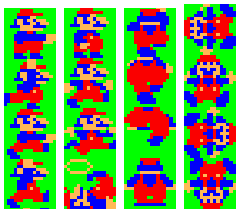
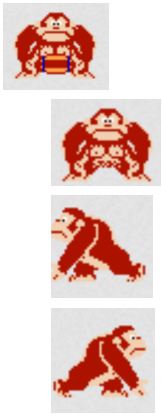
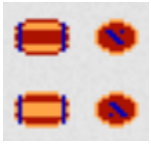
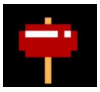


Figure 4. RAM for Mario: Here address indicates the location of the pixel within the RAM that we are trying to access. Both the Write Enable and Data signals were unused and the Read Enable signal was the corresponding enable signal for that sprite (described above). We used 4096 words for the 16 mario sprites, each made up of 16x16 pixels.

iii) Resource Budgets

A fundamental design constraint when using the FPGA is the amount of available memory. To use our available memory efficiently, we reduced the number of sprites used by roughly a half by storing their “flip” state in the register as a one bit value indicating if the sprite should be mirrored or not. This, combined with our use of a color palette to encode colors rather than storing the RGB values allowed us to stay well within the memory constraints.

Table 1. Memory used to store sprites and tiles

Category	Graphics	Size (bits)	Number of images	Total Size (bits)
Mario		32 x 32	16	3x32x32x16 = 49152
Donkey Kong		50x50	1	3x50x50x1 = 7500
Barrel		16x16	2	3x16x16x2 = 1536
Ladders		20x30	9	3x20x30x9= 16 200
Platform		20x30	15	3x20x30x15= 27000
Hammer		10x10	1	3x10x10x1= 300
Total				101, 688 bits

III. The Hardware/Software Interface

i) Controller Protocol:

We used an NES controller that connected to the Dev SoC board via USB port.
The read in controller input was as follows:

Nothing: 01 7f 7f 7f 7f 0f 00 00

Left: 01 7f 7f 00 7f 0f 00 00
Right: 01 7f 7f ff 7f 0f 00 00

Up: 01 7f 7f 7f 00 0f 00 00
Down: 01 7f 7f 7f ff 0f 00 00

A: 01 7f 7f 7f 7f 2f 00 00
B: 01 7f 7f 7f 7f 4f 00 00
A + B: 01 7f 7f 7f 7f 6f 00 00

Start: 01 7f 7f 7f 7f 0f 20 00
Select: 01 7f 7f 7f 7f 0f 10 00
Start + Select: 01 7f 7f 7f 7f 0f 30 00

Buttons change their separate bytes independently

The input from the controller was read into the software every cycle, with the exception of cycles in which mario was in the jump state. In these cases, input from the controller was read as though no button was pressed.

ii) Registers

Each register contains 4 bytes (32 bits), and hence ou. Data is stored in the big endian format. The first three registers stored the necessary data and address values needed to display the background tiles. Register 4 and registers 6-10 stored the mario coordinates and the coordinates for barrels 0-5 respectively. We ultimately chose to use separate registers for each of the barrels to allow for easier coordinate manipulation.

Register 5 was reserved for the "states" of mario and all the barrels. In our case the state of mario includes 3 bits indicating which Mario sprite to display as well as an additional bit to indicate the orientation of mario (where he's facing left or right). We did the same for the barrel sprites, except using only two bits to indicate which of the 4 sprites were displayed. Rather than storing twice as many sprites, we used the fact that many sprites are mirrors of each other (e.g. Mario walking left is just the Mario walking right sprite flipped along the y-axis). Thus, within the Verilog file, we simply subtracted the address coordinates of each sprite pixel from the maximum value inside the tile to display the mirror image.

Since we stored all the bits for these sprites in one register, we updated the state of each sprite by adding it to a bit masked version of the current sprite state to avoid zeroing out any existing values in the register (i.e. we'd zero out only the bits we wanted to update in the state register and add the bit shifted corresponding values to them).

Table 2. List of all registers and what information they contain

Register 1	Background Array Data
Register 2	Background Array Address
Register 3	Background Array Tile Map
Register 4	Mario coordinates: Bytes 1-2 x Bytes 3-4 y
Register 5 Sprite states (This register has information for	Bit 0 - Mario faces right or left Bit 1-4 - Which mario sprite Bit 5-6 which barrel_0 sprite

6 barrels but we only implemented barrels 0 through 4)	Bit 7 barrel_0 flip Bit 8-9 which barrel_1 sprite Bit 10 barrel_1 flip Bit 11-12 which barrel_2 sprite Bit 13 barrel_2 flip Bit 14-15 which barrel_3 sprite Bit 16 barrel_3 flip Bit 17-18 which barrel_4 sprite Bit 19 barrel_4 flip Bit 20-21 UNUSED (formerly which barrel_5 sprite) Bit 22 UNUSED (formerly barrel_5 flip) Bit 23 Barrel_0 on/off Bit 24 Barrel_1 on/off Bit 25 Barrel_2 on/off Bit 26 Barrel_3 on/off Bit 27 Barrel_4 on/off Bit 28 Barrel_5 on/off
Register 6	Barrel 0 coordinates
Register 7	Barrel 1 coordinates
Register 8	Barrel 2 coordinates
Register 9	Barrel 3 coordinates
Register 10	Barrel 4 coordinates
Register 11 (Made but unused)	Bit 0: Display/Hide Heart Bit 1 - Hammer 1 Visible Bit 2 - Hammer 2 Visible

Table 3. Color palette used to encode RGB values used for each pixel

Color Pallette	
0	#000000
1	#FFFFFF
2	#FF2155
3	#FA0000
4	#00FBFF
5	#970000
6	#FF6800
7	#FFB855
8	#0000F8
9	#B70000
10 (added)	#FA4EF2

IV. Software

i) Game Logic

We'll be using a NES controller to control the movements of Mario and to start the game.

Move_left/Move_right - at a high level, pressing the left button corresponds to shifting the mario sprite over to the left and pressing the right button corresponds to shifting the mario sprite to the right

More specifically, a left press following a right press first changes the displayed sprite to a sprite of mario facing left with his front leg extended leftward and also shifts the sprite left some number of pixels (exact number will be determined through testing). Further left presses will rotate cyclically between displaying mario with his front leg extended, mario with both legs together and mario with his back leg extended to create the illusion of walking.

Regardless of the "version" of mario displayed, the sprite will shift left the same number of pixels. Pressing and holding the left button results in continual motion leftward until a barrier (ie the edge of the screen) is hit.

The same holds for a right press following a left press but with rightward motion and mario facing rightward.

Jump

when the designated button (different from the up/down and left/right buttons) is pressed, the mario sprite displayed will be mario in a jumping position (facing left or right based on his position before the press) and will be shifted up 18 pixels gradually and then lowered back down the same number of pixels ("falling a certain portion of the way down"). If a left or right key is pressed while this occurs, the Mario sprite display won't change, but the sprite will likewise shift left or right accordingly.

Climb_up/climb_down - pressing the up button near a ladder shifts mario up and changes the sprite displayed

When the up button is pressed and the mario sprite is within the some experimentally determined range of pixels near a ladder, the sprite displayed will change to a sprite of mario from behind. If up is continually pressed or pressed and held down, there will be a cyclic rotation displaying mario between two poses (with left foot raised and with right foot raised) to create the illusion of him climbing the ladder until the sprite reaches some threshold of pixels at the top of the ladder, at which point the sprite displayed will show mario from behind and he will not be able to move up any further. Each press within the appropriate range of a ladder also simultaneously corresponds to shifting the mario sprite up by 2 pixels.

The same procedure applies to downward motion except the sprite can only move downward when the down button is pressed and the sprite is at the top of a ladder and stops when hitting a ramp at the bottom of the ladder.

Enemies - In our case, the “enemies” are the barrels that Mario must avoid. If Mario is hit by a barrel, he dies

Barrels will also be sprites, moving along the ramps and eventually leaving the field of view when they reach the bottom left corner of the screen and then resetting themselves at the top.

Paulina:

If a Mario sprite’s coordinates overlap with the tile containing Paulina, the player wins! Mario does a victory dance and the game resets.

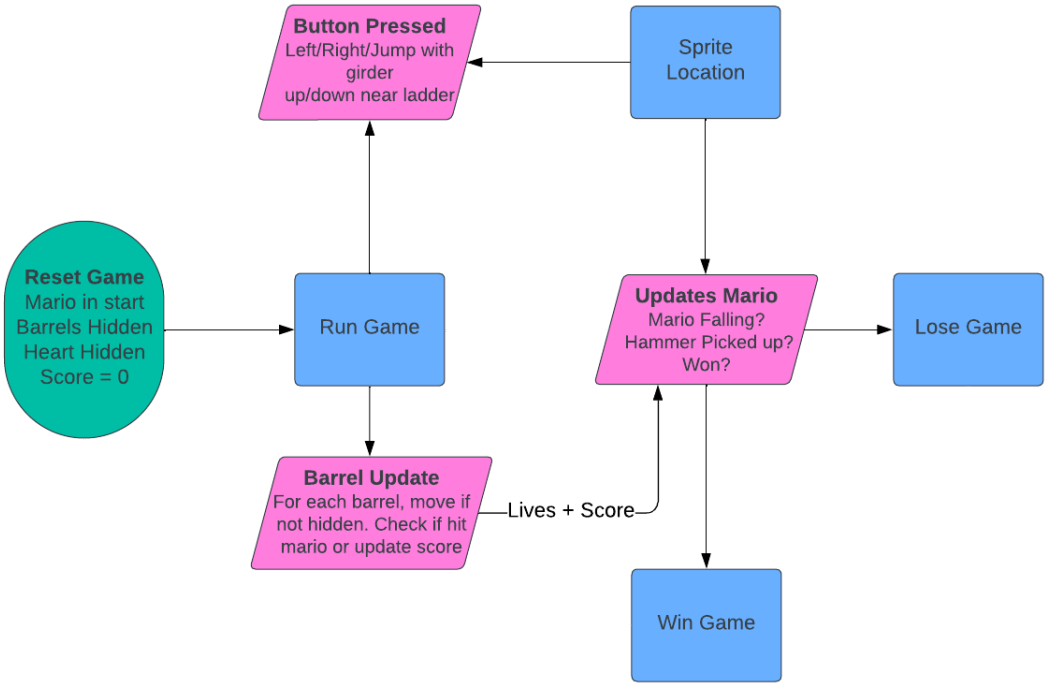


Figure 5. Game Logic

We used a revised 2 tile maps with integer values used to determine whether or not a ladder was present, if there was an edge a barrel could fall down, etc. With this logic we were able to create bounds on both the barrel motion and Mario's motion separately while allowing them to interact.We also implemented collision detection, causing Mario to "die" if he was within a certain range of the barrel sprite and the barrel was not hidden.

V. Discussion

i) Lessons Learned

- We found that an incremental design approach was best when it came to making efficient progress. We were most productive when we planned out our designs using block diagrams and discussed the pros and cons of each approach before actually attempting to implement it. We found that this allowed us to reject a lot of less promising ideas early on.
- Hardware is harder. Generally speaking, our group members were less familiar with the RAM generation process and the necessary verilog to interact with memory. Overtime we found that it was best to complete as many of our hardware goals as we could first and then focus on developing the software by implementing the existing hardware. We also found that this generally made debugging easier as, once we developed a solid hardware-software interface we were able to display the sprites and debug the software based on the displayed graphics.
- Perhaps one of our biggest challenges was figuring out how to use the tile map to manipulate the motion of our sprites. We ended up creating new tile maps with values indicating location where a barrel should fall, or where mario could climb up, for example. We also found the fact that the girders did not line up with the tiles to be a struggle as we had to compute separate offsets within each tile to allow the barrels and Mario to appear as though they were smoothly moving on top of them.

ii) Contributions

- Sean Stothers - in charge of the background tile map, vga_ball.sv hardware interaction, and mif generation
- Ania Krzyzanska- in charge of the controller protocol, sprite graphics, and the hardware software interface
- Ines Khoudier- in charge of software, particularly game logic and sprite animation

Note that we all helped each other with our tasks and largely did most of the planning as a group. With the unprecedented circumstances on campus, we also ended up having overlapping roles and generally working together on everything for the last leg of the project.

VI. Sources

http://www.ee.ic.ac.uk/pcheung/teaching/ee2_digital/de1-soc_user_manual.pdf

https://www.mariouniverse.com/sprites-nes-dk/

https://github.com/furrykef/dkdasm/blob/master/dkong-snd.asm

https://github.com/furrykef/dkdasm/blob/master/dkong-snd.asm

https://dev.to/joestrout/hwydt-donkey-kong-29me

For Graphics:

https://projectf.io/posts/fpga-graphics/

Audio:

https://zhehaomao.com/blog/fpga/2014/01/15/sockit-8.html

MIFs:

https://tomverbeure.github.io/2021/04/25/Intel-FPGA-RAM-Bitstream-Patching.html

Main software file: reset_and_barrel_spawn.c

```
/*

 *
 * CSEE 4840 Lab 2 for 2019
 * Ania Roza Krzyzanska ark2219
 * Ines Khouider ik2512
 * Sean Stothers sps2308
 */
//#include "fbputchar.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <fcntl.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <unistd.h>
#include "usbkeyboard.h"
#include "vga_ball.h"
#include <pthread.h>
#include <sys/ioctl.h>
#include <math.h>


#define SERVER_HOST "128.59.19.114"
#define SERVER_PORT 42000

#define BUFFER_SIZE 128

int sockfd; /* Socket file descriptor */

struct libusb_device_handle *keyboard;
uint8_t endpoint_address;
```

```
pthread_t network_thread;
void *network_thread_f(void *);
enum key_input{KEY_NONE, KEY_LEFT, KEY_RIGHT,KEY_UP, KEY_DOWN, KEY_A,
     KEY_B, KEY_START, KEY_SELECT};
int key_pressed;
int vga_ball_fd;

short MARIO = 0;
short BARREL1 = 1;
short BARREL2 = 2;
short BARREL3 = 3;
short BARREL4 = 4;
short BARREL5 = 5;
short DONKEYKONG = 6;
short HAMMER = 7;
short HEART = 8;
short SCORE_1 = 9; //leftmost
short SCORE_2 = 10;
short SCORE_3 = 11;
short SCORE_4 = 12; //rightmost

short DIR_RIGHT = 0;
short DIR_LEFT = 1;
short DIR_UPDOWN = 2;

struct Coordinates {
   short x;
   short y;
};

struct Labels{
   short tag;
   short direction;
};
```

```
struct Sprite {
   short x;
   short y;
   short tag;
   short on;
   short direction;
   short what_spite;
};

struct Barrels
{
   struct Sprite b1;
   struct Sprite b2;
   struct Sprite b3;
   struct Sprite b4;
   struct Sprite b5;

   struct Sprite barrel_list[5];

};


//sprite locations
short LOC_LINE = 0;
short LOC_SLOPE = 1;
short LOC_LADDER = 2;
short LOC_HAM = 3;
short LOC_FALL = 4;
short LOC_WIN = 5;
short OUT_OF_BOUNDS = 6;
short LOC_SNAP_UP = 7;

struct Result
```

```
{
      int loc;
      int tag;
   int snap;
};

struct Reset_Sprite{
  struct Sprite mario;
  struct Sprite b1;
  struct Sprite b2;
  struct Sprite b3;
  struct Sprite b4;
  struct Sprite b5;

};




static int collision(struct Sprite mario, struct Sprite barrel){
   int mx = mario.x + 4;
   int mxmax = mario.x + 32 - 4 ;

      int mtop = mario.y;
   int mbottom = mario.y + 32;

   int bx = barrel.x  + 4; //barrel is 24 pixels
   int bxmax = barrel.x + 32 - 4;

      int btop = barrel.y + 20;
   int bbottom = barrel.y + 32;
```

```
  if (mx > bx && mx < bxmax || mx < bx && mx > bxmax){

        if ((mtop >= bbottom && mtop <= btop) ||(mtop <= bbottom && mtop >= btop)){
            return 1;
        }
        if ((mbottom >= bbottom && mbottom <= btop) || (mbottom <= bbottom && mbottom
  >= btop)){
            return 1;
        }

  }
    else if ((mxmax > bxmax && mxmax < bx) || (mxmax < bxmax && mxmax > bx)){

        if (mtop >= bbottom && mtop <= btop || mtop <= bbottom && mtop >= btop){

            return 1;

        }
        if ((mbottom >= bbottom && mbottom <= btop) || (mbottom <= bbottom && mbottom
  >= btop)){
            return 1;

        }
    }
  return 0;
}
//mario deaths
int MARIO_DEATH_1 = 3;
int MARIO_DEATH_2 = 12;
int MARIO_DEATH_3 = 13;
int MARIO_DEATH_4 = 14;
int MARIO_DEATH_5 = 15;

//mario win dance
```

```
int MARIO_HAMMER1 = 8;
int MARIO_HAMMER2 = 9;
int MARIO_HAMMER3 = 10;

static void mario_die(struct Sprite mario, struct Sprite b1, struct Sprite b2, struct Sprite b3, struct
    Sprite b4, struct Sprite b5){
    for (int death_cycle = 0; death_cycle < 400; death_cycle ++){
            if (death_cycle < 100){
                        mario.tag = MARIO_DEATH_1;
            }
            else if ((death_cycle % 100) < 25){
                                    mario.tag = MARIO_DEATH_2;
                }
            else if ((death_cycle % 100) < 50){
                                    mario.tag = MARIO_DEATH_3;
                }
            else if ((death_cycle % 100) < 75){
                                    mario.tag = MARIO_DEATH_4;
                }
            else if ((death_cycle % 100) < 100){
                                    mario.tag = MARIO_DEATH_5;
            }
            send_to_HW(mario, b1, b2, b3, b4, b5, MARIO);

            usleep(20000);
    }

}

static void mario_hammer(struct Sprite mario, struct Sprite b1, struct Sprite b2, struct Sprite b3, struct
    Sprite b4, struct Sprite b5){
    for (int hammer_cycle = 0; hammer_cycle < 500; hammer_cycle ++){
```

```
            if ((hammer_cycle % 75) < 25){
                mario.tag = MARIO_HAMMER1;
            }
            else if ((hammer_cycle % 75) < 50){
                mario.tag = MARIO_HAMMER2;
            }
            else if ((hammer_cycle % 75) < 75){
                mario.tag = MARIO_HAMMER3;
            }
            if ((hammer_cycle % 75) == 0){
                if (mario.direction == 0){
                    mario.direction = 1;
                }
                else{
                    mario.direction = 0;
                }
            }
            send_to_HW(mario, b1, b2, b3, b4, b5, MARIO);

            usleep(20000);
    }

}

uint8_t tile_map_mario[30][35] = {
{0,0,0,0,0,0,0,0,0,0,0,0,0,11,0,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,11,0,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,11,0,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,11,0,11,19,19,19,6,19,19,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,11,0,11,0,0,0,0,0,11,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,11,0,11,0,0,0,0,0,11,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,20,20,20,20,20,20,20,20,12,20,12,20,20,20,20,20,12,20,84,84,85,85,86,86,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,100,46,19,19,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11,0,0,0,0,0,0,0},
```

```
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,90,70,91,91,92,92,0,0},
{0,0,0,0,0,0,0,0,0,0,0,90,90,91,91,92,92,93,93,94,94,95,95,96,96,19,19,100,100,0,0,90,90,0,0},
{0,0,0,0,0,0,0,96,96,15,19,100,100,0,51,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,11,0,0,0,0,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,85,85,86,86,11,0,0,0,0,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,100,100,13,19,80,80,81,61,82,82,83,83,84,84,85,85,86,86,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11,0,0,0,100,100,19,19,80,80,81,81,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11,0,0,0,0,0,0,0,0,40,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11,0,0,0,0,0,0,0,0,11,0,0,90,90,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,90,90,91,91,72,92,93,93,94,94,95,95,96,76,19,19,100,100,0,0},
{0,0,0,0,0,0,0,94,94,95,95,96,96,19,19,100,100,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,51,0,0,0,0,0,0,0,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,11,0,0,0,0,0,0,0,11,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,19,19,80,80,61,81,82,82,83,83,84,84,65,85,86,86,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,100,100,19,19,80,80,81,81,82,82,83,83,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,42,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,11,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,90,90,91,91,92,92,93,93,94,74,95,95,96,96,0,0},
{0,0,0,0,0,19,19,19,19,19,19,19,19,19,19,19,19,19,19,100,100,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}};

uint8_t tile_map_barrel[30][35] = {
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,20,20,20,20,20,20,5,5,20,20,20,20,20,20,20,20,20,20,84,84,85,85,86,86,0,0,0,0,0,0},
{0,0,0,0,0,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100
    ,100,100,100,19,19,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,90,90,91,91,92,92,0,0},
```

```
{0,0,0,0,0,0,0,0,0,0,0,90,90,91,91,92,92,93,93,94,94,95,95,96,96,18,18,100,100,100,100,100,100,0,0
},
{0,0,0,0,0,0,0,96,96,18,18,100,100,100,100,100,100,100,100,100,100,100,100,100,100,0,0,0,0,0,0,0,
0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,85,85,86,86,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,100,100,100,100,19,19,80,80,81,81,82,82,83,83,84,84,85,85,86,86,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,100,100,100,100,100,100,100,100,100,100,100,100,100,100,19,19,80,80,81,81,
0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,100,100,100,100,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,90,90,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,90,90,91,91,92,92,93,93,94,94,95,95,96,96,18,18,100,100,0,0},
{0,0,0,0,0,0,0,94,94,95,95,96,96,18,18,100,100,100,100,100,100,100,100,100,100,100,100,100,100,0
,0,0,0,0,0},
{0,0,0,0,0,0,0,100,100,100,100,100,100,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,19,19,80,80,81,81,82,82,83,83,84,84,85,85,86,86,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,100,100,100,100,100,100,100,100,100,100,100,100,100,100,19,19,80,80,81,81,82,82,8
3,83,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,100,100,100,100,100,100,100,100,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,90,90,91,91,92,92,93,93,94,94,95,95,96,96,0,0},
{0,0,0,0,0,6,6,18,18,18,18,18,18,18,18,18,18,18,18,100,100,100,100,100,100,100,100,100,100,1
00,100,100,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0},
{0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0}};

//BarrelGoing from (368, 127) to Going from (368, 128)

short BARREL_HIDDEN = 0;
short BARREL_CIRCLE_VIEW = 2;
short BARREL_CIRCLE_VIEW_FLIP = 3;
short BARREL_SIDEVIEW = 2;
```

```
//Barrel Flipping
short BARREL_NORM = 0;
short BARREL_FLIP = 1;



short BFALL = 0;
short BRIGHT = 1;
short BRIGHTSLANT = 2;

short BLEFT = 3;
short BLEFTSLANT = 4;
short BSTART = 5; //this is also 1 technically

static struct Result SpriteLocation(short x, short y, int old_loc, int old_tag){
int tile_num_x = (int)(x/16);
      int tile_num_y = (int)(y/16);
      int offset;



   uint8_t code = tile_map_mario[tile_num_y][tile_num_x];



   struct Result res;
     res.loc = old_loc;
     res.tag = old_tag;

   // printf("x is %d y is %d and tile number is %d", tile_num_x, tile_num_y, code);

   if (code == 0) {
   //    printf("FALLING");
       res.loc = LOC_FALL;
   } else if (code == 6) {
     res.loc = LOC_WIN;
```

```
} else if ((11 <= code) && (100 >= code)){ //
  if ((code == 19) || (code == 13) || (code == 15)) { //no offset
  //    printf("LINE");
     res.loc = LOC_LINE;
  } else if (((code / 10) == 8) || ((code / 10) == 9) || ((code / 10) == 6) || ((code / 10) == 7)) { //offset
     if (((code / 10) == 9) || ((code / 10) == 7)) {
        offset = 14 - (2 * (code % 10));// maps code to offset from 16 tile grid (ex. 96 -> 2, 90 -> 14)
     } else {
           offset = 2 + (2 * (code % 10));// maps code to offset from 16 tile grid (ex. 86 -> 14, 80 ->
  2)
     }
      if ((((y / 16) * 16) + offset) > y) {
  //          printf("FALLING");
          res.loc = LOC_FALL;
     } else if ((((y / 16) * 16) + offset) < y) {
           //printf("SNAP UP");
           res.loc = LOC_SNAP_UP;
           res.snap = (((y / 16) * 16) + offset) - y;
     } else {
           //printf("LINE");
           res.loc = LOC_LINE;
     }
  } else if ((code == 20) || (code == 12)) { //offset for no slope (numbers don't fit other equation)
     offset = 8;// always 8
      if ((((y / 16) * 16) + offset) > y) {
  //          printf("FALLING");
          res.loc = LOC_FALL;
     } else if ((((y / 16) * 16) + offset) < y) {
  //          printf("SNAP UP");
           res.loc = LOC_SNAP_UP;
           res.snap = (((y / 16) * 16) + offset) - y;
     } else {
           //printf("LINE");
           res.loc = LOC_LINE;
```

```
        }
    } else if (code == 100) {
            res.loc = LOC_SNAP_UP;
            res.snap = -2;
    }
  }
  if ((code == 11) || ((code / 10) == 4) || ((code / 10) == 5)){ //ladder tops here too for now
    if ((key_pressed == KEY_DOWN) || (key_pressed == KEY_UP)) {
     //    printf("LADDER should only go down and up");
        res.loc = LOC_LADDER;
        res.tag = DIR_UPDOWN;
    } else {
    //    printf("LINE");
       res.loc = LOC_LINE;
    }
  } else if (code == 12 || ((code / 10) == 6) || ((code / 10) == 7)){ //ladder bottoms
            if ((key_pressed == KEY_UP)) {
    //                      printf("LADDER can go up and down and left anf right ON SLOPE");
                            res.loc = LOC_LADDER;
                            res.tag = DIR_RIGHT;
            } else {
            //          printf("LINE");
                        res.loc = LOC_LINE;
            }
  }
     //if (res.loc == -1) {
            //exit(1);
    //}

  return res;

}
```

```
vga_ball_sprite_state_t st = {.state=0};
vga_ball_sprite_state_t st1 = {.state=0};


static int get_location(short x, short y){
   int tile_num_x = (int)(x/16);
     int tile_num_y = (int)(y/16);

   int code = tile_map_barrel[tile_num_y][tile_num_x];

   return code;

}


/* Set the background color */
void send_to_HW(struct Sprite sprite, struct Sprite barrel0, struct Sprite barrel1, struct Sprite barrel2,
     struct Sprite barrel3, struct Sprite barrel4, short type)
{
        vga_ball_arg4_t vla4;
         //printf("MARIO\n");
         vga_ball_xy_t coord;
         vga_ball_arg3_t vla3;
        // fprintf(stderr, "\n coordinates: (%u, %u)\n", sprite.x, sprite.y);
         coord.x_coor = sprite.x;
         coord.y_coor = sprite.y;
         //sprite.tag = indicate;
         st.state = (sprite.tag << 1) + sprite.direction + (st.state & 4294967264); //FFFFFFE0 in decimal
         // printf("Mario state %x\n", st.state);
         vla3.coor = coord;
         vla4.m_state = st;
         if (ioctl(vga_ball_fd, VGA_BALL_MARIO_COORDINATES, &vla3))
         {
            perror("ioctl(VGA_BALL_MARIO_COORDINATES) failed");
```

```
        return;
    }


 vga_ball_barrel0_xy_t coord0;
 vga_ball_arg5_t vla5;
 coord0.bx0 = barrel0.x;
 coord0.by0 = barrel0.y;
 vla5.coor = coord0;
 // printf("writing to barrel 0 (%d,%d)\n", barrel0.x, barrel0.y);
 vga_ball_barrel1_xy_t coord1;
 vga_ball_arg6_t vla6;
 coord1.bx1 = barrel1.x;
 coord1.by1 = barrel1.y;
 vla6.coor = coord1;
    //   printf("writing to barrel 1 (%d,%d)\n", barrel1.x, barrel1.y);
vga_ball_barrel2_xy_t coord2;
 vga_ball_arg7_t vla7;
 coord2.bx2 = barrel2.x;
 coord2.by2 = barrel2.y;
 vla7.coor = coord2;
 // printf("writing to barrel 2 (%d,%d)\n", barrel2.x, barrel2.y);
vga_ball_barrel3_xy_t coord3;
 vga_ball_arg8_t vla8;
 coord3.bx3 = barrel3.x;
 coord3.by3 = barrel3.y;
 vla8.coor = coord3;
 // printf("writing to barrel 3 (%d,%d)\n", coord3.bx3, coord3.by3);
vga_ball_barrel4_xy_t coord4;
 vga_ball_arg9_t vla9;
 coord4.bx4 = barrel4.x;
 coord4.by4 = barrel4.y;
 vla9.coor = coord4;
 // printf("writing to barrel 4 (%d,%d)\n", coord4.bx4, coord4.by4);
```

```
 st.state = (barrel0.tag << 5) + (barrel0.direction << 7) + (st.state & 4294967071);   // this is the value of the bitmask with 32 1s and 0s at bits 5-7 (11111111111111111111100011111111)
 st.state = (barrel1.tag << 8) + (barrel1.direction << 10) + (st.state & 4294965503);  // in binary 1111_1111_1111_1111_1111_1000_1111_1111
 st.state = (barrel2.tag << 11) + (barrel2.direction << 13) + (st.state & 4294952959);  // in binary 1111_1111_1111_1111_1111_1000_1111_1111
st.state = (barrel3.tag << 14) + (barrel3.direction << 16) + (st.state & 4294852607);
st.state = (barrel4.tag << 17) + (barrel4.direction << 19) + (st.state & 4294049791);
st.state = (barrel0.on << 23) + (st.state & (~(1 << 23)));
 st.state = (barrel1.on << 24) + (st.state & (~(1 << 24)));
st.state = (barrel2.on << 25) + (st.state & (~(1 << 25)));
st.state = (barrel3.on << 26) + (st.state & (~(1 << 26)));
st.state = (barrel4.on << 27) + (st.state & (~(1 << 27)));

 vla4.m_state = st; //
 // printf("\n Barrel state %x\n", st.state);
if (ioctl(vga_ball_fd, VGA_BALL_BARREL4_COORDINATES, &vla9))
 {
    perror("ioctl(VGA_BALL_b4) failed");
    return;
 }
if (ioctl(vga_ball_fd, VGA_BALL_BARREL3_COORDINATES, &vla8))
 {
    perror("ioctl(VGA_BALL_b3) failed");
    return;
 }
if (ioctl(vga_ball_fd, VGA_BALL_BARREL2_COORDINATES, &vla7))
 {
    perror("ioctl(VGA_BALL_b2) failed");
    return;
 }
 if (ioctl(vga_ball_fd, VGA_BALL_BARREL1_COORDINATES, &vla6))
 {
```

```
            perror("ioctl(VGA_BALL_b2) failed");
            return;
        }
        if (ioctl(vga_ball_fd, VGA_BALL_BARREL0_COORDINATES, &vla5))
        {
            perror("ioctl(VGA_BALL_b1) failed");
            return;
        }
        if (ioctl(vga_ball_fd, VGA_BALL_SPRITE_STATE, &vla4))
        {
            perror("ioctl(VGA_BALL_SPRITE_STATE) failed");
            return;
        }
}
static struct Coordinates move(short x, short y, short movement_x, short movement_y){
    //these represent the walls
    short new_x = x + movement_x;
    short new_y = y + movement_y;

    //check in range of pixels the donkey kong map is at
    int begin_x = 4*16;
    int end_x = 35*16;
    int begin_y = 0*16;
    int end_y = 29*16;

    if (new_x < begin_x){
      new_x = begin_x;
    }
    if (new_x > end_x){
      new_x = end_x;
    }
    if (new_y < begin_y){
      new_y = begin_y;
    }
```

```
    if (new_y > end_y){
      new_y = end_y;
    }

    // fprintf(stderr, "Going from (%d, %d) to ", x, y);
    // fprintf(stderr, "Going from (%d, %d)\n", new_x, new_y);


    struct Coordinates result = { new_x, new_y };
    return result;

}

struct Reset_Sprite reset(){
   struct Reset_Sprite res;
   struct Sprite mario;
   struct Sprite b1;
   struct Sprite b2;
   struct Sprite b3;
   struct Sprite b4;
   struct Sprite b5;

    mario.x = 5*16 +8;
    mario.y = 27*16;
    mario.tag = 0;
    mario.on = 1;
    mario.direction = 0;

    b1.x = 0*16;
    b1.y = 6*16;
    b1.tag = BARREL_CIRCLE_VIEW;
    b1.direction = BARREL_NORM;
    b1.on = 0;
```

```
    b2.x = 0*16; //change this later
    b2.y = 27*16;
    b2.tag = BARREL_CIRCLE_VIEW;
    b2.direction = BARREL_NORM;
    b2.on = 0;

    b3.x = 1*16; //change this later
    b3.y = 27*16;
    b3.tag = BARREL_CIRCLE_VIEW;
    b3.direction = BARREL_NORM;
    b3.on = 0;

    b4.x = 0*16; //change this later
    b4.y = 11*16;
    b4.tag = BARREL_CIRCLE_VIEW;
    b4.direction = BARREL_NORM;
    b4.on = 0;

    b5.x = 0*16; //change this later
    b5.y = 7*16;
    b5.tag = BARREL_CIRCLE_VIEW;
    b5.direction = BARREL_NORM;
    b5.on = 0;

    res.mario = mario;
    res.b1  =b1;
    res.b2 = b2;
    res.b3 = b3;
    res.b4 = b4;
    res.b5 = b5;
    send_to_HW(mario, b1, b2, b3, b4, b5, MARIO);
    return res;
}
struct Sprite update_barrel(struct Sprite b) {
```

```
struct Coordinates coords;
int offset;
int loc = get_location(b.x, b.y - 6); //account for the fact the barrel is shifted down by 6 since the bottom of the sprite is transparent
if (loc != 0) {
    printf("\nlocation is %d: %d, %d\n", loc, b.x, b.y);
}

if (loc == 0){
    //printf("location is 0, shoudl be falling");
    coords = move(b.x, b.y,0,1);
    b.x = coords.x;
    b.y = coords.y;
 }
 if (loc == 20 || loc == 5 || loc == 19) { //flat right offset 8
    //printf("location is 2, should be right ");
   if ((loc == 20) || (loc == 5)) {
            offset = 6 + 8;
   } else {
            offset = 6;
   }
   if (((((b.y - 6) / 16) * 16) + offset) > b.y) {
          //fall
          coords = move(b.x, b.y,0,1);
   } else if (((((b.y - 6) / 16) * 16) + offset) < b.y) {
          //snap up
          coords = move(b.x, b.y,0,(((b.y / 16) * 16) + offset) - b.y);
   } else {
          //go horizontal
          coords = move(b.x, b.y,1,0);
   }
b.x = coords.x;
 b.y = coords.y;
 }
```

```
if (((loc / 10) == 9)){ //slope positive (roll south west)

   offset = 6 + (14 - (2 * (loc % 10)));
  if (((((b.y - 6) / 16) * 16) + offset) > b.y) {
        //fall
        coords = move(b.x, b.y,0,1);
  } else if (((((b.y - 6) / 16) * 16) + offset) < b.y) {
        //snap up
        //res.loc = LOC_SNAP_UP;
        //res.snap = (((b.y / 16) * 16) + offset) - b.y;
        coords = move(b.x, b.y,0,((((b.y - 6) / 16) * 16) + offset) - b.y);
  } else {
        //go horizontal
        coords = move(b.x, b.y,-1,0);
  }
  b.x = coords.x;
   b.y = coords.y;
}
if (((loc / 10) == 8)){ //slope negative (roll south east)
   offset = 6 + (2 + (2 * (loc % 10)));
   if (((((b.y - 6) / 16) * 16) + offset) > b.y) {
        //fall
        printf("\ngirder location: %d\n", ((((b.y - 6) / 16) * 16) + offset));
        coords = move(b.x, b.y,0,1);
  } else if (((((b.y - 6) / 16) * 16) + offset) < b.y) {
        //snap up
        coords = move(b.x, b.y,0,((((b.y - 6) / 16) * 16) + offset) - b.y);
  } else {
        //go horizontal
        coords = move(b.x, b.y,1,0);
  }
  b.x = coords.x;
   b.y = coords.y;
}
```

```
if (loc == 18) { //flat go left
   offset = 6;
   if (((((b.y - 6) / 16) * 16) + offset) > b.y) {
         //fall
         coords = move(b.x, b.y,0,1);
   } else if ((((b.y / 16) * 16) + offset) < b.y) {
         //snap up
         coords = move(b.x, b.y,0,(((b.y / 16) * 16) + offset) - b.y);
   } else {
         //go horizontal
         coords = move(b.x, b.y,-1,0);
   }
   b.x = coords.x;
    b.y = coords.y;
}

if (loc == 100) {
   offset = 6;
   if (((((b.y - 6) / 16) * 16) + offset) > b.y) {
         //snap up
         coords = move(b.x, b.y,0,0);
   } else {
         //fall
         coords = move(b.x, b.y,0,1);
   }
   b.x = coords.x;
    b.y = coords.y;
}
 if (loc == 6){
    b.x = 5*16+8; //loop back to start
   b.y = 6*16;
 }

return b;
```

```
}


int main()
{

   //struct usb_keyboard_packet pac72:16: error: expected identifier or '(' before ')' token
   unsigned char data[8];
   int transferred;

   static const char filename[] = "/dev/vga_ball";

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


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

   //write mario to start position
   int no_jump =1;
   int jump_cycle = 0;
   int walk_cycle = 0;


   struct Reset_Sprite reset_val= reset();
```

```
struct Sprite barrels[5];


barrels[0] = reset_val.b1;
barrels[1] = reset_val.b2;
barrels[2] = reset_val.b3;
barrels[3] = reset_val.b4;
barrels[4] = reset_val.b5;
struct Sprite mario = reset_val.mario;

int loc = get_location(barrels[1].x, barrels[1].y);
struct Coordinates coords;



printf("starting program");

int walk_counter_r = 0;
int walk_counter_l = 0;
int offset;

int location  = -1;
int direction = -1;

int spawn_barrels_counter = 0;
for (;;) {

        if (collision(mario, barrels[0]) == 1 || collision(mario, barrels[1]) == 1 ||collision(mario,
  barrels[2]) == 1 ||collision(mario, barrels[3]) == 1 ||collision(mario, barrels[4]) == 1){
                mario_die(mario, barrels[0], barrels[1], barrels[2], barrels[3], barrels[4]);
                spawn_barrels_counter = 0;
                reset_val = reset();

                barrels[0] = reset_val.b1;
                barrels[1] = reset_val.b2;
```

```
                        barrels[2] = reset_val.b3;
                        barrels[3] = reset_val.b4;
                        barrels[4] = reset_val.b5;
                        mario = reset_val.mario;
                        send_to_HW(mario, barrels[0], barrels[2], barrels[2], barrels[3],
barrels[4], MARIO);

        }

 struct Result current_loc = SpriteLocation(mario.x, mario.y, location, direction);

 int location = current_loc.loc;
 int direction = current_loc.tag;
int snap_up = current_loc.snap;

   if (location == LOC_WIN) {
            mario_hammer(mario, barrels[0], barrels[1], barrels[2], barrels[3], barrels[4]);
           spawn_barrels_counter = 0;
           reset_val = reset();

           barrels[0] = reset_val.b1;
           barrels[1] = reset_val.b2;
           barrels[2] = reset_val.b3;
           barrels[3] = reset_val.b4;
           barrels[4] = reset_val.b5;
           mario = reset_val.mario;
           send_to_HW(mario, barrels[0], barrels[2], barrels[2], barrels[3], barrels[4], MARIO);
   }

if (((spawn_barrels_counter % 500) == 0) && (barrels[spawn_barrels_counter / 500].on == 0)) {
//spawn 1 every 50, check if barrel offscreen ak not spawn yet
        barrels[spawn_barrels_counter / 500].x = (5 * 16) + 8;
        barrels[spawn_barrels_counter / 500].y = 6 * 16;
        barrels[spawn_barrels_counter / 500].tag = BARREL_CIRCLE_VIEW;
```

```
            barrels[spawn_barrels_counter / 500].direction = BARREL_NORM;
            barrels[spawn_barrels_counter / 500].on = 1;
}

libusb_interrupt_transfer(keyboard, endpoint_address, data, sizeof(data), &transferred, 0);
 if (transferred == sizeof(data)) {
    key_pressed= KEY_NONE;
            if(!no_jump){
                        if(jump_cycle <16){
                                    if(mario.direction == 0){
                                                coords = move(mario.x, mario.y, 2, -2);
                                                mario.x = coords.x;
                                                mario.y = coords.y;

                                    }else{
                                                coords = move(mario.x, mario.y, -2, -2);
                                                mario.x = coords.x;
                                                mario.y = coords.y;
                                    }
                        } else if (jump_cycle < 20) {
                                    if(mario.direction == 0){
                                                coords = move(mario.x, mario.y, 2, -1);
                                                mario.x = coords.x;
                                                mario.y = coords.y;
                                    }else{
                                                coords = move(mario.x, mario.y, -2, -1);
                                                mario.x = coords.x;
                                                mario.y = coords.y;
                                    }
                        } else if (jump_cycle < 24) {
                                    if(mario.direction == 0){
                                                coords = move(mario.x, mario.y, 2, 1);
                                                mario.x = coords.x;
                                                mario.y = coords.y;
```

```
                        }else{
                                coords = move(mario.x, mario.y, -2, 1);
                                mario.x = coords.x;
                                mario.y = coords.y;
                        }
                } else if (jump_cycle < 34) {
                        if(mario.direction == 0){
                                coords = move(mario.x, mario.y, 2, 2);
                                mario.x = coords.x;
                                mario.y = coords.y;
                        }else{
                                coords = move(mario.x, mario.y, -2, 2);
                                mario.x = coords.x;
                                mario.y = coords.y;
                        }
                } else {
                        no_jump = 1;
                        mario.tag = 0;
                        jump_cycle=0;
                }
                jump_cycle++;
}else if (location == LOC_FALL && no_jump) {
        coords = move(mario.x, mario.y,0,1);
                // fprintf(stderr, "Falling");
                mario.x = coords.x;
                mario.y = coords.y;

} else if (location == LOC_SNAP_UP && no_jump) {
   coords = move(mario.x, mario.y,0,snap_up);
      // fprintf(stderr, "Snap Up");
      mario.x = coords.x;
      mario.y = coords.y;
} else {
      if(data[3]== 0 && no_jump){ //left pressed
```

```
    key_pressed = KEY_LEFT;
    if (location == LOC_LINE || location == LOC_FALL){
        coords = move(mario.x, mario.y,-1,0);
    }
    else if (location == LOC_SLOPE){
        if (direction == DIR_RIGHT){
            coords = move(mario.x, mario.y,-1,-1);
        }
        else{
            coords = move(mario.x, mario.y,-1,1);
        }
    }
    else{
        // printf("not in a place where you can go left");
    }
    mario.x = coords.x;
    mario.y = coords.y;
    mario.direction = 1;

    if((walk_cycle % 6) == 0){
        mario.tag =1;
    } else if ((walk_cycle % 6) == 2){
        mario.tag=0;
        }
    walk_cycle++;

    // fprintf(stderr, "Left");
}

else if(data[3]== 255 && no_jump){ //right pressed
    key_pressed = KEY_RIGHT;
    if (location == LOC_LINE  || location == LOC_FALL){
        coords = move(mario.x, mario.y,1,0);
    }
```

```
    else if (location == LOC_SLOPE){
        if (direction == DIR_RIGHT){
            coords = move(mario.x, mario.y,1,-1);
        }
        else{
            coords = move(mario.x, mario.y,1,1);

        }
    }
    else{
        // printf("not in a place where you can go right");
    }

    mario.x = coords.x;
    mario.y = coords.y;
    mario.direction = 0;
    if((walk_cycle % 6) == 0){
        mario.tag =1;
    } else if ((walk_cycle % 6) == 2){
        mario.tag=0;
    }
    walk_cycle++;

    // fprintf(stderr, "Right");
}


else if(data[4]== 0 && no_jump){ //up pressed
    key_pressed = KEY_UP;
    coords = move(mario.x, mario.y,0,-1);
    mario.x = coords.x;
    mario.y = coords.y;
    // fprintf(stderr, "Up");
    if((walk_cycle % 6) == 0 && mario.tag !=4){
```

```
        mario.tag =7;
    } else if ((walk_cycle % 6) == 2){
        mario.direction = 0;
        mario.tag=4;
    } else if ((walk_cycle % 6) == 5){
        mario.direction = 1;
        mario.tag=4;
    }
    walk_cycle++;
}

else if(data[4]== 255 && no_jump){ //Down pressed
    key_pressed = KEY_DOWN;
    coords = move(mario.x, mario.y,0,1);
    // fprintf(stderr, "Down");
    mario.x = coords.x;
    mario.y = coords.y;
    if((walk_cycle % 6) == 0 && mario.tag !=4){
        mario.tag =7;
    } else if ((walk_cycle % 6) == 2){
        mario.direction = 0;
        mario.tag=4;
    } else if ((walk_cycle % 6) == 5){
        mario.direction = 1;
        mario.tag=4;
    }
    walk_cycle++;
}


else if(data[5]== 47 && no_jump){ //A pressed
    // fprintf(stderr, "Jump");
    // if (location != LOC_FALL){
    //     coords = move(mario.x, mario.y,0,5);
```

```
                // }
                key_pressed = KEY_A;
                mario.tag = 2;
                no_jump = 0; //temp commented out
                        // printf("\nno jump in JUMP: %d \n", no_jump);
                fprintf(stderr, "A");
              }

            else if(data[5]== 79){ //B pressed
               key_pressed = KEY_B;
               fprintf(stderr, "B");
            }
            else if(data[5]== 111){ //A +B
               fprintf(stderr, "A+B");
            }
            else if(data[6]== 32){ //start
               key_pressed = KEY_START;
               fprintf(stderr, "start");
            }else if(data[6]== 16){ //select
               key_pressed = KEY_SELECT;
               fprintf(stderr, "select");
            }else if(data[6]== 48){ //start + select
               fprintf(stderr, "start + select");
            }
         }
 // int loc = get_location(b1.x, b1.y - 6); //account for the fact the barrel is shifted down by 6 since the bottom of the sprite is transparent


 usleep(20000);
if (spawn_barrels_counter < 500 * 5) {
   ++spawn_barrels_counter;
}
```

```
        // printf("0:%d 1:%d 2:%d 3:%d 4:%d ", barrels[0].on, barrels[1].on, barrels[2].on, barrels[3].on,
        barrels[4].on);
        send_to_HW(mario, barrels[0], barrels[1], barrels[2], barrels[3], barrels[4], MARIO);

        barrels[0] = update_barrel(barrels[0]);
        barrels[1] = update_barrel(barrels[1]);
        barrels[2] = update_barrel(barrels[2]);
        barrels[3] = update_barrel(barrels[3]);
        barrels[4] = update_barrel(barrels[4]);
         //printf("\n loc of b5 is: %d %d", b5.x, b5.y);
         //printf("\nloc of mario is: %d %d", mario.x, mario.y);
         int colide = collision(mario, barrels[4]);
        // printf("\n COLLUSION %d", colide);


         }


    }

}
```

Usbkeyboard.h

```
#ifndef _USBKEYBOARD_H
#define _USBKEYBOARD_H

#include <libusb-1.0/libusb.h>

#define USB_HID_KEYBOARD_PROTOCOL 0

/* 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
```

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;
}
```

Vga_ball.c

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>
//#include <stdint.h>

typedef struct {
  char tile_row0, tile_row1, tile_row2, tile_row3;
} vga_ball_tile_array_data_t;

typedef struct {
  unsigned int address;
} vga_ball_tile_array_address_t;

typedef struct {
    char data;
    short address;
    char blank;
} vga_ball_tile_map_t;

//arg structs for ioctl

typedef struct {
    short x_coor;
    short y_coor;
} vga_ball_xy_t;

typedef struct {
    short bx0;
    short by0;
} vga_ball_barrel0_xy_t;
```

```
typedef struct {
   short bx1;
   short by1;
} vga_ball_barrel1_xy_t;

typedef struct {
   short bx2;
   short by2;
} vga_ball_barrel2_xy_t;

typedef struct {
   short bx3;
   short by3;
} vga_ball_barrel3_xy_t;

typedef struct {
   short bx4;
   short by4;
} vga_ball_barrel4_xy_t;

typedef struct {
   unsigned int state;
} vga_ball_sprite_state_t;

typedef struct {
   unsigned int hhstate;
} vga_ball_hh_state_t;

typedef struct {
  vga_ball_tile_array_data_t data;
  vga_ball_tile_array_address_t address;
} vga_ball_arg_t;

typedef struct {
```

```
  vga_ball_tile_map_t tile_map;
} vga_ball_arg2_t;

typedef struct {
  vga_ball_xy_t coor;
} vga_ball_arg3_t;

typedef struct {
  vga_ball_sprite_state_t m_state;
} vga_ball_arg4_t;

typedef struct {
  vga_ball_barrel0_xy_t coor;
} vga_ball_arg5_t;

typedef struct {
  vga_ball_barrel1_xy_t coor;
} vga_ball_arg6_t;

typedef struct {
  vga_ball_barrel2_xy_t coor;
} vga_ball_arg7_t;

typedef struct {
  vga_ball_barrel3_xy_t coor;
} vga_ball_arg8_t;

typedef struct {
  vga_ball_barrel4_xy_t coor;
} vga_ball_arg9_t;

typedef struct {
  vga_ball_hh_state_t m_state;
} vga_ball_arg10_t;
```

```
#define VGA_BALL_MAGIC 'q'

/* ioctls and their arguments */

#define VGA_BALL_TILE_ARRAY _IOW(VGA_BALL_MAGIC, 1, vga_ball_arg_t *)
#define VGA_BALL_TILE_MAP  _IOW(VGA_BALL_MAGIC, 2, vga_ball_arg2_t *)
#define VGA_BALL_MARIO_COORDINATES _IOW(VGA_BALL_MAGIC, 3, vga_ball_arg3_t *)
#define VGA_BALL_SPRITE_STATE  _IOW(VGA_BALL_MAGIC, 4, vga_ball_arg4_t *)
#define VGA_BALL_BARREL0_COORDINATES  _IOW(VGA_BALL_MAGIC, 5, vga_ball_arg5_t *)
#define VGA_BALL_BARREL1_COORDINATES  _IOW(VGA_BALL_MAGIC, 6, vga_ball_arg6_t *)
#define VGA_BALL_BARREL2_COORDINATES  _IOW(VGA_BALL_MAGIC, 7, vga_ball_arg7_t *)
#define VGA_BALL_BARREL3_COORDINATES  _IOW(VGA_BALL_MAGIC, 8, vga_ball_arg8_t *)
#define VGA_BALL_BARREL4_COORDINATES  _IOW(VGA_BALL_MAGIC, 9, vga_ball_arg9_t *)

#define VGA_BALL_HH_STATE  _IOW(VGA_BALL_MAGIC, 10, vga_ball_arg10_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_ARRAY_DATA(x) (x)
#define BG_ARRAY_ADDRESS(x) ((x) + 4)
#define BG_TILE_MAP(x) ((x) + 8)
#define MARIO_COORDINATES(x) ((x) + 12)
#define SPRITE_STATE(x) ((x)+16)
#define BARREL0_COORDINATES(x) ((x)+20)
#define BARREL1_COORDINATES(x) ((x)+24)
#define BARREL2_COORDINATES(x) ((x)+28)
#define BARREL3_COORDINATES(x) ((x)+32)
#define BARREL4_COORDINATES(x) ((x)+36)
#define HEART_HAMMER_STATE(x) ((x)+40)



/*
 * 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_tile_array_data_t tile_data;
      vga_ball_tile_array_address_t tile_address;
      vga_ball_tile_map_t tile_map;
        vga_ball_xy_t mario_coordinates;
        vga_ball_sprite_state_t sprite_state;
        vga_ball_barrel0_xy_t barrel0;
        vga_ball_barrel1_xy_t barrel1;
        vga_ball_barrel2_xy_t barrel2;
        vga_ball_barrel3_xy_t barrel3;
```

```
        vga_ball_barrel4_xy_t barrel4;
        vga_ball_hh_state_t hh_state;
} dev;

/*
 * Write segments of a single digit
 * Assumes digit is in range and the device information has been set up
 */

static void load_barrel0_coordinates(vga_ball_barrel0_xy_t *bxy){
        unsigned int bxy_coor =0;
        //fprintf(stderr, "vga.c barrel 0 (%d,%d)\n", bxy->bx0, bxy->by0);
        bxy_coor = (((unsigned int) bxy->bx0)<<16)+((unsigned int) bxy->by0);
        iowrite32(bxy_coor, BARREL0_COORDINATES(dev.virtbase));
        dev.barrel0 = *bxy;
}
static void load_barrel1_coordinates(vga_ball_barrel1_xy_t *bxy){
        unsigned int bxy_coor =0;
        //fprintf(stderr, "vga.c barrel 1 (%d,%d)\n", bxy->bx1, bxy->by1);

        bxy_coor = (((unsigned int) bxy->bx1)<<16)+((unsigned int) bxy->by1);
        iowrite32(bxy_coor, BARREL1_COORDINATES(dev.virtbase));
        dev.barrel1 = *bxy;
}
static void load_barrel2_coordinates(vga_ball_barrel2_xy_t *bxy){
        unsigned int bxy_coor =0;
        //fprintf(stderr,"vga.c barrel 2 (%d,%d)\n", bxy->bx2, bxy->by2);
        bxy_coor = (((unsigned int) bxy->bx2)<<16)+((unsigned int) bxy->by2);
        iowrite32(bxy_coor, BARREL2_COORDINATES(dev.virtbase));
        dev.barrel2 = *bxy;
}

static void load_barrel3_coordinates(vga_ball_barrel3_xy_t *bxy){
        unsigned int bxy_coor =0;
```

```
        //fprintf(stderr,"vga.c barrel 2 (%d,%d)\n", bxy->bx2, bxy->by2);
        bxy_coor = (((unsigned int) bxy->bx3)<<16)+((unsigned int) bxy->by3);
        iowrite32(bxy_coor, BARREL3_COORDINATES(dev.virtbase));
        dev.barrel3 = *bxy;
}

static void load_barrel4_coordinates(vga_ball_barrel4_xy_t *bxy){
        unsigned int bxy_coor =0;
        //fprintf(stderr,"vga.c barrel 2 (%d,%d)\n", bxy->bx2, bxy->by2);
        bxy_coor = (((unsigned int) bxy->bx4)<<16)+((unsigned int) bxy->by4);
        iowrite32(bxy_coor, BARREL4_COORDINATES(dev.virtbase));
        dev.barrel4 = *bxy;
}

static void load_sprite_state(vga_ball_sprite_state_t *s){
        iowrite32(s->state, SPRITE_STATE(dev.virtbase));
        dev.sprite_state = *s;
}

static void load_hh_state(vga_ball_hh_state_t *h){
        iowrite32(h->hhstate, HEART_HAMMER_STATE(dev.virtbase));
        dev.hh_state = *h;
}

static void load_mario_coordinates(vga_ball_xy_t *xy){
        unsigned int xy_coor =0;
        xy_coor = (((unsigned int) xy->x_coor)<<16)+((unsigned int) xy->y_coor);
        iowrite32(xy_coor, MARIO_COORDINATES(dev.virtbase));
        dev.mario_coordinates = *xy;
}

static void load_tile_array(vga_ball_tile_array_data_t *data, vga_ball_tile_array_address_t *address)
{
        unsigned int row = 0;
```

```
	row = (((unsigned int) data->tile_row0) << 24) + (((unsigned int) data->tile_row1) << 16) +
(((unsigned int) data->tile_row2) << 8) + ((unsigned int) data->tile_row3);
	//fprintf(stderr, "ROW: %x, ADDRESS: %x\n", row, address->address);
	iowrite32(row, BG_ARRAY_DATA(dev.virtbase) );
	iowrite32(address->address, BG_ARRAY_ADDRESS(dev.virtbase) );
	dev.tile_data = *data;
	dev.tile_address = *address;
}

static void load_tile_map(vga_ball_tile_map_t *map)
{
	unsigned int map_int = 0;
	map_int = (((unsigned int) map->address) << 8) + ((unsigned int) map->data);
	//fprintf(stderr, "MAP: %x\n", map_int);
	iowrite32(map_int, BG_TILE_MAP(dev.virtbase) );
	dev.tile_map = *map;
}

/*
 * 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;
      vga_ball_arg2_t vla2;
	vga_ball_arg3_t vla3;
	vga_ball_arg4_t vla4;
	vga_ball_arg5_t vla5;
	vga_ball_arg6_t vla6;
	vga_ball_arg7_t vla7;
	vga_ball_arg8_t vla8;
	vga_ball_arg9_t vla9;
```

```
vga_ball_arg10_t vla10;

switch (cmd) {
case VGA_BALL_TILE_ARRAY:
        if (copy_from_user(&vla, (vga_ball_arg_t *) arg,
                           sizeof(vga_ball_arg_t)))
                return -EACCES;
        load_tile_array(&vla.data, &vla.address);
        break;

case VGA_BALL_TILE_MAP:
        if (copy_from_user(&vla2, (vga_ball_arg2_t *) arg,
                           sizeof(vga_ball_arg2_t)))
                return -EACCES;
        load_tile_map(&vla2.tile_map);
        break;

case VGA_BALL_MARIO_COORDINATES:
        if (copy_from_user(&vla3, (vga_ball_arg3_t *) arg,
                           sizeof(vga_ball_arg3_t *)))
                return -EACCES;
        load_mario_coordinates(&vla3.coor);
        break;

case VGA_BALL_SPRITE_STATE:
        if (copy_from_user(&vla4, (vga_ball_arg4_t *) arg,
                           sizeof(vga_ball_arg4_t *)))
                return -EACCES;
        load_sprite_state(&vla4.m_state);
        break;

case VGA_BALL_BARREL0_COORDINATES:
        if (copy_from_user(&vla5, (vga_ball_arg5_t *) arg,
                           sizeof(vga_ball_arg5_t *)))
```

```
                return -EACCES;
            load_barrel0_coordinates(&vla5.coor);
            break;

    case VGA_BALL_BARREL1_COORDINATES:
            if (copy_from_user(&vla6, (vga_ball_arg6_t *) arg,
                               sizeof(vga_ball_arg6_t *)))
                return -EACCES;
            load_barrel1_coordinates(&vla6.coor);
            break;

    case VGA_BALL_BARREL2_COORDINATES:
            if (copy_from_user(&vla7, (vga_ball_arg7_t *) arg,
                               sizeof(vga_ball_arg7_t *)))
                return -EACCES;
            load_barrel2_coordinates(&vla7.coor);
            break;
case VGA_BALL_BARREL3_COORDINATES:
            if (copy_from_user(&vla8, (vga_ball_arg8_t *) arg,
                               sizeof(vga_ball_arg8_t *)))
                return -EACCES;
            load_barrel3_coordinates(&vla8.coor);
            break;

    case VGA_BALL_BARREL4_COORDINATES:
            if (copy_from_user(&vla9, (vga_ball_arg9_t *) arg,
                               sizeof(vga_ball_arg9_t *)))
                return -EACCES;
            load_barrel4_coordinates(&vla9.coor);
            break;
case VGA_BALL_HH_STATE:
            if (copy_from_user(&vla10, (vga_ball_arg10_t *) arg,
                               sizeof(vga_ball_arg10_t *)))
                return -EACCES;
```

```
                load_hh_state(&vla10.m_state);
                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");
```

Makefile:

```
ifneq (${KERNELRELEASE},)

# KERNELRELEASE defined: we are being compiled as part of the Kernel
	obj-m := vga_ball.o

else

# We are being compiled as a module: use the Kernel build system

	KERNEL_SOURCE := /usr/src/linux-headers-$(shell uname -r)
	PWD := $(shell pwd)

default: module Mario_Move_Test reset

reset: reset.o
	gcc -Wall -o reset reset.o usbkeyboard.o -lusb-1.0

reset.o: reset.c usbkeyboard.h
	gcc -g -Wall -c reset.c

Mario_Move_Test: Mario_Move_Test.o usbkeyboard.o
	gcc -Wall -o Mario_Move_Test Mario_Move_Test.o usbkeyboard.o -lusb-1.0

Mario_Move_Test.o: Mario_Move_Test.c usbkeyboard.h
	gcc -g -Wall -c Mario_Move_Test.c

WhereIsMario: WhereIsMario.o usbkeyboard.o
	gcc -Wall -o WhereIsMario WhereIsMario.o usbkeyboard.o -lusb-1.0

WhereIsMario.o: WhereIsMario.c usbkeyboard.h
	gcc -g -Wall -c WhereIsMario.c
```

```
reset_and_barrel_spawn: reset_and_barrel_spawn.o usbkeyboard.o
	gcc -Wall -o reset_and_barrel_spawn reset_and_barrel_spawn.o usbkeyboard.o -lusb-1.0

reset_and_barrel_spawn.o: reset_and_barrel_spawn.c usbkeyboard.h
	gcc -g -Wall -c reset_and_barrel_spawn.c



barrel_no_mario_motion: barrel_no_mario_motion.o usbkeyboard.o
	gcc -Wall -o barrel_no_mario_motion barrel_no_mario_motion.o usbkeyboard.o -lusb-1.0

barrel_no_mario_motion.o: barrel_no_mario_motion.c usbkeyboard.h
	gcc -g -Wall -c barrel_no_mario_motion.c

Barrel_roll: Barrel_roll.o usbkeyboard.o
	gcc -Wall -o Barrel_roll Barrel_roll.o usbkeyboard.o -lusb-1.0

Barrel_roll.o: Barrel_roll.c usbkeyboard.h
	gcc -g -Wall -c Barrel_roll.c

barrel_roll_attempt2: barrel_roll_attempt2.o usbkeyboard.o
	gcc -Wall -o barrel_roll_attempt2 barrel_roll_attempt2.o usbkeyboard.o -lusb-1.0

barrel_roll_attempt2.o: barrel_roll_attempt2.c usbkeyboard.h
	gcc -g -Wall -c barrel_roll_attempt2.c



old_wheres_mario: old_wheres_mario.o usbkeyboard.o
	gcc -Wall -o old_wheres_mario old_wheres_mario.o usbkeyboard.o -lusb-1.0

old_wheres_mario.o: old_wheres_mario.c usbkeyboard.h
	gcc -g -Wall -c old_wheres_mario.c

combined: combined.o usbkeyboard.o
	gcc -Wall -o combined combined.o usbkeyboard.o -lusb-1.0
```

```
combined.o: combined.c usbkeyboard.h
	gcc -g -Wall -c combined.c



reset_mario: reset_mario.o usbkeyboard.o
	gcc -Wall -o reset_mario reset_mario.o usbkeyboard.o -lusb-1.0

reset_mario.o: reset_mario.c usbkeyboard.h
	gcc -g -Wall -c reset_mario.c



usbkeyboard.o: usbkeyboard.c usbkeyboard.h

module:
	${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} modules

clean:
	${MAKE} -C ${KERNEL_SOURCE} SUBDIRS=${PWD} clean
	${RM} hello

TARFILES = Makefile README vga_ball.h vga_ball.c
TARFILE = lab3-sw.tar.gz
.PHONY : tar
tar : $(TARFILE)

$(TARFILE) : $(TARFILES)
	tar zcfC $(TARFILE) .. $(TARFILES:%=lab3-sw/%)

endif
```

Vga_ball.sv

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


        /*if (hcount[10:4] == 7'd0 && vcount[9:3] == 7'd0) begin
                    {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
                end else if (hcount[10:4] == 7'd1 && vcount[9:3] == 7'd0) begin
                    {VGA_R, VGA_G, VGA_B} = {8'hff, 8'h0, 8'h0};
        end else if (hcount[10:4] == 7'd0 && vcount[9:3] == 7'd1) begin
           {VGA_R, VGA_G, VGA_B} = {8'h0, 8'hff, 8'h0};
        end else begin
           {VGA_R, VGA_G, VGA_B} = {background_r, background_g, background_b};
        end*/

/*if (graphics_loaded == 1'b0) begin loading in background not working

          if (tile_array_address[10:0] == 11'd1111) begin
             //graphics_loaded <= 1'b1;
             array_write_en <= 1'b0;
                    bg_map_address <= 10'b00101_00110; //tile 121 is at hcount tile 6 (00110), vcount tile 5
(00101)
           //bg_array_address <= (bg_map_q << 3) + vcount[3:1];
           bg_array_address <= (hcount[10:1] << 1); //testing just displaying something non black from tile
array
              end


              if (tile_map[17:8] == 10'd1023) begin
             map_write_en <= 1'b0;
             bg_map_address <= 10'b00101_00110; //tile 121 is at hcount tile 6 (00110), vcount tile 5
(00101)
```

```
        //bg_array_address <= (bg_map_q << 3) + vcount[3:1];
        bg_array_address <= (hcount[10:1] << 1); //testing just displaying something non black from tile
array
            end
    end else begin
        map_write_en <= 1'b0;
            array_write_en <= 1'b0;
        bg_map_address <= 10'b00101_00110; //tile 121 is at hcount tile 6 (00110), vcount tile 5 (00101)
        //bg_array_address <= (bg_map_q << 3) + vcount[3:1];
        bg_array_address <= (hcount[10:1] << 1); //testing just displaying something non black from tile
array

    end
    end*/



module vga_ball(input logic        clk,
                input logic               reset,
                    input logic [31:0]  writedata,
                    input logic           write,
                    input                 chipselect,
                    input logic [3:0]  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,

                    input L_READY,
                    input R_READY,

                    output logic [15:0] L_SAMPLE_DATA,
                    output logic [15:0] R_SAMPLE_DATA,
                    output logic L_VALID,
```

```
                    output logic R_VALID);


logic [10:0]       hcount;
logic [9:0]     vcount;

logic [5:0]    background_x;
logic [5:0]    background_y;

logic [10:0]       X;
logic [9:0]     Y;

logic [7:0]         background_r, background_g, background_b;

logic [23:0]    display_color;
logic [3:0]     display_color_code;

logic [5:0]     h_address;

//graphics logic
logic graphics_loaded;

logic [31:0]   tile_array_data;
logic [31:0]   tile_array_address;
logic [31:0]   tile_map;

logic [10:0]    bg_array_address;
logic [7:0]     bg_map_q;
logic [7:0][3:0]    bg_array_q;
logic [9:0]     bg_map_address;

        logic [3:0] color_code_in;

//sprites
```

```
    logic[31:0] sprite_state, barrel_0, barrel_1, barrel_2, barrel_3, barrel_4;

    //logic array_write_en;
    //logic REN;
    //logic map_write_en;

    //logic [639:0][3:0] sprite_buf;

    logic BG_REN;
    logic [9:0] load_bg_region;
    logic [7:0] tile_map_x_component; //make it 5 bits later
    logic [4:0] tile_map_y_component;

    logic [319:0][3:0] bg_buf;
    logic [7:0] load_bg_tile;

     //// MARIO ////

     logic [11:0] mario_address;
     logic [9:0] mario_address_x;
     logic [9:0] mario_address_y;
     logic [3:0] mario_output;
            //   soc_system_mario_unit
mario_unit(.address(mario_address[7:0]),.clk(clk),.clken(1),.reset_req(0),.readdata(mario_output));
     logic mario_en;

     logic [9:0] mario_x;
     logic [9:0] mario_y;
     logic [3:0] mario_n;
     logic mario_hammer_flip;


     ///////////////
```

```
//// HAMMER ////

logic [11:0] hammer_address;
logic [9:0] hammer_address_x;
logic [9:0] hammer_address_y;
logic [3:0] hammer_output;
logic hammer_en;

logic hammer_n;

///////////////

//// barrel 0 ///

logic [9:0] barrel_0_address;
logic [9:0] barrel_0_address_x;
logic [9:0] barrel_0_address_y;
logic [3:0] barrel_0_output;
logic barrel_0_en;

logic [15:0] barrel_0_x;
logic [15:0] barrel_0_y;
logic [1:0]  barrel_0_n;
logic barrel_0_flip;

//// barrel 1 ///

logic [9:0] barrel_1_address;
logic [9:0] barrel_1_address_x;
logic [9:0] barrel_1_address_y;
logic [3:0] barrel_1_output;
logic barrel_1_en;

logic [15:0] barrel_1_x;
```

```
logic [15:0] barrel_1_y;
logic [1:0]  barrel_1_n;
logic barrel_1_flip;

//// barrel 2 ///

logic [9:0] barrel_2_address;
logic [9:0] barrel_2_address_x;
logic [9:0] barrel_2_address_y;
logic [3:0] barrel_2_output;
logic barrel_2_en;

logic [15:0] barrel_2_x;
logic [15:0] barrel_2_y;
logic [1:0]  barrel_2_n;
logic barrel_2_flip;

//// barrel 3 ///

logic [9:0] barrel_3_address;
logic [9:0] barrel_3_address_x;
logic [9:0] barrel_3_address_y;
logic [3:0] barrel_3_output;
logic barrel_3_en;

logic [15:0] barrel_3_x;
logic [15:0] barrel_3_y;
logic [1:0]  barrel_3_n;
logic barrel_3_flip;

//// barrel 4 ///

logic [9:0] barrel_4_address;
logic [9:0] barrel_4_address_x;
```

```
    logic [9:0] barrel_4_address_y;
    logic [3:0] barrel_4_output;
    logic barrel_4_en;

    logic [15:0] barrel_4_x;
    logic [15:0] barrel_4_y;
    logic [1:0]  barrel_4_n;
    logic barrel_4_flip;

    ///////////////

    // mario_dir: // for reference based on tank implementation, should be updated for mario
    // 2'b0 --> up
    // 2'b1 --> down
    // 2'b2 --> left
    // 2'b3 --> right

    //// MARIO ////

    //assign mario_x = mario[31:16];
    //assign mario_y = mario[15:0];
    assign mario_n = sprite_state[4:1];
    assign mario_hammer_flip = sprite_state[0];

    assign mario_address_x = hcount[10:1] - mario_x;
    assign mario_address_y = vcount[9:0] - mario_y;

    // assign mario_address = mario_address_x[4:1] + (mario_address_y[4:1] << 4) + (mario_n << 8);
    // assign mario_en = hcount[10:1] >= mario_x && hcount[10:1] <= (mario_x + 16'd31) &&
vcount[9:0] >= mario_y && vcount[9:0] <= (mario_y + 10'd31);
    // assign mario_en = (mario_address_x[9:5] == 5'b0) && (mario_address_y[9:5] == 5'b0); //these bits
non zero when they're not 0 < x < 32

    ///////////////
```

```
    //// HAMMER ////

    assign hammer_address_x = hcount[10:1] - mario_x[9:0] + 10'd48;
    assign hammer_address_y = vcount[9:0] - mario_y[9:0] + 10'd32;
    assign hammer_n = ~mario_n[0]; //horizontal hammer lines up with even mario sprites, vertical odd

    // assign hammer_address = hammer_address_x[6:1] + (hammer_address_y[5:1] << 6) + (mario_n <<
11);
    assign hammer_en = ((4'd8 <= mario_n) && (4'd11 >= mario_n)) && (hammer_address_x[9:7] ==
7'b0) && (hammer_address_y[9:6] == 6'b0); //these bits non zero when they're not 0 < x < 32

    ///////////////

    //// BARREL 0 ////

    assign barrel_0_x = barrel_0[31:16];
    assign barrel_0_y = barrel_0[15:0];
    assign barrel_0_n = sprite_state[6:5];
    assign barrel_0_flip = sprite_state[7];

    assign barrel_0_address_x = hcount[10:1] - barrel_0_x[9:0];
    assign barrel_0_address_y = vcount[9:0] - barrel_0_y[9:0];

    //assign barrel_0_address = barrel_0_address_x[4:1] + (barrel_0_address_y[4:1] << 4) + (barrel_0_n
<< 8);
    assign barrel_0_en = sprite_state[23] && (barrel_0_address_x[9:5] == 5'b0) &&
(barrel_0_address_y[9:5] == 5'b0);

    //// BARREL 1 ////

    assign barrel_1_x = barrel_1[31:16];
    assign barrel_1_y = barrel_1[15:0];
    assign barrel_1_n = sprite_state[9:8];
```

```
    assign barrel_1_flip = sprite_state[10];

    assign barrel_1_address_x = hcount[10:1] - barrel_1_x[9:0];
    assign barrel_1_address_y = vcount[9:0] - barrel_1_y[9:0];

    //assign barrel_1_address = barrel_1_address_x[4:1] + (barrel_1_address_y[4:1] << 4) + (barrel_1_n
<< 8);
    assign barrel_1_en = sprite_state[24] && (barrel_1_address_x[9:5] == 5'b0) &&
(barrel_1_address_y[9:5] == 5'b0);

    //// BARREL 2 ////

    assign barrel_2_x = barrel_2[31:16];
    assign barrel_2_y = barrel_2[15:0];
    assign barrel_2_n = sprite_state[12:11];
    assign barrel_2_flip = sprite_state[13];

    assign barrel_2_address_x = hcount[10:1] - barrel_2_x[9:0];
    assign barrel_2_address_y = vcount[9:0] - barrel_2_y[9:0];

    //assign barrel_2_address = barrel_2_address_x[4:1] + (barrel_2_address_y[4:1] << 4) + (barrel_2_n
<< 8);
    assign barrel_2_en = sprite_state[25] && (barrel_2_address_x[9:5] == 5'b0) &&
(barrel_2_address_y[9:5] == 5'b0);

    //// BARREL 3 ////

    assign barrel_3_x = barrel_3[31:16];
    assign barrel_3_y = barrel_3[15:0];
    assign barrel_3_n = sprite_state[15:14];
    assign barrel_3_flip = sprite_state[16];

    assign barrel_3_address_x = hcount[10:1] - barrel_3_x[9:0];
    assign barrel_3_address_y = vcount[9:0] - barrel_3_y[9:0];
```

```
    //assign barrel_3_address = barrel_3_address_x[4:1] + (barrel_3_address_y[4:1] << 4) + (barrel_3_n
<< 8);
    assign barrel_3_en = sprite_state[26] && (barrel_3_address_x[9:5] == 5'b0) &&
(barrel_3_address_y[9:5] == 5'b0);

    //// BARREL 4 ////

    assign barrel_4_x = barrel_4[31:16];
    assign barrel_4_y = barrel_4[15:0];
    assign barrel_4_n = sprite_state[18:17];
    assign barrel_4_flip = sprite_state[19];

    assign barrel_4_address_x = hcount[10:1] - barrel_4_x[9:0];
    assign barrel_4_address_y = vcount[9:0] - barrel_4_y[9:0];

    //assign barrel_4_address = barrel_4_address_x[4:1] + (barrel_4_address_y[4:1] << 4) + (barrel_4_n
<< 8);
    assign barrel_4_en = sprite_state[27] && (barrel_4_address_x[9:5] == 5'b0) &&
(barrel_4_address_y[9:5] == 5'b0);

    ///////////////

        ////////////////////End mario sprite stuff//////////////////////////////////////////////////////////////
    assign background_x = hcount[10:5] - 6'd4; //offset to centre it
    assign background_y = vcount[9:4]; //bottom 4 LSBs define height of background tile

    assign load_bg_region = hcount[10:1] - 10'd640;
    assign BG_REN = ~(load_bg_region[0] | load_bg_region[1]); //NOR, load from bg rams when both 0
(so 1 every 4 pixels)
    //assign tile_map_x_component = background_x[4:0];
    assign tile_map_y_component = background_y[4:0];
    assign bg_map_address = (tile_map_y_component << 5) + tile_map_x_component[4:0];
    assign bg_array_address = (bg_map_q << 3) + vcount[3:1];
```

```
   assign load_bg_tile = load_bg_region[9:2]; //set so load_bg_tile goes from 0 to 39 in a region 160 pixels wide
   assign tile_map_x_component = hcount[10:3] - 8'd160 - 8'd4; //it increments by 1 every 4 pixels since ignoring bottom two lsbs of hcount[10:1]

   //assign bg_address = (vcount[3:1] << 3) + hcount[4:2];s

   // assign bg_map_address = 11'b11110010000 + vcount[3:1];



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

  always_ff @(posedge clk) begin
    if (reset) begin
          /*graphics_loaded <= 1'b0;
      array_write_en <= 1'b1;
      map_write_en <= 1'b1;
      bg_map_address <= tile_map[17:8]; //address is 10 bits for map
          bg_array_address <= tile_array_address[10:0]; //address is 11 bits for array*/
      bg_buf <= {320{4'h0}};
      // sprite_buf <= {640{4'hf}};
          //mario <= 32'b0000_0001_0001_0000___0000_0001_1011_0000;
      mario_x <= 10'b01_0001_0000;
      mario_y <= 10'b01_1011_0000;
          sprite_state <= 32'b0;
      barrel_0 <= 32'b0000_0000_1101_1000___0000_0000_1101_1000;
          barrel_1 <= 32'b0000_0000_0110_1100___0000_0000_0110_1100;
      barrel_2 <= 32'b0000_0000_0011_0110___0000_0000_0011_0110;
      barrel_3 <= 32'b0000_0000_0001_1011___0000_0000_0001_1011;
      barrel_4 <= 32'b0000_0000_0000_0101___0000_0000_0000_0101;
    end else if (chipselect && write) begin
      case (address)
```

```
            4'h0 : tile_array_data <= writedata;
            4'h1 : tile_array_address <= writedata;
            4'h2 : tile_map <= writedata;
            4'h3 : begin
                mario_x <= writedata[25:16]; //sprite
                mario_y <= writedata[9:0];
             end
        4'h4 : sprite_state <= writedata; //sprite
        4'h5 : barrel_0 <= writedata;
            4'h6 : barrel_1 <= writedata;
        4'h7 : barrel_2 <= writedata;
        4'h8 : barrel_3 <= writedata;
        4'h9 : barrel_4 <= writedata;

      endcase
    end

////////////////////////////////////SPRITE LOGIC START/////////////////////////////////////////////////////////////////

            //     if (hcount[10:1] >= mario_x && hcount[10:1] <= (mario_x + 16'd31) && vcount[9:0] >=
mario_y && vcount[9:0] <= (mario_y + 10'd31) ) begin

        //mario_en <= 1'b1;

            //case(p1tank_dir)
                        // mario_address <= mario_address_x[4:1] + (mario_address_y[4:1] << 4); //neglect
lsb so 1 pixel from rom is displayed over 2 by 2 pixels on the screen
             //2'b00 : mario_address <= hcount[10:1] - mario_x + (vcount[9:0] - mario_y) * 32;
        //2'b01 : mario_address <= hcount[10:1] - mario_x + (vcount[9:0] - mario_y) * 32 + 1024;
        //2'b10 : mario_address <= hcount[10:1] - mario_x + (vcount[9:0] - mario_y) * 32 + 2048;
        //2'b11 : mario_address <= hcount[10:1] - mario_x + (vcount[9:0] - mario_y) * 32 + 3072;
            //endcase

      //end else begin
```

```
        //mario_en <= 1'b0;

      //end
////////////////////////////////////SPRITE LOGIC END//////////////////////////////////////////////////////////////////////

    if ((hcount[10:1] > 655) && (hcount[10:1] < 784)) begin
        if (BG_REN) begin
        bg_buf[((hcount[10:1] - 10'd656) * 2) + 10'd16] <= bg_array_q[7]; // + 10'd32
        bg_buf[((hcount[10:1] - 10'd656) * 2) + 10'd16 + 10'd1] <= bg_array_q[6];
        bg_buf[((hcount[10:1] - 10'd656) * 2) + 10'd16 + 10'd2] <= bg_array_q[5];
        bg_buf[((hcount[10:1] - 10'd656) * 2) + 10'd16 + 10'd3] <= bg_array_q[4];
        bg_buf[((hcount[10:1] - 10'd656) * 2) + 10'd16 + 10'd4] <= bg_array_q[3];
        bg_buf[((hcount[10:1] - 10'd656) * 2) + 10'd16 + 10'd5] <= bg_array_q[2];
        bg_buf[((hcount[10:1] - 10'd656) * 2) + 10'd16 + 10'd6] <= bg_array_q[1];
        bg_buf[((hcount[10:1] - 10'd656) * 2) + 10'd16 + 10'd7] <= bg_array_q[0];
        //$display("BG buf index: %d", ((hcount[10:1] - 10'd640) << 1) + 10'd32);
        end
    end


    if (hcount[10:1] >= mario_x && hcount[10:1] <= (mario_x + 16'd31) && vcount[9:0] >= mario_y &&
vcount[9:0] <= (mario_y + 10'd31)) begin
        mario_en <= 1'b1;
    end else begin
        mario_en <= 1'b0;
    end

    if (mario_hammer_flip == 1) begin
        mario_address <= (mario_address_y[4:1] << 4) + (mario_n << 8) + (4'hf - mario_address_x[4:1]);
        hammer_address <= (hammer_address_y[5:1] << 6) + (mario_n << 11) + (6'b111111 -
hammer_address_x[6:1]);
    end else begin
        mario_address <= mario_address_x[4:1] + (mario_address_y[4:1] << 4) + (mario_n << 8);
        hammer_address <= hammer_address_x[6:1] + (hammer_address_y[5:1] << 6) + (mario_n << 11);
```

```
    end

    if (barrel_0_flip == 1) begin
        barrel_0_address <= (4'hf - barrel_0_address_x[4:1]) + (barrel_0_address_y[4:1] << 4) +
(barrel_0_n << 8);
    end else begin
        barrel_0_address <= barrel_0_address_x[4:1] + (barrel_0_address_y[4:1] << 4) + (barrel_0_n << 8);
    end

    if (barrel_1_flip == 1) begin
        barrel_1_address <= (4'hf - barrel_1_address_x[4:1]) + (barrel_1_address_y[4:1] << 4) +
(barrel_1_n << 8);
    end else begin
        barrel_1_address <= barrel_1_address_x[4:1] + (barrel_1_address_y[4:1] << 4) + (barrel_1_n << 8);
    end

    if (barrel_2_flip == 1) begin
        barrel_2_address <= (4'hf - barrel_2_address_x[4:1]) + (barrel_2_address_y[4:1] << 4) +
(barrel_2_n << 8);
    end else begin
        barrel_2_address <= barrel_2_address_x[4:1] + (barrel_2_address_y[4:1] << 4) + (barrel_2_n << 8);
    end

    if (barrel_3_flip == 1) begin
        barrel_3_address <= (4'hf - barrel_3_address_x[4:1]) + (barrel_3_address_y[4:1] << 4) +
(barrel_3_n << 8);
    end else begin
        barrel_3_address <= barrel_3_address_x[4:1] + (barrel_3_address_y[4:1] << 4) + (barrel_3_n << 8);
    end

    if (barrel_4_flip == 1) begin
        barrel_4_address <= (4'hf - barrel_4_address_x[4:1]) + (barrel_4_address_y[4:1] << 4) +
(barrel_4_n << 8);
    end else begin
```

```
        barrel_4_address <= barrel_4_address_x[4:1] + (barrel_4_address_y[4:1] << 4) + (barrel_4_n << 8);
    end


   end //always ff




//choose between background and sprite
always_comb begin
    if (mario_en & (!(mario_output[3] & mario_output[2] & mario_output[1] & mario_output[0]))) begin
            color_code_in = mario_output[3:0];
    end else if (hammer_en & (!(hammer_output[3] & hammer_output[2] & hammer_output[1] &
hammer_output[0]))) begin
            color_code_in = hammer_output[3:0];
    end else if (barrel_0_en & (!(barrel_0_output[3] & barrel_0_output[2] & barrel_0_output[1] &
barrel_0_output[0]))) begin
            color_code_in = barrel_0_output[3:0];
    end else if (barrel_1_en & (!(barrel_1_output[3] & barrel_1_output[2] & barrel_1_output[1] &
barrel_1_output[0]))) begin
            color_code_in = barrel_1_output[3:0];
    end else if (barrel_2_en & (!(barrel_2_output[3] & barrel_2_output[2] & barrel_2_output[1] &
barrel_2_output[0]))) begin
            color_code_in = barrel_2_output[3:0];
    end else if (barrel_3_en & (!(barrel_3_output[3] & barrel_3_output[2] & barrel_3_output[1] &
barrel_3_output[0]))) begin
            color_code_in = barrel_3_output[3:0];
    end else if (barrel_4_en & (!(barrel_4_output[3] & barrel_4_output[2] & barrel_4_output[1] &
barrel_4_output[0]))) begin
            color_code_in = barrel_4_output[3:0];
    end else begin
            color_code_in = bg_buf[hcount[10:2]][3:0];
    end
```

```
end

   //sprite memory

   mario mario_ram(.address(mario_address), .clock(clk), .data(4'b0), .rden(mario_en), .wren(1'b0),
.q(mario_output));
   hammer hammer_ram(.address(hammer_address), .clock(clk), .data(4'b0), .rden(hammer_en),
.wren(1'b0), .q(hammer_output));

   //barrels
   barrel barrel_0_ram(.address(barrel_0_address), .clock(clk), .data(4'b0), .rden(barrel_0_en),
.wren(1'b0), .q(barrel_0_output));
   barrel barrel_1_ram(.address(barrel_1_address), .clock(clk), .data(4'b0), .rden(barrel_1_en),
.wren(1'b0), .q(barrel_1_output));
   barrel barrel_2_ram(.address(barrel_2_address), .clock(clk), .data(4'b0), .rden(barrel_2_en),
.wren(1'b0), .q(barrel_2_output));
   barrel barrel_3_ram(.address(barrel_3_address), .clock(clk), .data(4'b0), .rden(barrel_3_en),
.wren(1'b0), .q(barrel_3_output));
   barrel barrel_4_ram(.address(barrel_4_address), .clock(clk), .data(4'b0), .rden(barrel_4_en),
.wren(1'b0), .q(barrel_4_output));

   //backgrounds
   tile_map bg_map_ram(.address(bg_map_address), .clock(clk), .data(tile_map[7:0]), .rden(BG_REN),
.wren(1'b0), .q(bg_map_q));
   tile_array bg_array_ram(.address(bg_array_address), .clock(clk), .data(tile_array_data),
.rden(BG_REN), .wren(1'b0), .q(bg_array_q));

   //H H_Highscore(.address(bg_address), .clock(clk), .q(display_color_code));
   sprite_color_pallette color_table(.color_code(color_code_in), .color(display_color));

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

```
      end
  end

  //   always_ff @(posedge clk) begin

  //  end //alwaysff 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
```

Sprite color palette:

```
module sprite_color_pallette(
        input logic [3:0]       color_code,
        output logic [23:0]     color
        );
        always_comb begin
                case(color_code)
                        4'h0 : color = 24'h000000;
                        4'h1 : color = 24'hffffff;
                        4'h2 : color = 24'hff2155;
                        4'h3 : color = 24'hfa0000;
                        4'h4 : color = 24'h00fbff;
                        4'h5 : color = 24'h970000;
                        4'h6 : color = 24'hff6800;
                        4'h7 : color = 24'hffb855;
                        4'h8 : color = 24'h0000f8;
                        4'h9 : color = 24'hb70000;
                        4'ha : color = 24'hfa4ef2;
    //colors b-f unused so far
                        4'hb : color = 24'h203090;
                        4'hc : color = 24'h2040A0;
                        4'hd : color = 24'h2060C0;
                        4'he : color = 24'h2090E0;
                        4'hf : color = 24'h64a460;
                        default : color = 24'h000000;
                endcase
        end
endmodule
```