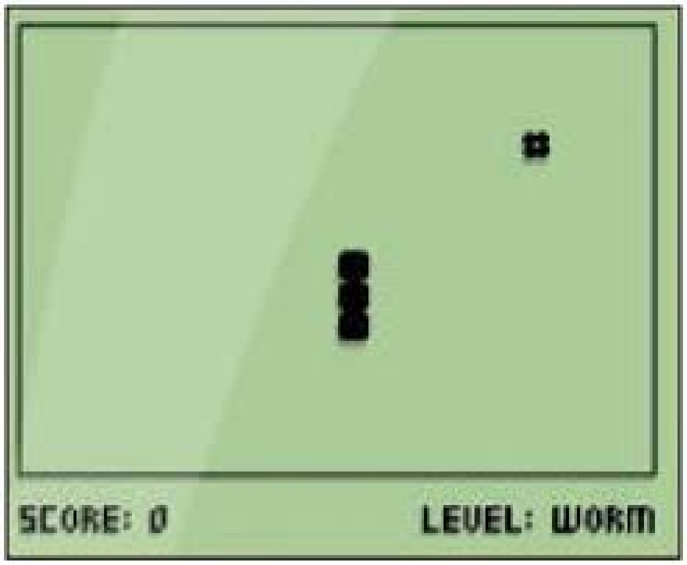
Snake Game on DE1-SOC Board
Final Report

CSEE 4840 Embedded System Design - Spring 2023



Project members:

Yue Rao(yr2425)

Tao Yan(ty2481)

Yuyang Wang(yw3912)

Jiawen Liu(jl6337)

Yuheng Zhang(yz4398)

Table of Contents

1. Introduction

The Snake game is a timeless and engaging video game that has enthralled players for generations. First introduced in the late 1970s, the game skyrocketed in popularity on early home computers and Nokia mobile phones in the 1990s. The game features a snake that navigates the screen, consuming food items while avoiding collisions with its ever-growing body and the boundaries of the play area. As the snake devours food, its length increases, progressively ramping up the challenge. For our embedded systems course, we aim to recreate the Snake game on the DE1-SoC board. Equipped with a plethora of interfaces and peripherals, the DE1-SoC board is an ideal platform for this project. Our goal is to design a game that closely mirrors the original Snake game experience. To this end, we have implemented several key elements of the classic 1990s Snake game, which include:

1. The Snake: The central character of the game, the snake, starts with a short length and grows longer with each food item it consumes. The player's objective is to grow the snake as long as possible while avoiding obstacles.

2. Food Items: Randomly appearing on the screen, food items are essential for the snake's growth and increasing the player's score. Each time the snake eats a food item, its length increases, and the score goes up.

3. Movement Controls: Typically, Snake utilizes four directional keys (up, down, left, and right) to control the snake's movement.

4. Walls: The playing area is limited by Walls, and the snake must not collide with these Walls.

5. Snake's Body: As the snake grows, the player must avoid running into its body. A collision with the snake's own body results in a game over.

6. Speed: The snake's movement speed increases as it grows, raising the difficulty level and demanding faster reactions from the player.

7. Scoring System: The player's score increases as the snake consumes food items, with each item typically worth a set number of points.

8. Background music: Adding background music to the Snake game on the DE1-SoC board is an interesting feature that enhances the gaming experience.

We use the Logitech F310 Wired Gamepad Controller to control the snake and a VGA monitor to display the game.

Figure 1: Logitech F310 Wired Gamepad Controller

Figure 2: vga monitor

2. System Block Diagram

Snake Game combines both hardware and software design. The software parts contain the device driver and game logic unit. The device driver provides an interface between hardware (audio generator, Logitech F310 Wired Gamepad Controller, and VGA monitor) data ports and logic control unit. The game logic includes object position display, collision detection, score tracking, recording, and food generation. The hardware section mainly contains the design of the VGA image display module, Logitech F310 Wired Gamepad Controller module, and audio generation module. Specifically, the Game logic unit sends each object's position and status data to hardware through device drivers. The hardware peripherals receive data from drivers, they decode the data and display the corresponding images or sound effects. On the other hand, the game logic unit receives the Logitech F310 Wired Gamepad Controller reading data from the hardware to update the player's position. The overall flowchart can be shown as follows:

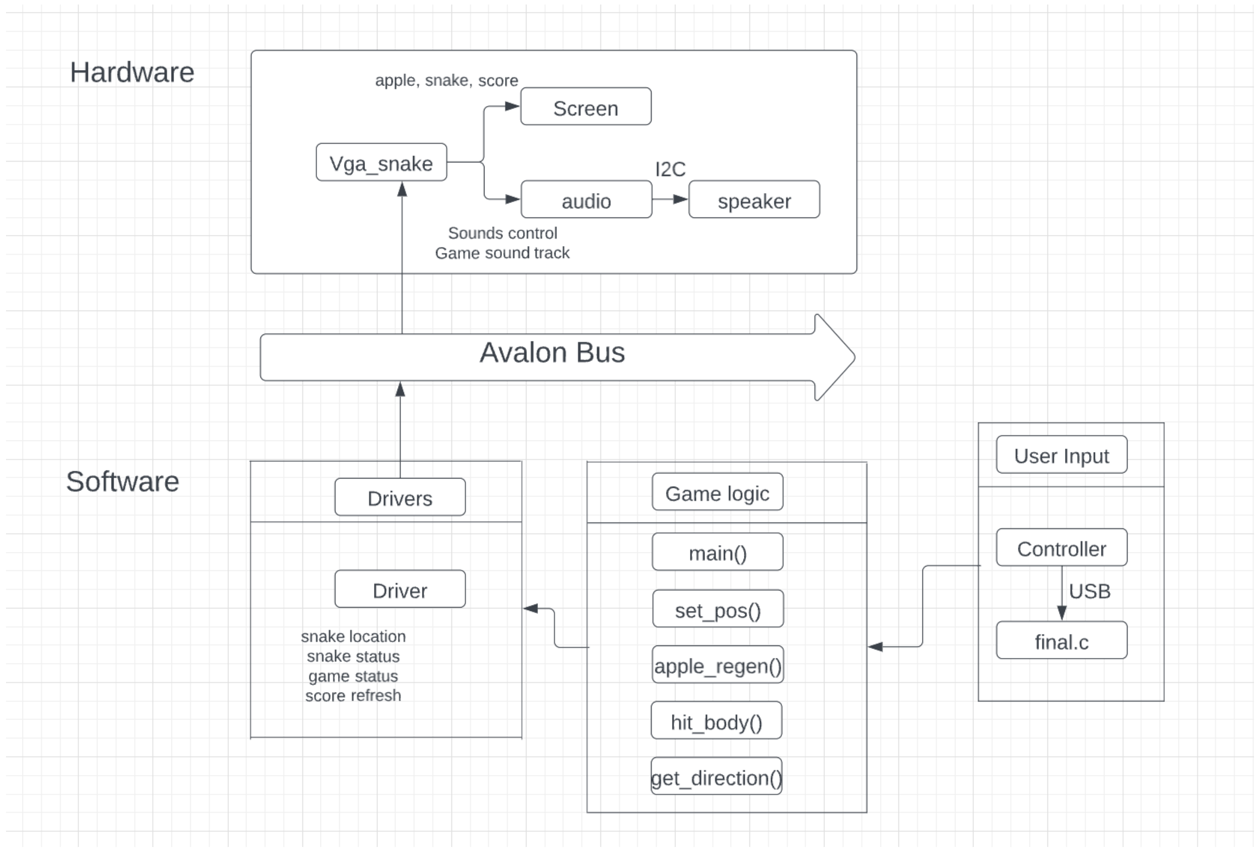


Figure 3: System Block Diagram

3. Software Logic

The game logic for the Snake game on the DE1-SoC board has been implemented using the C programming language, with communication between the software and hardware facilitated through device drivers. The software structure builds on the foundation provided in Lab3. The file `vga_ball.c` and `final.c` contains all the device driver functions required for communication with the hardware. The `vga_ball.h` header file defines the data types that are sent to the hardware. The primary game logic resides in the `final.c` file, which calls the device driver functions to transmit data to the hardware. In this section, we will delve into the game logic and explain its gameplay.

3.1 Game Basics

The software logic for the Snake game manages the overall game flow, encompassing game state initialization, user input handling, game object management, game state updates in response to various events, and game state rendering on the display. The game loop is the central component of the software logic, running continuously until the game is terminated. During each iteration of the loop, the game state is updated and rendered based on the elapsed time since the previous frame, ensuring smooth and consistent gameplay.

3.2 Game Objects

The primary game objects in the Snake game are the snake and the food item.

3.2.1 Food item (Apple)

The food item object is generated at random locations on the game grid. To ensure that it does not overlap with the snake or the boundaries, a function is implemented that checks the proposed food item location against the snake's coordinates and the grid boundaries. If the location is suitable, the food item is placed there. When the snake's head collides with the food item, the snake's length is increased by adding a new segment to the tail, the food item is eaten by the snake, and a new food item is generated at another random location.

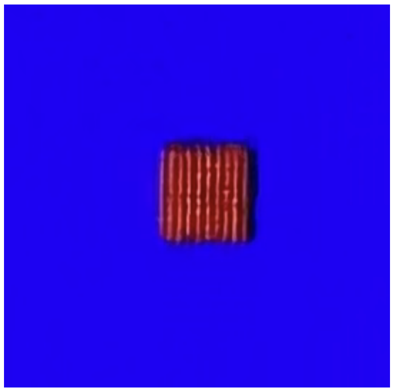


Figure 4: Apple

3.2.2 Snake

The snake object consists of a series of segments, each represented by a coordinate pair (x, y) on the game grid. We employed an array of specific data structures written by ourselves to store the snake's segments. The snake's head moves in the direction dictated by user input, with the body segments following the position of the previous segment. This can be achieved by updating the head's coordinates based on the current direction and then shifting the rest of the body segments accordingly. The snake's movement is refreshed at regular intervals, determined by the game's speed, which can be modulated using a game clock mechanism.

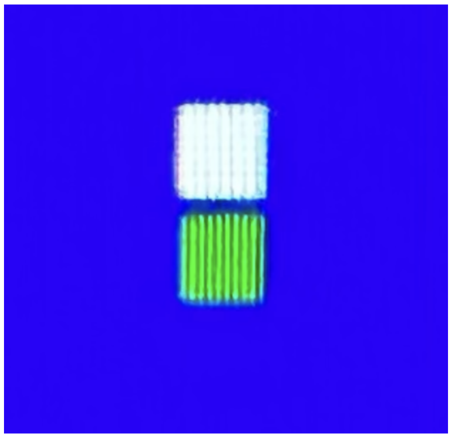


Figure 5: Snake

3.3 Collision Detection

The software logic checks for collisions between the snake's head and its body segments or the boundaries of the playing area. This is done by comparing the coordinates of the snake's head with the coordinates of each body segment and the grid boundaries. If a collision is detected, the game state is updated to reflect the game-over condition, and the player is prompted to restart or quit the game. This may involve displaying a game over screen, showing the score during the entire game session.

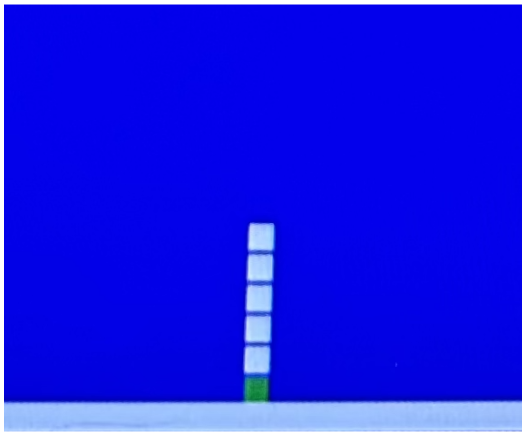


Figure 6: Snake Collision

3.4 Logitech F310 Wired Gamepad Controller reading handling

By mapping the controller's buttons to the corresponding directional inputs, the controller can be used to control the snake's movement in the game. The software logic includes a function and interrupt service routine to read the controller input via USB and update the snake's direction accordingly. Proper handling of the controller input involves debouncing, which ensures that only single input events are registered even if the user holds down or quickly presses a button multiple times. This contributes to smooth and responsive gameplay.

3.5 Game Speed Adjustment

Players can choose their own speed by first pressing the “start” button to pause the game, and then change the game speed by pressing the “X”, “Y”, “B”, and “A” buttons, in an increasing order of speed, respectively. Finally, press the “start” button again to resume the game with the speed newly set.

3.6 Background Music and Sound Effects

The software logic manages the playback of background music and sound effects during the game. This includes loading the audio files into memory, initializing and configuring the audio playback module, and controlling the playback in response to game events. For example, the background music loop seamlessly throughout the game, while sound effects should be played when the snake consumes food, collides with its body, or encounters the boundaries of the playing area.

3.7 Game State Management

Our software logic is designed to handle various game states, such as snake position, game speed, pause status, score, etc. Each game state is represented by a specific data structure/type, with transitions between states triggered by user input or game events. The main game loop should be designed to accommodate these state transitions, ensuring that the game state is updated and rendered correctly for the current state. For example, during gameplay, the snake's movement, collision detection, and score updates should be active; while in the pause state, these updates should be suspended.

4. Hardware-Software Interface

The software logic achieves its functionality through interaction with the DE1-SoC board's hardware. This involves writing driver or interface code, managing memory access, and configuring peripherals. After computing the game logic, the software updates the coordinates of the snake's head, the length of the snake's body, and the apple's coordinates, and transmits them to the registers of the DE1-SoC board via a kernel program. The hardware reads the data in the registers and accordingly updates the VGA display. Through appropriate hardware and software communication, we ensure the best gaming experience and efficient use of the DE1-SoC mainboard resources.

5. Hardware design

5.1 VGA display

To display the Snake game on the VGA monitor, it needs to generate the required VGA signals from the FPGA. This includes horizontal synchronization, vertical synchronization, and red, green, and blue color signals. These signals are generated by a VGA controller module implemented in the FPGA. The VGA controller reads the contents of the screen buffer and generates the corresponding RGB signals for the VGA display. Once the VGA signal generation is designed and tested individually, it will be integrated into the FPGA design.

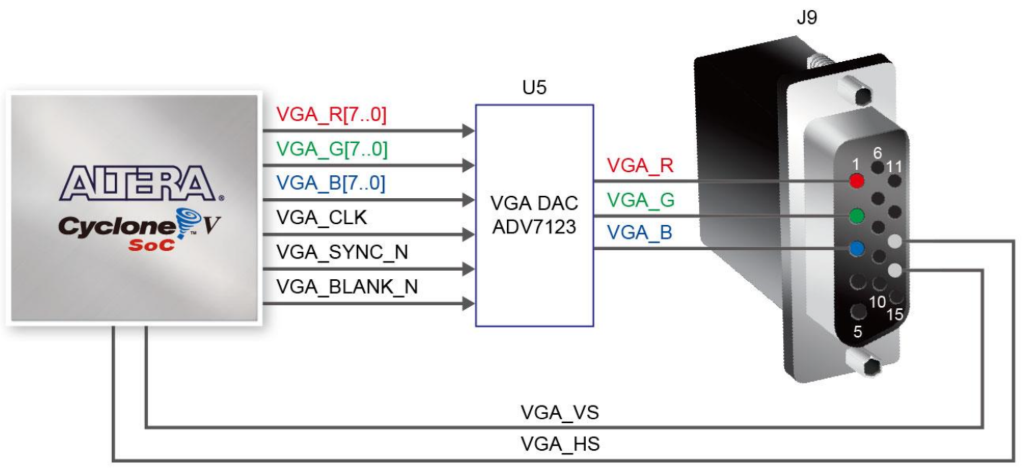


Figure 7: Connections between the FPGA and VGA

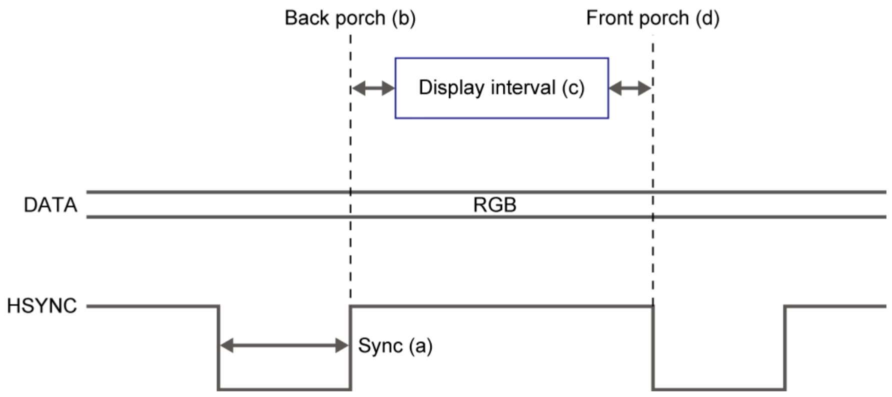
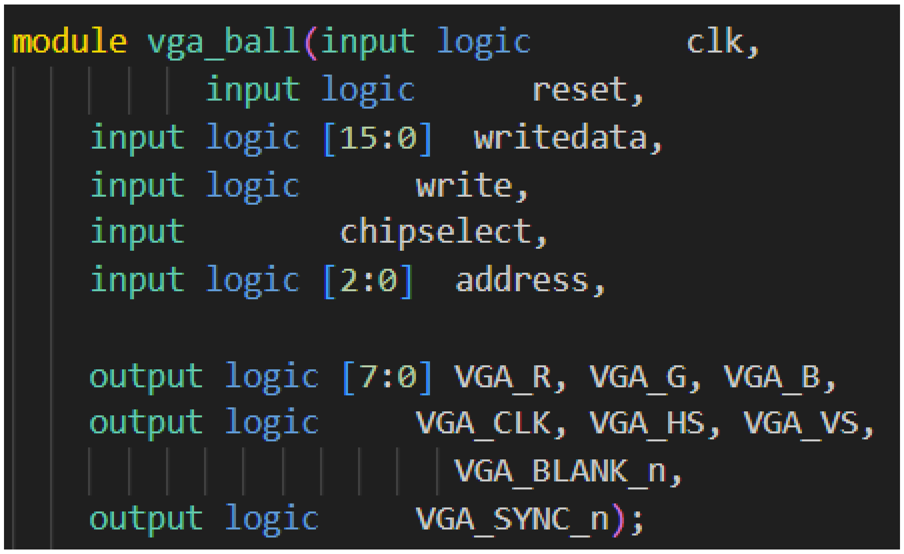


Figure 8: VGA waveform



```
module vga_ball(input logic        clk,
                input logic      reset,
        input logic [15:0]  writedata,
        input logic      write,
        input         chipselect,
        input logic [2:0]  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);
```

Figure 9: Module output & input

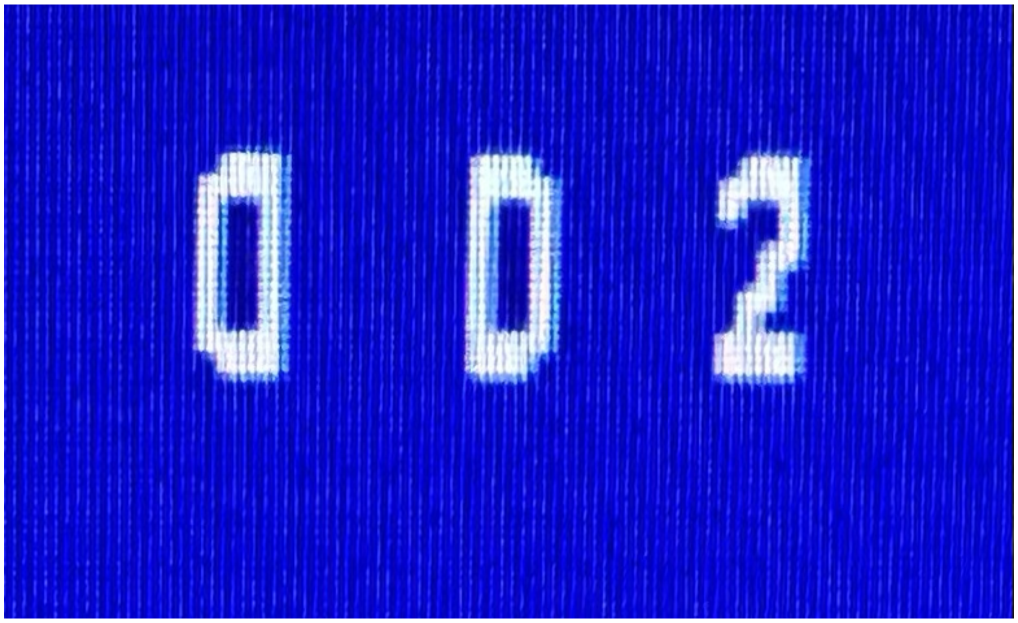


Figure 10: Number

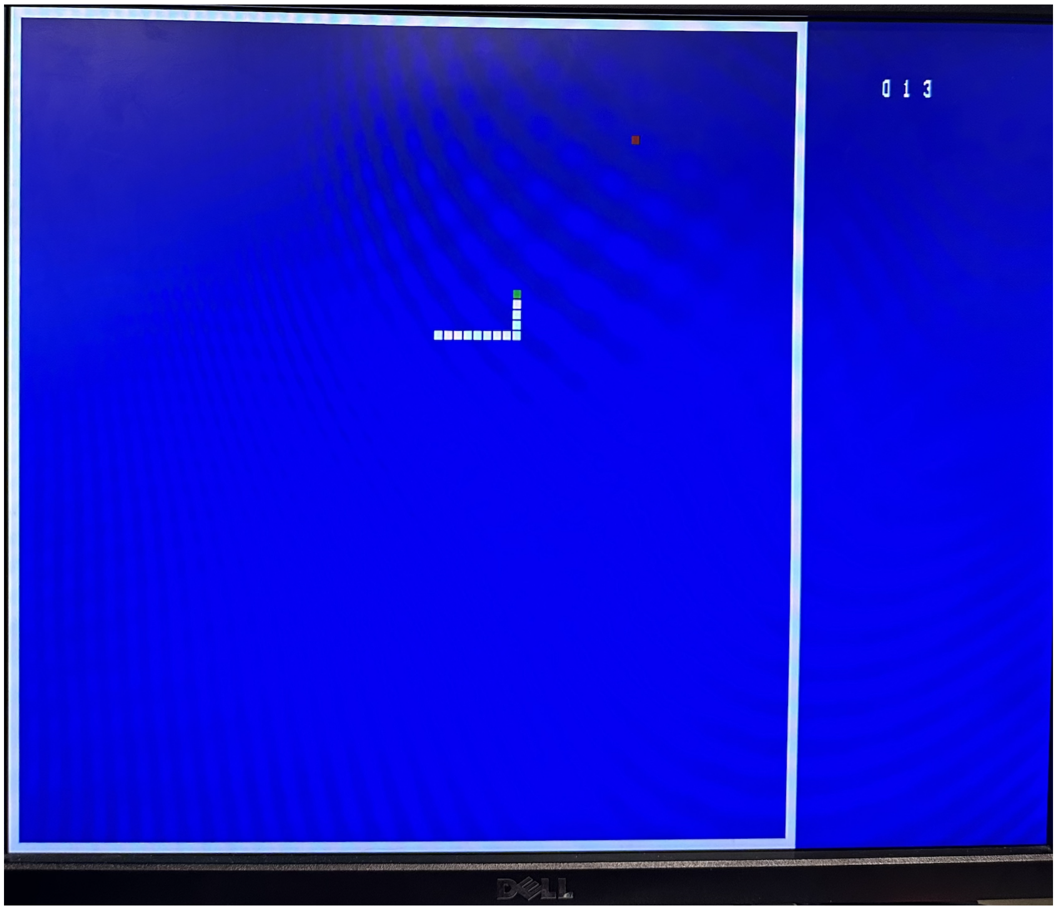


Figure 11: Game Screen

5.2 Logitech F310 Wired Gamepad Controller

We establish communication between the DE1-SoC board and the Logitech F310 controller by initializing the USB interface with the appropriate drivers and protocols. We design a hardware logic module that interprets the controller's inputs, such as button presses, and maps them to game actions. Implement input handling routines within the hardware logic to debounce button presses. This can be achieved by using timers or counters to measure the time elapsed since the last button press and registering the input event only when the elapsed time exceeds a predetermined threshold.

By following these steps, we successfully designed a hardware logic that processes the Logitech F310 controller's inputs and translates them into game actions, resulting in a smooth and responsive gaming experience.

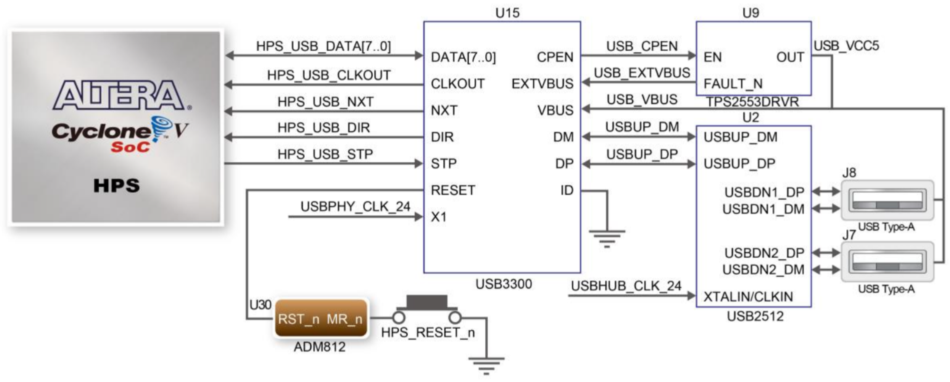


Figure 12: Connections between the HPS and USB OTG PHY

F310 Gamepad features		
Control	**XInput games**	**DirectInput games**
1. Left button/ trigger	Button is digital; trigger is analog	Button and trigger are digital and programmable*
2. Right button/ trigger	Button is digital; trigger is analog	Button and trigger are digital and programmable*
3. D-pad	8-way D-pad	8-way programmable D-pad*
4. Two analog mini-sticks	Clickable for button function	Programmable* (clickable for button function)
5. Mode button	Selects flight or sports mode. Flight mode: analog sticks control action and D-pad controls POV; Status light is off. Sports mode: D-pad controls action and analog sticks control POV; Status light is on.	
6. Mode/status light	Indicates sports mode (left analog stick and D-pad are swapped); controlled by Mode button	
7. Four action buttons	A, B, X, and Y	Programmable*
8. Start button	Start	Secondary programmable action button*
9. Logitech button	Guide button or keyboard's Home key	No function
10. Back button	Back	Secondary programmable action button*

Table 1: Gamepad features

5.3 Snake Game Audio Implementation

The DE1-SoC board offers 24-bit audio capabilities through the WM8731 audio CODEC. This chip comprises 2 DACs and 2 ADCs, and we are trying to configure it via the I2C bus while providing it with a clock at a rate of 12MHz. To synchronize the CODEC with the snake game's audio generation module, we need to introduce a FIFO buffer between the two. All these IPs are available in Qsys, specifically, Audio and Video Config (to configure the CODEC), Audio Clock for DE-series Boards (a PLL that generates 12MHz), and Audio (a FIFO buffer for audio samples).

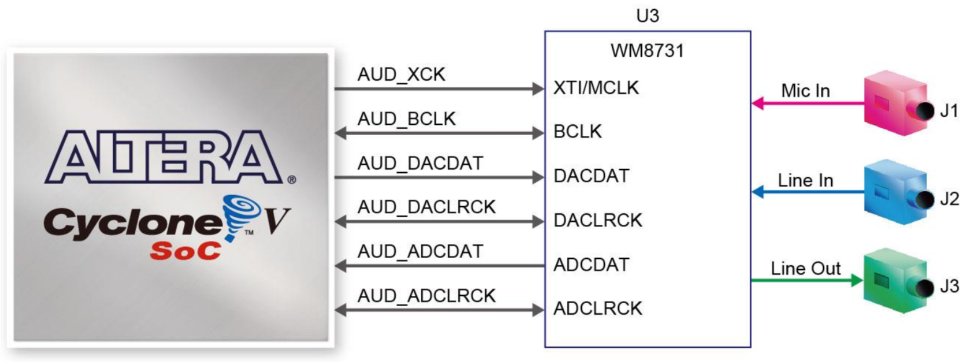


Figure 13: Connections between the FPGA and audio CODEC

5.4 Resource Budgets

RAM limit: 512 kB = 4096 kb

Category	Size(bits)
Snake body coordinates	200*32
Apple coordinates	1*32
Score number	3*4
Background music	120000*16
Eat apple music	2900*16
Hit wall/body music	2900*16
Wall coordinates	4*8
Total bits	2019276

Table 2: Resource Budgets

6. Challenges

6.1 Controller connection

During the process of connecting the game controller to the DE1-SOC Board, we encountered some issues. We were stuck on this problem for quite some time. Initially, we planned to use an Xbox controller to connect to the DE1-SOC Board. However, the Xbox controller is relatively new and some of its features like Bluetooth may hinder the connection between the controller and the board. So, we then decided to purchase a Logitech game controller. We chose this particular controller because it has been in production for a longer time, and some of its settings are more compatible with the DE1-SOC Board. Unfortunately, we also encountered connection issues with the Logitech game controller when attempting to connect it to the DE1-SOC Board. We checked our code, the DE1-SOC Board, and the SD card. Eventually, we discovered that the problem was related to the connection method of the controller. We needed to first connect the game controller to the DE1-SOC Board and then restart the DE1-SOC Board. At this point, the running file would detect a successful connection with the game controller. During the use of the controller, it must not be disconnected from the DE1-SOC Board; otherwise, the running file will not detect the controller again. In such a case, it would be necessary to restart the DE1-SOC Board and unplug and replug the controller's USB interface.

6.2 Game Audio

The inability to connect music to the FPGA board was primarily due to the challenge of locating the correct music file. we think several factors could have contributed to this issue:

File Format: The music file might not be in a format compatible with the FPGA board. Therefore, even if the file is present, the board might not recognize it, leading to a failure in establishing the connection.

Directory Structure: The music file might be located in a directory or folder that the FPGA board does not have access to. In such cases, the file effectively becomes "invisible" to the board.

7. Future Works

In this project, we have successfully developed a snake game utilizing a Logitech F310 Wired Gamepad Controller on the DE1-SoC board. However, there are several areas for potential improvements and further research.

7.1 Optimization of Game Performance:

Further optimization of the game's performance can be achieved by refining the hardware and software logic, including improvements in memory management, game state transitions.

7.2 Audio:

Currently, we have not identified a suitable sample capable of providing 16-bit audio via the WM8731 audio CODEC. Our analysis indicates that the issue could be attributed to the audio device's data format or communication protocol being incompatible with the DE1-SoC board. This incompatibility may result in incorrect audio data transmission. The audio device might use a different data format than what the WM8731 audio CODEC expects, causing the received audio data to be garbled and lead to distorted or noisy sound output. In future work, we plan to modify the audio device's output or adjust the WM8731's input settings to ensure proper compatibility.

7.3 Multiplayer Support: Implementing multiplayer support for the snake game on the DE1-SoC board will enable players to compete against each other. This will require the development of additional game mechanics, such as determining the winner based on a certain score threshold. Furthermore, it will be necessary to address the recognition of two Logitech gamepads by the DE1-SoC board simultaneously, as well as how different players can identify which snake is controlled by them.

By addressing these key points in future work, we can further improve the snake game, expand its features, and provide an even more engaging and enjoyable gaming experience.

8. Contribution

Each team member has made active contributions to the project design and collaborated closely in shaping the System Architecture.

Team members Yuyang Wang, Jiawen Liu, and Yue Rao were mainly responsible for developing the software components of the project, including the game logic and device driver.

Meanwhile, members Yuheng Zhang and Tao Yan are working on crafting the hardware code with the DE1-SoC board.

Furthermore, all members came together to produce a comprehensive project documentation, detailing its execution from conception to completion.

9.Software Code

9.1 final.c :

```
#include <time.h>
#include <libusb-1.0/libusb.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdint.h>
#include <unistd.h>
#include <pthread.h>
#include <string.h>
#include "vga_ball.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>

#define UP 0
#define RIGHT 2
#define DOWN 4
#define LEFT 6
#define ROWS 79
#define COLS 79

struct libusb_device_handle *f310;
uint8_t endpoint_address;

struct coordinate *bodies;
struct coordinate apple;
int len;
struct f310_packet packet;
int transferred;
int vga_ball_fd;
static const char filename[] = "/dev/vga_ball";
vga_ball_pos position;
int game_pause;
int level;
int can_change;
struct coordinate *buf;

struct f310_packet {
 uint8_t modifiers;
 uint8_t reserved;
 uint8_t keycode[6];
};

struct head {
   int x;
   int y;
   int direction;
} head;

struct coordinate {
   int x;
   int y;
};

struct libusb_device_handle *open_f310(uint8_t *endpoint_address) {
   libusb_device **devs;
   struct libusb_device_handle *f310 = NULL;
   struct libusb_device_descriptor desc;
   ssize_t num_devs, d;
   uint8_t i, k;

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

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

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

       if (desc.idVendor == 0x046d && desc.idProduct == 0xc216) {
           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 == 0x03 && inter->bInterfaceProtocol ==
0x00) {
                        int r;
                        if ((r = libusb_open(dev, &f310)) != 0) {
                            fprintf(stderr, "Error: libusb_open failed: %d\n", r);
                            exit(1);
                        }
                        if (libusb_kernel_driver_active(f310, i))
libusb_detach_kernel_driver(f310, i);
                        libusb_set_auto_detach_kernel_driver(f310, 1);
                        if ((r = libusb_claim_interface(f310, 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 f310;
}

void set_pos(const vga_ball_pos *c) {
   vga_ball_arg_t vla;
   vla.position = *c;
   if (ioctl(vga_ball_fd, VGA_BALL_WRITE_BACKGROUND, &vla)) {
       perror("ioctl(VGA_BALL_SET_BACKGROUND) failed");
       return;
   }
}
```

```
void draw() {
   int i, j;

   for (i = 0; i <= ROWS; i++) {
       for (j = 0; j <= COLS; j++) {
           if (i == 0 || j == 0 || i == ROWS || j == COLS) {
               printf("#");
           } else if (i == head.x && j == head.y) {
               printf("O");
           } else if (i == apple.x && j == apple.y) {
               printf("A");
           } else {
               printf(" ");
           }
       }
       printf("\n");
   }
}

int hit_body() {
   int i;

   for (i = 0; i < len; i++) if (head.x == bodies[i].x && head.y == bodies[i].y)
return 1;

   return 0;
}

/*
int consume() {
   if (head.x == apple.x && head.y == apple.y) return 1;

   return 0;
}
*/

void regen_apple() {
   int i, x, y;

x_gen:
   x = rand() % COLS;
   if (x == head.x || x == 0) goto x_gen;
   for (i = 0; i < len; i++) if (x == bodies[i].x) goto x_gen;
y_gen:
   y = rand() % ROWS;
   if (y == head.y || y == 0 ) goto y_gen;
   for (i = 0; i < len; i++) if (y == bodies[i].y) goto y_gen;

   apple.x = x;
   apple.y = y;

   position.ax = x;
   position.ay = y;

   return;
}

void *get_direction(void *arg) {
   int threadID = *((int *)arg);
   // printf("Thread ID: %d\n", threadID);
   // Perform thread-specific operations here
   for (;;) {
       libusb_interrupt_transfer(f310, endpoint_address, (unsigned char *) &packet,
sizeof(packet), &transferred, 0);
       if (transferred == sizeof(packet)) {
           if (can_change) {
                   if (packet.keycode[2] == 0 && head.direction != DOWN) {
                       head.direction = UP;
                   } else if (packet.keycode[2] == 2 && head.direction != LEFT) {
                       head.direction = RIGHT;
                   } else if (packet.keycode[2] == 4 && head.direction != UP) {
                       head.direction = DOWN;
                   } else if (packet.keycode[2] == 6 && head.direction != RIGHT) {
                       head.direction = LEFT;
                   }
               }
               can_change = 0;

               if (packet.keycode[3] == 32) {
                   game_pause = !game_pause;
               }

               if (game_pause) {
                   if (packet.keycode[2] == 24) {
                       level = 1;
                   } else if (packet.keycode[2] == 136) {
                       level = 2;
                   } else if (packet.keycode[2] == 72) {
                       level = 3;
                   } else if (packet.keycode[2] == 40) {
                       level = 4;
                   }
               }
           }
   }
   pthread_exit(NULL);
}

int main() {
   int i, input, threadID;
   pthread_t thread;

   srand(time(NULL));

   if ((f310 = open_f310(&endpoint_address)) == NULL) {
       fprintf(stderr, "Did not find f310\n");
       exit(1);
   }

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

   bodies = malloc(200 * sizeof(struct coordinate));
   buf = malloc(200 * sizeof(struct coordinate));

   head.x = COLS / 2;
   head.y = ROWS / 2;
   head.direction = RIGHT;

   position.x = head.x;
   position.y = head.y;

   level = 1;
   game_pause = 1;
```

```
   can_change = 0;

   pthread_create(&thread, NULL, get_direction, &threadID);

start:
   game_pause = 1;
   can_change = 1;

   head.x = COLS / 2;
   head.y = ROWS / 2;
   head.direction = RIGHT;
   position.x = head.x;
   position.y = head.y;

   regen_apple();

   len = 1;
   position.length = len + 1;

   bodies[0].x = head.x - 1;
   bodies[0].y = head.y;

logic:
   for (;;) {
       if (game_pause) goto logic;

       memcpy(buf, bodies, len * sizeof(struct coordinate));
       memcpy(bodies + 1, buf, len * sizeof(struct coordinate));
       bodies[0].x = head.x;
       bodies[0].y = head.y;

       switch (head.direction) {
           case UP:
               head.y--;
               position.y--;
               if (head.y <= 0 || hit_body()) {
                   goto start;
               }
               break;

           case DOWN:
               head.y++;
```

```
            position.y++;
            if (head.y >= ROWS || hit_body()) {
                goto start;
            }
            break;

        case LEFT:
            head.x--;
            position.x--;
            if (head.x <= 0 || hit_body()) {
                goto start;
            }
            break;

        case RIGHT:
            head.x++;
            position.x++;
            if (head.x >= COLS || hit_body()) {
                goto start;
            }
            break;
    }

    if (head.x == apple.x && head.y == apple.y) {
        len++;
        position.length++;
        regen_apple();
    }

    set_pos(&position);

    can_change = 1;

    switch (level) {
        case 1:
            usleep(100000);
            break;

        case 2:
            usleep(65000);
            break;
```

```
           case 3:
               usleep(25000);
               break;

           case 4:
               usleep(10000);
               break;
       }
   }
   pthread_join(thread, NULL);
   free(bodies);
   free(buf);
}
```

9.2 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 POS_X(x)  (x)
#define POS_Y(x)  ((x)+2)
#define LENGTH(x)  ((x)+4)
#define APPLE_X(x)  ((x)+6)
#define APPLE_Y(x)  ((x)+8)

/*
* 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_pos position;
} dev;

/*
* Write segments of a single digit
* Assumes digit is in range and the device information has been set up
*/
static void write_background(vga_ball_pos *position)
{
   iowrite16(position->x, POS_X(dev.virtbase) );
   iowrite16(position->y, POS_Y(dev.virtbase) );
   iowrite16(position->length, LENGTH(dev.virtbase) );
   iowrite16(position->ax, APPLE_X(dev.virtbase) );
       iowrite16(position->ay, APPLE_Y(dev.virtbase) );
   dev.position = *position;
}

/*
* Handle ioctl() calls from userspace:
* Read or write the segments on single digits.
* Note extensive error checking of arguments
*/
static long vga_ball_ioctl(struct file *f, unsigned int cmd, unsigned long arg)
{
   vga_ball_arg_t vla;

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

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

   default:
       return -EINVAL;
   }

   return 0;
}

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

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

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

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

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

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

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

   /* Set an initial color */
```

```
       write_background(&beige);

   return 0;

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

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

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

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

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

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

module_init(vga_ball_init);
module_exit(vga_ball_exit);

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

vga_ball.h

```
#ifndef _VGA_BALL_H
#define _VGA_BALL_H

#include <linux/ioctl.h>

typedef struct {
 unsigned int x, y, length,ax,ay;
} vga_ball_pos;

typedef struct {
 vga_ball_pos position;
} vga_ball_arg_t;

#define VGA_BALL_MAGIC 'q'

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

#endif
```

10.Hardware Code

```
module vga_ball(input logic        clk,
               input logic             reset,
                  input logic [15:0]  writedata,
                  input logic          write,
                  input                chipselect,
                  input logic [2:0]  address,

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

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

  logic [20:0]   dis;
  logic [15:0]     background_r, background_g, background_b;
  logic [15:0]     pos_x[0:200];
  logic [15:0]     pos_y[0:200];
  logic [15:0]     apple_x;
  logic [15:0]     apple_y;
  logic [15:0]     head_x,head_y;
  logic [15:0]     snake_length=2;
  logic [9:0]       sim_time = 0;
  logic            extend=0;
  logic [0:15]    number_zero[0:15], number_one[0:15],
number_two[0:15],number_three[0:15],number_four[0:15];
  logic [0:15]    number_five[0:15], number_six[0:15],
number_seven[0:15],number_eight[0:15],number_nine[0:15];
  logic [3:0]     score_a,score_b,score_c;




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

  always_ff @(posedge clk)
    if (reset) begin
         background_r <= 8'h0;
         background_g <= 8'h0;
         background_b <= 8'h80;
      number_zero[0][0:15] <=                    16'b0000000000000000;
      number_zero[1][0:15] <=                    16'b0000000000000000;
```

```
number_zero[2][0:15] <=             16'b0000000000000000;
number_zero[3][0:15] <=             16'b0000011111100000;
number_zero[4][0:15] <=             16'b0000111111110000;
number_zero[5][0:15] <=             16'b0000110000110000;
number_zero[6][0:15] <=             16'b0000110000110000;
number_zero[7][0:15] <=             16'b0000110000110000;
number_zero[8][0:15] <=             16'b0000110000110000;
number_zero[9][0:15] <=             16'b0000110000110000;
number_zero[10][0:15] <=       16'b0000110000110000;
number_zero[11][0:15] <=       16'b0000111111110000;
number_zero[12][0:15] <=       16'b0000011111100000;
number_zero[13][0:15] <=       16'b0000000000000000;
number_zero[14][0:15] <=       16'b0000000000000000;
number_zero[15][0:15] <=       16'b0000000000000000;
//number1
number_one[0][0:15] <=         16'b0000000000000000;
number_one[1][0:15] <=         16'b0000000000000000;
number_one[2][0:15] <=         16'b0000000000000000;
number_one[3][0:15] <=         16'b0000000110000000;
number_one[4][0:15] <=         16'b0000001110000000;
number_one[5][0:15] <=         16'b0000011110000000;
number_one[6][0:15] <=         16'b0000000110000000;
number_one[7][0:15] <=         16'b0000000110000000;
number_one[8][0:15] <=         16'b0000000110000000;
number_one[9][0:15] <=         16'b0000000110000000;
number_one[10][0:15] <=             16'b0000000110000000;
number_one[11][0:15] <=             16'b0000011111100000;
number_one[12][0:15] <=             16'b0000011111100000;
number_one[13][0:15] <=             16'b0000000000000000;
number_one[14][0:15] <=             16'b0000000000000000;
number_one[15][0:15] <=             16'b0000000000000000;
//number2
number_two[0][0:15] <=         16'b0000000000000000;
number_two[1][0:15] <=         16'b0000000000000000;
number_two[2][0:15] <=         16'b0000000000000000;
number_two[3][0:15] <=         16'b0000011111100000;
number_two[4][0:15] <=         16'b0000111111110000;
number_two[5][0:15] <=         16'b0000110000110000;
number_two[6][0:15] <=         16'b0000000001110000;
number_two[7][0:15] <=         16'b0000000011100000;
number_two[8][0:15] <=         16'b0000000111000000;
number_two[9][0:15] <=         16'b0000001110000000;
number_two[10][0:15] <=             16'b0000011100000000;
number_two[11][0:15] <=             16'b0000111111110000;
```

```
number_two[12][0:15] <=             16'b0000111111110000;
number_two[13][0:15] <=             16'b0000000000000000;
number_two[14][0:15] <=             16'b0000000000000000;
number_two[15][0:15] <=             16'b0000000000000000;
//number3
number_three[0][0:15] <=       16'b0000000000000000;
number_three[1][0:15] <=       16'b0000000000000000;
number_three[2][0:15] <=       16'b0000000000000000;
number_three[3][0:15] <=       16'b0000011111100000;
number_three[4][0:15] <=       16'b0000111111110000;
number_three[5][0:15] <=       16'b0000110000110000;
number_three[6][0:15] <=       16'b0000000000110000;
number_three[7][0:15] <=       16'b0000000111100000;
number_three[8][0:15] <=       16'b0000000111100000;
number_three[9][0:15] <=       16'b0000000000110000;
number_three[10][0:15] <=      16'b0000110000110000;
number_three[11][0:15] <=      16'b0000111111110000;
number_three[12][0:15] <=      16'b0000011111100000;
number_three[13][0:15] <=      16'b0000000000000000;
number_three[14][0:15] <=      16'b0000000000000000;
number_three[15][0:15] <=      16'b0000000000000000;
//number4
number_four[0][0:15] <=        16'b0000000000000000;
number_four[1][0:15] <=        16'b0000000000000000;
number_four[2][0:15] <=        16'b0000000000000000;
number_four[3][0:15] <=        16'b0000000011100000;
number_four[4][0:15] <=        16'b0000000111100000;
number_four[5][0:15] <=        16'b0000001111100000;
number_four[6][0:15] <=        16'b0000011101100000;
number_four[7][0:15] <=        16'b0000111001100000;
number_four[8][0:15] <=        16'b0000110001100000;
number_four[9][0:15] <=        16'b0000111111110000;
number_four[10][0:15] <=       16'b0000111111110000;
number_four[11][0:15] <=       16'b0000000001100000;
number_four[12][0:15] <=       16'b0000000001100000;
number_four[13][0:15] <=       16'b0000000000000000;
number_four[14][0:15] <=       16'b0000000000000000;
number_four[15][0:15] <=       16'b0000000000000000;
//number5
number_five[0][0:15] <=        16'b0000000000000000;
number_five[1][0:15] <=        16'b0000000000000000;
number_five[2][0:15] <=        16'b0000000000000000;
number_five[3][0:15] <=        16'b0000111111110000;
number_five[4][0:15] <=        16'b0000111111110000;
```

```
number_five[5][0:15] <=       16'b0000110000000000;
number_five[6][0:15] <=       16'b0000110000000000;
number_five[7][0:15] <=       16'b0000111111100000;
number_five[8][0:15] <=       16'b0000111111110000;
number_five[9][0:15] <=       16'b0000000000110000;
number_five[10][0:15] <=      16'b0000000000110000;
number_five[11][0:15] <=      16'b0000111111110000;
number_five[12][0:15] <=      16'b0000111111100000;
number_five[13][0:15] <=      16'b0000000000000000;
number_five[14][0:15] <=      16'b0000000000000000;
number_five[15][0:15] <=      16'b0000000000000000;
//number6
number_six[0][0:15] <=        16'b0000000000000000;
number_six[1][0:15] <=        16'b0000000000000000;
number_six[2][0:15] <=        16'b0000000000000000;
number_six[3][0:15] <=        16'b0000011111110000;
number_six[4][0:15] <=        16'b0000111111110000;
number_six[5][0:15] <=        16'b0000110000000000;
number_six[6][0:15] <=        16'b0000110000000000;
number_six[7][0:15] <=        16'b0000111111100000;
number_six[8][0:15] <=        16'b0000111111110000;
number_six[9][0:15] <=        16'b0000110000110000;
number_six[10][0:15] <=       16'b0000110000110000;
number_six[11][0:15] <=       16'b0000111111110000;
number_six[12][0:15] <=       16'b0000011111100000;
number_six[13][0:15] <=       16'b0000000000000000;
number_six[14][0:15] <=       16'b0000000000000000;
number_six[15][0:15] <=       16'b0000000000000000;
//number7
number_seven[0][0:15] <=      16'b0000000000000000;
number_seven[1][0:15] <=      16'b0000000000000000;
number_seven[2][0:15] <=      16'b0000000000000000;
number_seven[3][0:15] <=      16'b0000111111110000;
number_seven[4][0:15] <=      16'b0000111111110000;
number_seven[5][0:15] <=      16'b0000000000110000;
number_seven[6][0:15] <=      16'b0000000001110000;
number_seven[7][0:15] <=      16'b0000000011100000;
number_seven[8][0:15] <=      16'b0000000111000000;
number_seven[9][0:15] <=      16'b0000001110000000;
number_seven[10][0:15] <=     16'b0000011100000000;
number_seven[11][0:15] <=     16'b0000111000000000;
number_seven[12][0:15] <=     16'b0000110000000000;
number_seven[13][0:15] <=     16'b0000000000000000;
number_seven[14][0:15] <=     16'b0000000000000000;
```

```
    number_seven[15][0:15] <=        16'b0000000000000000;
    //number8
    number_eight[0][0:15] <=         16'b0000000000000000;
    number_eight[1][0:15] <=         16'b0000000000000000;
    number_eight[2][0:15] <=         16'b0000000000000000;
    number_eight[3][0:15] <=         16'b0000011111100000;
    number_eight[4][0:15] <=         16'b0000111111110000;
    number_eight[5][0:15] <=         16'b0000110000110000;
    number_eight[6][0:15] <=         16'b0000110000110000;
    number_eight[7][0:15] <=         16'b0000011111100000;
    number_eight[8][0:15] <=         16'b0000011111100000;
    number_eight[9][0:15] <=         16'b0000110000110000;
    number_eight[10][0:15] <=        16'b0000110000110000;
    number_eight[11][0:15] <=        16'b0000111111110000;
    number_eight[12][0:15] <=        16'b0000011111100000;
    number_eight[13][0:15] <=        16'b0000000000000000;
    number_eight[14][0:15] <=        16'b0000000000000000;
    number_eight[15][0:15] <=        16'b0000000000000000;
    //number9
    number_nine[0][0:15] <=                  16'b0000000000000000;
    number_nine[1][0:15] <=                  16'b0000000000000000;
    number_nine[2][0:15] <=                  16'b0000000000000000;
    number_nine[3][0:15] <=                  16'b0000011111100000;
    number_nine[4][0:15] <=                  16'b0000111111110000;
    number_nine[5][0:15] <=                  16'b0000110000110000;
    number_nine[6][0:15] <=                  16'b0000110000110000;
    number_nine[7][0:15] <=                  16'b0000111111110000;
    number_nine[8][0:15] <=                  16'b0000011111110000;
    number_nine[9][0:15] <=                  16'b0000000000110000;
    number_nine[10][0:15] <=         16'b0000000000110000;
    number_nine[11][0:15] <=         16'b0000111111110000;
    number_nine[12][0:15] <=         16'b0000111111100000;
    number_nine[13][0:15] <=         16'b0000000000000000;
    number_nine[14][0:15] <=         16'b0000000000000000;
    number_nine[15][0:15] <=         16'b0000000000000000;
  end else if (chipselect && write)
   case (address)
     3'h0 : head_x <= writedata;
     3'h1 : head_y <= writedata;
        3'h2 : snake_length <= writedata;
     3'h3 : apple_x <= writedata;
     3'h4 : apple_y <= writedata;
   endcase
   always_ff @(posedge clk)
     if(reset)begin
        pos_x[0] <= 50;
        pos_y[0] <= 40;
        pos_x[1] <= 51;
        pos_y[1] <= 40;
     end
     else begin
     if(pos_x[0] != head_x||pos_y[0] != head_y )begin
       extend<=1;
       score_a <= snake_length%10;
       score_b <= (snake_length/10)%10;
       score_c <= snake_length/100;
       for (int j=200; j>0;j--)begin
        if(j<=snake_length-1)begin
         pos_x[j]=pos_x[j-1];
         pos_y[j]=pos_y[j-1];
        end
       end
       pos_x[0]=head_x;
       pos_y[0]=head_y;
       extend<=0;
       /*if(pos_x[1]>30&&pos_y[1]<=40&&pos_x[1]<70)begin pos_x[0]=pos_x[1]-1;
pos_y[0]=pos_y[1];end
        else if(pos_x[1]<=30&&pos_y[1]<60) begin pos_x[0]=pos_x[1]; pos_y[0]=pos_y[1]+1;end
         else if(pos_x[1]>30&&pos_y[1]>=60&&pos_x[1]<70) begin pos_x[0]=pos_x[1]+1;
pos_y[0]=pos_y[1];end
          else if(pos_x[1]>=70) begin pos_x[0]=pos_x[1]; pos_y[0]=pos_y[1]-1;end
       sim_time = 0;*/
     end
     end

     always_comb begin
     {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
     if (VGA_BLANK_n && !extend)begin
      {VGA_R, VGA_G, VGA_B} ={8'h0, 8'h0, 8'h80};
      for(int i=0; i<200;i++) begin
       if(i<snake_length&&i!=0)begin
        if (hcount > 12*pos_x[i] && hcount < 12*(pos_x[i]+1)
                    && vcount > 6*pos_y[i] && vcount < 6*(pos_y[i]+1))
         {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        end
       else if(i==0&&hcount > 12*pos_x[i] && hcount < 12*(pos_x[i]+1)
                  && vcount > 6*pos_y[i] && vcount < 6*(pos_y[i]+1)) {VGA_R, VGA_G, VGA_B}
={8'h0, 8'h80, 8'h0};
             /*else if (hcount[10:6] == 5'd3 &&
              vcount[9:5] == 5'd3)
             {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};*/

      end
     if (hcount > 12*apple_x && hcount < 12*(apple_x+1)
                  && vcount > 6*apple_y && vcount < 6*(apple_y+1))
        {VGA_R, VGA_G, VGA_B} = {8'h80, 8'h0, 8'h0};
     if ((hcount>=0&&hcount<12) || (hcount >= 79*12 && hcount<80*12)
                  || (vcount >= 0&&vcount<6&& hcount<79*12)||(vcount >= 79*6&&vcount<80*6&&
hcount<79*12))
        {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};


      if (hcount >= 1100 && hcount <1116
                  && vcount >=30 && vcount < 46&&score_a==0)begin
        if(number_zero[vcount-30][hcount-1100]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1100 && hcount <1116
                  && vcount >=30 && vcount < 46&&score_a==1)begin
        if(number_one[vcount-30][hcount-1100]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1100 && hcount <1116
                  && vcount >=30 && vcount < 46&&score_a==2)begin
        if(number_two[vcount-30][hcount-1100]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1100 && hcount <1116
                  && vcount >=30 && vcount < 46&&score_a==3)begin
        if(number_three[vcount-30][hcount-1100]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1100 && hcount <1116
                  && vcount >=30 && vcount < 46&&score_a==4)begin
        if(number_four[vcount-30][hcount-1100]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
```

```
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1100 && hcount <1116
                && vcount >=30 && vcount < 46&&score_a==5)begin
        if(number_five[vcount-30][hcount-1100]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1100 && hcount <1116
                && vcount >=30 && vcount < 46&&score_a==6)begin
        if(number_six[vcount-30][hcount-1100]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1100 && hcount <1116
                && vcount >=30 && vcount < 46&&score_a==7)begin
        if(number_seven[vcount-30][hcount-1100]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1100 && hcount <1116
                && vcount >=30 && vcount < 46&&score_a==8)begin
        if(number_eight[vcount-30][hcount-1100]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1100 && hcount <1116
                && vcount >=30 && vcount < 46&&score_a==9)begin
        if(number_nine[vcount-30][hcount-1100]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end




      if (hcount >= 1075 && hcount <1091
                && vcount >=30 && vcount < 46&&score_b==0)begin
        if(number_zero[vcount-30][hcount-1075]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1075 && hcount <1091
                && vcount >=30 && vcount < 46&&score_b==1)begin
        if(number_one[vcount-30][hcount-1075]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1075 && hcount <1091
                && vcount >=30 && vcount < 46&&score_b==2)begin
        if(number_two[vcount-30][hcount-1075]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1075 && hcount <1091
                && vcount >=30 && vcount < 46&&score_b==3)begin
        if(number_three[vcount-30][hcount-1100]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1075 && hcount <1091
                && vcount >=30 && vcount < 46&&score_b==4)begin
        if(number_four[vcount-30][hcount-1075]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1075 && hcount <1091
                && vcount >=30 && vcount < 46&&score_b==5)begin
        if(number_five[vcount-30][hcount-1075]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1075 && hcount <1091
                && vcount >=30 && vcount < 46&&score_b==6)begin
        if(number_six[vcount-30][hcount-1075]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1075 && hcount <1091
                && vcount >=30 && vcount < 46&&score_b==7)begin
        if(number_seven[vcount-30][hcount-1075]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1075 && hcount <1091
                && vcount >=30 && vcount < 46&&score_b==8)begin
        if(number_eight[vcount-30][hcount-1075]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1075 && hcount <1091
                && vcount >=30 && vcount < 46&&score_b==9)begin
        if(number_nine[vcount-30][hcount-1075]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end



     if (hcount >= 1050 && hcount <1066
                && vcount >=30 && vcount < 46&&score_c==0)begin
        if(number_zero[vcount-30][hcount-1050]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1050 && hcount <1066
                && vcount >=30 && vcount < 46&&score_c==1)begin
        if(number_one[vcount-30][hcount-1050]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1050 && hcount <1066
                && vcount >=30 && vcount < 46&&score_c==2)begin
        if(number_two[vcount-30][hcount-1050]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1050 && hcount <1066
                && vcount >=30 && vcount < 46&&score_c==3)begin
        if(number_three[vcount-30][hcount-1050]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1050 && hcount <1066
                && vcount >=30 && vcount < 46&&score_c==4)begin
        if(number_four[vcount-30][hcount-1050]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1050 && hcount <1066
                && vcount >=30 && vcount < 46&&score_c==5)begin
        if(number_five[vcount-30][hcount-1050]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
```

```
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1050 && hcount <1066
                && vcount >=30 && vcount < 46&&score_c==6)begin
        if(number_six[vcount-30][hcount-1050]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff, 8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1050 && hcount <1066
                && vcount >=30 && vcount < 46&&score_c==7)begin
        if(number_seven[vcount-30][hcount-1050]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1050 && hcount <1066
                && vcount >=30 && vcount < 46&&score_c==8)begin
        if(number_eight[vcount-30][hcount-1050]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
      if (hcount >= 1050 && hcount <1066
                && vcount >=30 && vcount < 46&&score_c==9)begin
        if(number_nine[vcount-30][hcount-1050]==1) {VGA_R, VGA_G, VGA_B} = {8'hff, 8'hff,
8'hff};
        else {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h80};
      end
     end
     end

endmodule

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

/*
 * 640 X 480 VGA timing for a 50 MHz clock: one pixel every other cycle
 *
 * HCOUNT 1599 0           1279       1599 0
 *           _______________          ________
 * ___________|   Video     |____________|  Video
 *
 *
```

```
* |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE
*      ___________________________     _______________
* |____|       VGA_HS          |____|
*/
  // Parameters for hcount
  parameter HACTIVE      = 11'd 1280,
            HFRONT_PORCH = 11'd 32,
            HSYNC        = 11'd 192,
            HBACK_PORCH  = 11'd 96,
            HTOTAL       = HACTIVE + HFRONT_PORCH + HSYNC +
                           HBACK_PORCH; // 1600

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

  logic endOfLine;

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

  assign endOfLine = hcount == HTOTAL - 1;

  logic endOfField;

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

  assign endOfField = vcount == VTOTAL - 1;

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

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

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

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

endmodule
```