# SNAKE+

## CSEE 4840 Embedded System Design Final Report

Nina Berg - nb2555
Joseph Corbisiero - jc3790
Ilan Elkobi - ie2147
Molly Karcher - mdk2133
Brian Wagner - bhw2113

# Table of Contents

# 1 Introduction

This game was inspired by the old Nokia cell phone game snake from the 1990s. Since the creation of the classic handheld game, attempted remakes have consistently fallen short of the original, leaving us ’90s children wanting. We aimed to recreate this snake game with two things in mind: keeping the memorable game style reminiscent of the original, but also improving it by adding more exciting gameplay, multiple players, power-ups, and better graphics.

The game allows for a multiplayer version, so that two players can play together in a battle to the death! The players will use old Nintendo NES controllers, which allows us to stay true to the original game—players can only move up, down, right, or left. In addition, the use of the buttons on the NES controllers allows us to add extra features, such as starting and pausing the game manually, and utilizing functions of some special power-ups.

Throughout the game, pieces of food and power-ups appear randomly around the screen, to which all snakes must race to be the first ones to eat. Most of the gameplay is as you would expect. Eating pieces of food will lengthen your own tail, and colliding with your own tail or with a wall will cause you to lose the game. When adding in multiple players, you will also lose the game if your head collides with another player’s tail. The power-ups add fun effects to the game, such as speeding up your snake or slowing down your opponent, and make it much more interesting than your traditional snake game, allowing for opportunities to interact more with the other players. Throughout the game, there will be sound effects associated with certain game actions, such as eating food, receiving a power-up, and losing or winning the game.

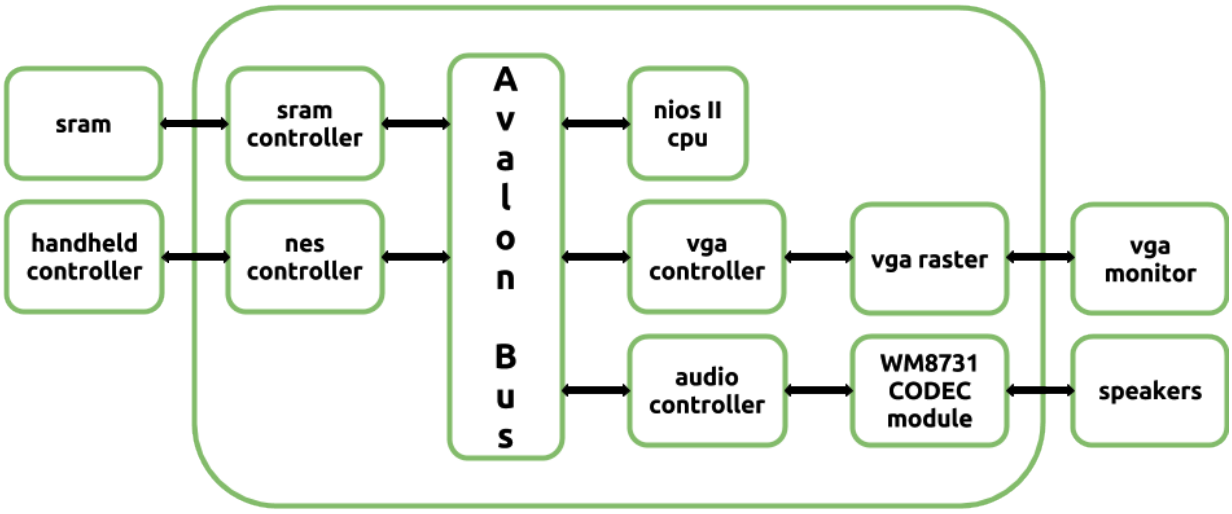
On order to play the game, all players will hold their individual NES controllers all connected to the same Altera DE2 Board, VGA Display, and set of speakers.

# 2 Design & Architecture

Our system is built using the Altera Cyclone II FPGA, and is comprised of three main modules:

- NES Controller Inputs
- VGA Display and Graphics
- Audio Outputs

These components will be controlled by the NIOS Processor after being programmed onto the FPGA. All components are connected through the Avalon Bus and all software will be run on the processor. The overall design of our architecture as a block diagram is shown below. This diagram is fairly general and shows each component at a very high level. However, each of the following sections will show a slightly lower-level diagram so that all the components of our architecture can be seen. All peripherals and components are explained in detail in the following sections.




## 2.1 Hardware

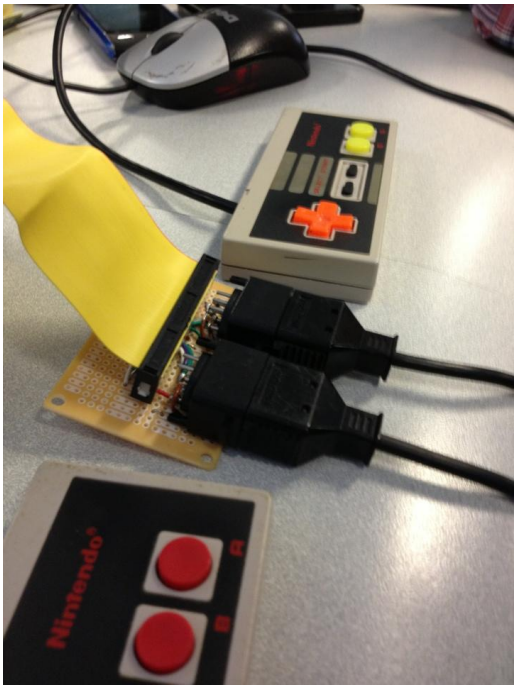
As discussed above, in order to modernize the game of snake, three crucial hardware components were needed. We needed to update the graphics, which was all handled by the VGA Controller and it's helper functions. We needed to implement a Nintendo NES Controller so that we could use them to play the game, and we needed to add an Audio Controller to play sound effects throughout the game. These individual components are all described in detail below.

### *2.1.1 NES Controller*

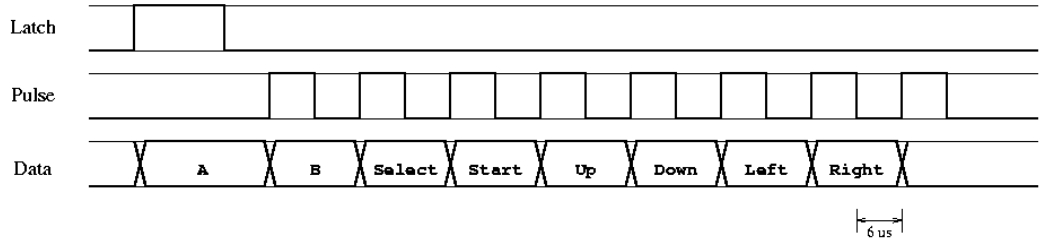
Integration of the Nintendo NES controllers proved to be a crucial part of our goal to create a more interesting interpretation of Snake. For those unfamiliar with the classic controllers, an image is provided below.


For gameplay, players will utilize all available buttons on the controller. The “up,” “down,” “left,” and “right” buttons on the controller’s d-pad control the direction of the snake. The “select” button can be used to pause the game, the “start” button can be used to pause the game, the “A” button can be used to utilize the “freeze” power-up once it’s attained, and the “B” button can be used to utilize the “edwards” power-up once it is attained.

To connect the NES controllers to the DE2 board, we used the general I/O pins. In order to make the adapters on the NES controllers compatible, we purchased controller sockets, and wired the pins in the controllers to in the appropriate configuration. A picture is below.

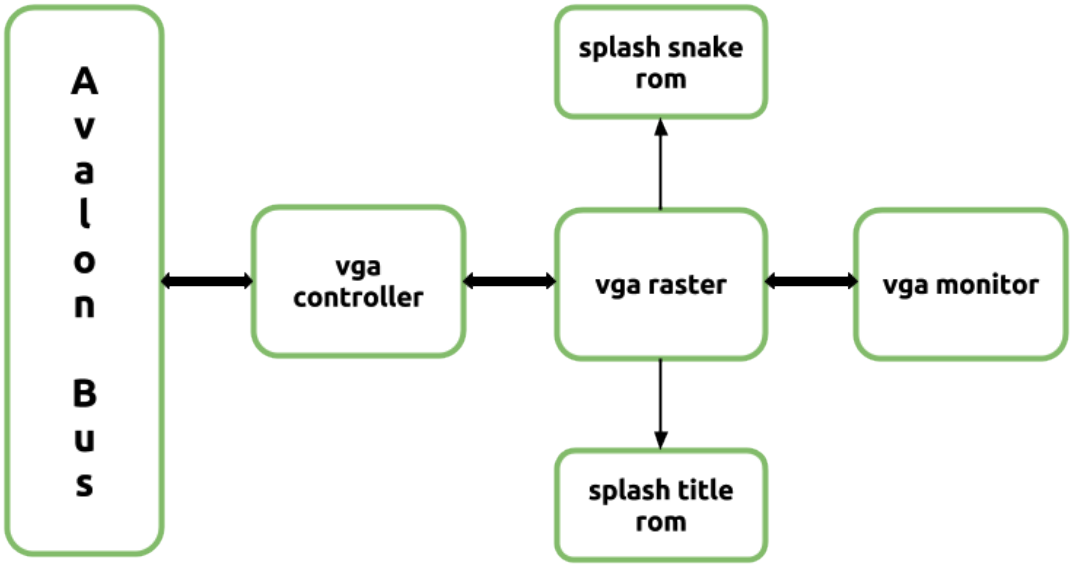


Once the NES controllers were adapted, we constructed an FSM in hardware to interpret the protocol. The NES protocol is slightly unconventional: the state of each button is sent as part of a serial byte. Our FSM is able to determine which button(s) the user pressed, and communicate that to the software. A picture of the protocol is below.




### *2.1.2 Graphics*

All of our graphics were are controlled through the VGA controller. Below is a more specific architectural block diagram, including ROM elements required to support our graphics.

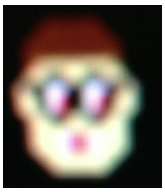


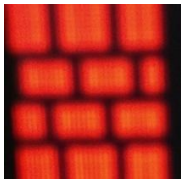

You will notice that we only require ROM to store the splash screen, shown at the beginning of a game of Snake+. We do not require any addition ROM for the sprites because they are stored in SRAM.

#### 2.1.2.1 Sprites

The graphics of our game are controlled by the de2_vga_controller.vhd file, which appropriates which sprites (whose definitions are specified within de2_vga_raster.vhd) will be drawn to the

VGA Display. The following is a comprehensive list of all sprites used in our game. Images were difficult to capture on the VGA screen, we apologize for the low-quality images.

| SPRITE | IMAGE |
|---|---|
| Snake Head |  |
| Snake Body |  |
| Snake Body Turn | |
| Snake Tail |  |
| Rabbit (food) |  |
| Mouse (food) |  |
| Edwards (power-up) |  |

| | |
|---|---|
| Freeze (power-up) |  |
| Speed (power-up) |  |
| Brick (obstacle) |  |

Because our snake is a moving part, we must also store different sprites in order to draw a snake head (or tail) when the snake is moving in a direction other than rightwards. As a result, we have separate directional sprites for a snake head moving upwards, downwards, leftwards, and rightwards. The same is true of snake tails, snake body, and snake body-turn pieces.

Sprites are stored in a series of 16x16 bit matrices, allotting one 16x16 matrix per color that needs to be drawn into a given snake. For example, in order to store a single snake body sprite, we need four 16x16 matrices; one to store the orange coloring, one to store the gray coloring, one to store the white coloring, and one to store the black coloring. Each element of each 16x16 matrix consists of a single bit, indicating whether or not the given color is present in the final image of the sprite. To illustrate this more clearly, we have provided the following code snippets for storage of our snake body sprite.

```
--sprite body coloring
sprite_body_right_black(0)          <=      "0000000110000000";
sprite_body_right_black(1)          <=      "0000000000000000";
sprite_body_right_black(2)          <=      "0000000000000000";
sprite_body_right_black(3)          <=      "0000000000000000";
sprite_body_right_black(4)          <=      "0000000000000000";
sprite_body_right_black(5)          <=      "0000000000000000";
sprite_body_right_black(6)          <=      "0000000000000000";
sprite_body_right_black(7)          <=      "0000000000000000";
sprite_body_right_black(8)          <=      "0000000000000000";
sprite_body_right_black(9)          <=      "0000000000000000";
sprite_body_right_black(10)                 <=      "0000000000000000";
sprite_body_right_black(11)                 <=      "0000000000000000";
```

```
sprite_body_right_black(12)                <=      "1111111111111111";
sprite_body_right_black(13)                <=      "1111111111111111";
sprite_body_right_black(14)                <=      "0000000000000000";
sprite_body_right_black(15)                <=      "1111111111111111";

sprite_body_right_grey(0)             <=      "1111111000000000";
sprite_body_right_grey(1)             <=      "0000000000000000";
sprite_body_right_grey(2)             <=      "1111111100000000";
sprite_body_right_grey(3)             <=      "1111111100000000";
sprite_body_right_grey(4)             <=      "1111111100000000";
sprite_body_right_grey(5)             <=      "1111111100000000";
sprite_body_right_grey(6)             <=      "1111111100000000";
sprite_body_right_grey(7)             <=      "1111111100000000";
sprite_body_right_grey(8)             <=      "1111111100000000";
sprite_body_right_grey(9)             <=      "1111111100000000";
sprite_body_right_grey(10)            <=      "1111111100000000";
sprite_body_right_grey(11)            <=      "1111111100000000";
sprite_body_right_grey(12)            <=      "0000000000000000";
sprite_body_right_grey(13)            <=      "0000000000000000";
sprite_body_right_grey(14)            <=      "1111111100000000";
sprite_body_right_grey(15)            <=      "0000000000000000";

sprite_body_right_orange(0)                <=      "0000000001111111";
sprite_body_right_orange(1)                <=      "0000000000000000";
sprite_body_right_orange(2)                <=      "0000000011111111";
sprite_body_right_orange(3)                <=      "0000000011111111";
sprite_body_right_orange(4)                <=      "0000000011111111";
sprite_body_right_orange(5)                <=      "0000000011111111";
sprite_body_right_orange(6)                <=      "0000000011111111";
sprite_body_right_orange(7)                <=      "0000000011111111";
sprite_body_right_orange(8)                <=      "0000000011111111";
sprite_body_right_orange(9)                <=      "0000000011111111";
sprite_body_right_orange(10)               <=      "0000000011111111";
sprite_body_right_orange(11)               <=      "0000000011111111";
sprite_body_right_orange(12)               <=      "0000000000000000";
sprite_body_right_orange(13)               <=      "0000000000000000";
sprite_body_right_orange(14)               <=      "0000000011111111";
sprite_body_right_orange(15)               <=      "0000000000000000";

sprite_body_right_white(0)            <=      "0000000000000000";
sprite_body_right_white(1)            <=      "1111111111111111";
sprite_body_right_white(2)            <=      "0000000000000000";
```

```
sprite_body_right_white(3)              <=      "0000000000000000";
sprite_body_right_white(4)              <=      "0000000000000000";
sprite_body_right_white(5)              <=      "0000000000000000";
sprite_body_right_white(6)              <=      "0000000000000000";
sprite_body_right_white(7)              <=      "0000000000000000";
sprite_body_right_white(8)              <=      "0000000000000000";
sprite_body_right_white(9)              <=      "0000000000000000";
sprite_body_right_white(10)                     <=      "0000000000000000";
sprite_body_right_white(11)                     <=      "0000000000000000";
sprite_body_right_white(12)                     <=      "0000000000000000";
sprite_body_right_white(13)                     <=      "0000000000000000";
sprite_body_right_white(14)                     <=      "0000000000000000";
sprite_body_right_white(15)                     <=      "0000000000000000";
```

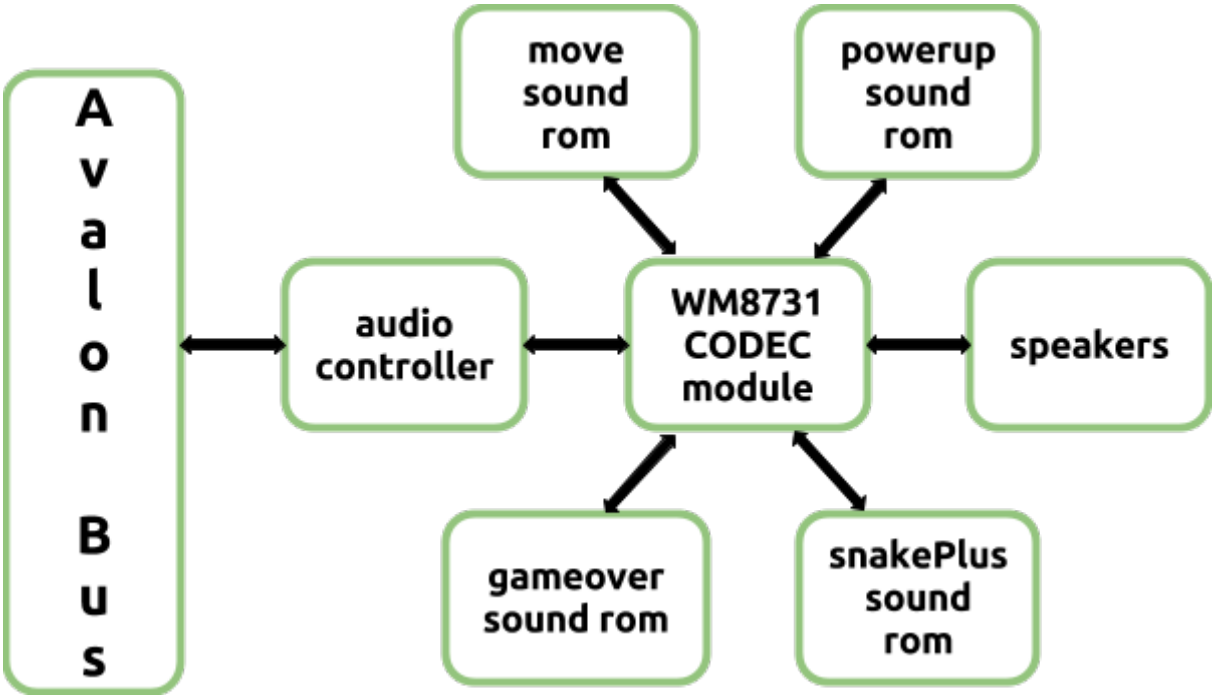
This series of matrices are drawn over one another, but colors are only drawn in a specific pixel when there is a "1" present in the specified matrix. Thus, when we draw each sprite, we're really drawing over each pixel multiple times, depending on how many colors are present in the image. But because of this storage system, we are not limited on how many colors we are able to draw, we could draw as many colors as how many matrices we were able to store.

#### 2.1.2.2 Splash Screen

The other main graphic component of our game is the splash screen shown at the very beginning of the game. The image is shown here:


It is worth noting that due to its size, the splash screen was handled completely differently than we handled the sprites in our design. The splash screen image of the snake and the splash screen title "Snake+" are each stored separately in ROM. Whether or not the splash screen is displayed is still controlled by de2_vga_controller.vhd. They are simply mapped to their corresponding .mif file, and stored in ROM when the board is programmed. The controller determines whether the splash screen needs to be enabled, and alerts the raster to display it on the VGA.

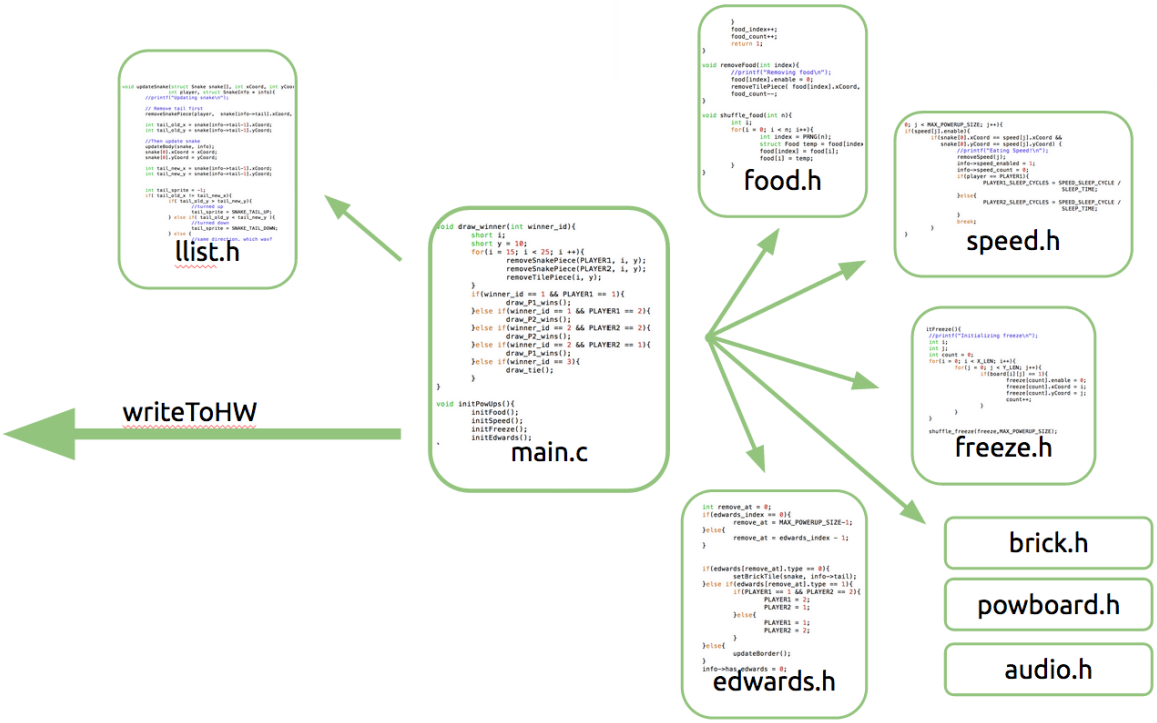
*2.1.3 Audio Controller*




Sound effects are played whenever a snake moves, eats food/powerups, when the game ends, and when the splash screen is displayed. All sounds were found online, with the exception of the splash screen sound which we recorded. The sounds were resampled at 2 kHz, and stored in different roms. A read signal sent to the audio controller will trigger sound playback, the address "read from" determines which sound plays. A separate counter is used to index into each sound rom, and playback continues until the counter reaches the number of samples stored in the rom (hard-coded for each sound).

## 2.2 Software & Game Logic

The software and game logic for Snake+ are relatively simple, but do require some explanation. All of the software we have written is implemented in the C programming language. The overall structure of our software, including all vital .c and .h files can be simply described by the following block diagram:




### *2.2.1 Setup*

We use arrays to keep track of gameplay objects, such as snakes, edibles, and obstacles. A pseudo-random number generator is used to populate an array of food and power-ups with various locations. The PRNG is seeded based on how long it takes for the players to press start (a counter is kept in hardware). Then, the snakes are initialized with only a head and a tail. The snakes always start in the same positions. Afterwards, three pieces of food, and one of each kind of power-up is placed on the board. Whenever the software is placing a piece of food or a power-up, it checks to ensure that the location of the next unused item in the array is not where a snake, obstacle, or other edible is located.

### *2.2.2 Food*

The game will always keep three food objects on the board as long as it has room for them. Once one of the players reaches one, it will be “eaten” and removed, and the corresponding player’s snake will grow. This is done by adding to the end of the snake. When a piece of food is eaten, it is marked as disabled in the array, and the next food item in the array will be placed on the board (as long as it is not set for a location that would collide with a snake, obstacle, or other edible).

### *2.2.3 Power-ups*

Our game features three types of power-ups. Power-ups are technically “edibles” in that snakes must run them over in order to obtain or activate them. Some power-ups are granted

instantaneously upon being acquired, and others are deployed by pressing a button on the controller. A listing of power-ups, their effects, and how they are deployed is below.

| Power-up | Effect | Use |
|---|---|---|
| Speed | Grants a temporary boost in speed to the snake that eats it | Instantaneous |
| Freeze | Stops the other snake for a short duration | *A* button on the NES controller |
| Edwards | **Either:** snake will drop an obstacle in its wake **or** the snakes will switch controllers | *B* button on the NES controller |

### *2.2.4 Multi-Player*

Possibly the greatest feature in Snake+ is the support for multiple players. By default, the game is a two-player game, though it can easily be adapted to three or four players. All players use the same screen and Altera board, but have their own NES controller. The controllers are connected to the FPGA using its general I/O pins, and their inputs are read by the hardware using the protocol described above.

In software, each player is represented by a unique snake struct. These keep track of the location of the snake and its size. The inputs of the controllers are also read separately, so that the button presses are only mapped to the correct snakes. The software is optimized for speed (avoiding time-consuming operators and computation) so that there will not be a lag with an increase in the number of players.

### *2.2.5 Game Logic*

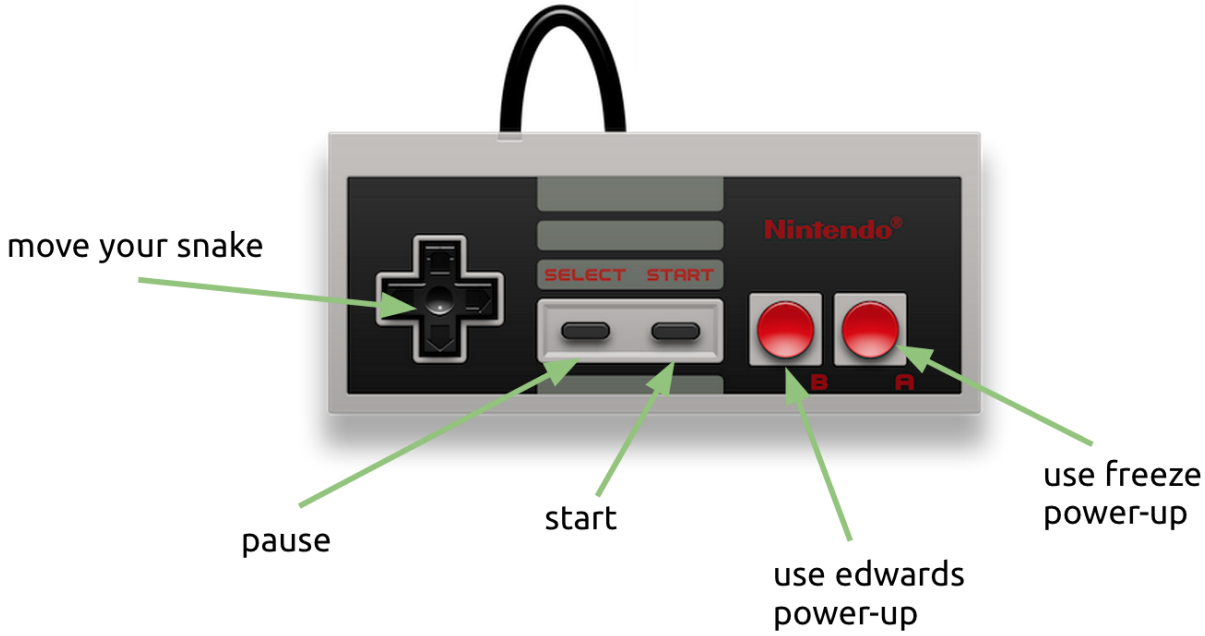
Because there are multiple players in Snake+, we've done away with the original game's notion of a time-based score. Players are no longer competing with their personal records for how long they can survive, but are rather engaged in a battle to the death with other snakes. The strategy remains the same, however. Survive!

There is no "score" per se, but rather each game results in a winner and a loser. The rules are as follows:  a player loses if his or her snake crashes into an obstacle, its own tail, or another player's tail. Once one player loses, the other player is declared the winner. There can also be a draw in the event that both players crash at the same time, or they crash into each other's heads.

# 3 Playing Snake+

## 3.1 Using the NES Controller

Players control their snakes using a NES controller. A diagram of how to use the controller is included below:




## 3.2 Playing the Game

### *3.2.1 Starting and Playing*

The game starts when one of the players presses the start button on their controller. Once that happens, the game screen replaces the splash screen and the game begins. The game screen is initialized with pieces of food and power-ups already placed at random locations throughout the screen. The two players move their snakes around the screen using the NES controllers and race towards the different items.

### *3.2.2 Game Over and Restarting*

Once one of the players loses, a game over sound will be heard, and the winning player is printed on the screen. The game will then wait for one of the players to press the start button. Once that happens, the game returns to the splash screen and again waits for someone to press start and begin a new game.

# 4 Summary

## 4.1 Responsibilities

Throughout the long development process of Snake+, all team members basically ended up contributing to all of the different components of the system. However, the main contributions of everyone fell into the following divisions:

Brian: VGA raster, sprite design, and software
Molly: Sprite design and VGA raster/controller
Ilan: VGA controller, NES controller, NES-to-Altera hardware
Nina: Audio and software
Joe: Software, design of gameplay and game elements

## 4.2 Lessons Learned & Advice to Future Teams

**There Are Many Solutions, but Not All Are Good**
As with designing software algorithms, when choosing how to implement something in hardware (how to encode sprites, how to divide the game screen into tiles, etc.), often several methods will seem feasible, and choosing the correct one isn't always easy. A couple of times during our project, we discovered that a method we had decided on was not working. When this sort of thing happened, we had to backtrack, revert to an old version of our program, and rework things. Often it's hard to accept that what you've been working on for two days straight has to be completely scrapped, but the best thing to do is accept this early and move on.

**Good Milestones Are Key**
When planning out your design, take care in dividing up the work into the individual milestones. This is the single most important factor in determining how happy or miserable you will be during the rest of the semester. If the work is divided evenly, you will only have to spend a few crazed nights in the lab before each milestone. If the deadlines are uneven, you can easily overload either the middle of the semester or finals week. We did okay here, and we're grateful for it.

**A Little Design Goes a Long Way**
Most projects that people make in this class, like ours, are games, or something else that will make heavy use of sprites. As we progressed through the milestones, we stuck with our original sprites, which were quickly assembled early on. However, towards the end once we began replacing them with prettier versions, we were surprised by how much this impacted the overall appeal of our game. Apparently, looks matter, at least in hardware.

**Patience is a Virtue; Quartus will test it**
Embedded system design can be extraordinarily frustrating. The amount of times we were faced with strange hardware glitches and dysfunctional software (read: "error downloading elf file") is too high for us to remember. We must have redone our SOPC file hundreds of times along the

way as we uncovered glitches, and it turns out that just “restarting it,” whether “it” was the Altera board, Quartus, or Nios, would fix a surprisingly large number of problems. In short: try to be patient. Things will work out (maybe).

# 5 Code Listing

```
*****************************************************************************************************
main.c


#include "basic_io.h"
#include <alt_types.h>
#include <unistd.h>
#include "snake_io.h"
#include "powboard.h"
#include "llist.h"
#include "food.h"
#include "speed.h"
#include "freeze.h"
#include "brick.h"
#include "drop_brick.h"
#include "edwards.h"
#include "constants.h"
#include "audio.h"

/* should update snake when new direction is known*/
/* go left */
void moveLeft(int dir_array []){
	//printf("Changed direction to left\n");
	dir_array[right_dir]  = 0;
	dir_array[left_dir]   = 1;
	dir_array[up_dir]            = 0;
	dir_array[down_dir]   = 0;
}

/* go right */
void moveRight(int dir_array []){
	//printf("Changed direction to right\n");
	dir_array[right_dir]  = 1;
	dir_array[left_dir]   = 0;
	dir_array[up_dir]            = 0;
	dir_array[down_dir]   = 0;
```

```
}

/* go up */
void moveUp(int dir_array []){
	//printf("Changed direction to up\n");
	dir_array[right_dir]  = 0;
	dir_array[left_dir]   = 0;
	dir_array[up_dir]            = 1;
	dir_array[down_dir]   = 0;
}

/* go down */
void moveDown(int dir_array []){
	//printf("Changed direction to down\n");
	dir_array[right_dir]  = 0;
	dir_array[left_dir]   = 0;
	dir_array[up_dir]            = 0;
	dir_array[down_dir]   = 1;
}

// track snake movement
int movement(alt_u8 key, struct Snake snake[], int dir_array [],
		int pressed, int player, struct SnakeInfo * info){

	int old_dir = -1;
	if( dir_array[left_dir] )
		old_dir = left_dir;
	else if( dir_array[right_dir] )
		old_dir = right_dir;
	else if( dir_array[up_dir] )
		old_dir = up_dir;
	else if( dir_array[down_dir] )
		old_dir = down_dir;

	int xCoor = snake[0].xCoord;
	int yCoor = snake[0].yCoord;

	int pressed_dir = get_dir_from_pressed(pressed);

	//printf("Pressed: %d\n", pressed);
```

```
switch(pressed_dir){
case 1://0x1C://'a'
        if( dir_array[up_dir] || dir_array[down_dir] )
                moveLeft(dir_array);
        break;
case 2://0x1D://'w'
        if( dir_array[left_dir] || dir_array[right_dir] )
                moveUp(dir_array);
        break;
case 0://0x23://'d'
        if( dir_array[up_dir] || dir_array[down_dir] )
                moveRight(dir_array);
        break;
case 3://0x1B://'s'
        if( dir_array[left_dir] || dir_array[right_dir])
                moveDown(dir_array);
        break;
default:
        break;
}
if(dir_array[right_dir]){
        xCoor+= 1;//16;
        if(xCoor > RIGHT_BOUND - 2/**col_offset*/){
                //printf("Collision with right boundary!\n");
                return 1;
        }
        updateSnake(snake, xCoor, yCoor, right_dir, old_dir, player, info);
        dir_arg = right_dir;
        //checkFood(snake, right_dir, player, info);
}else if(dir_array[left_dir]){
        xCoor-=1;//16;
        if(xCoor < (LEFT_BOUND+1)){//16
                //printf("Collision with left boundary!\n");
                return 1;
                //collision
        }
        updateSnake(snake, xCoor, yCoor, left_dir, old_dir, player, info);
        dir_arg = left_dir;
        //checkFood(snake, left_dir, player, info);
}else if(dir_array[up_dir]){
        yCoor-=1;//16;
```

```
            if(yCoor < (TOP_BOUND + 1)){//16
                    //printf("Collision with top boundary!\n");
                    return 1;
                    //collision
            }
            updateSnake(snake, xCoor, yCoor, up_dir, old_dir,player, info);
            dir_arg = up_dir;
            //checkFood(snake, up_dir, player, info);
        }else if(dir_array[down_dir]){
            yCoor+=1;//16;
            if(yCoor > BOT_BOUND - 2/**col_offset*/){
                    //printf("Collision with bottom boundary!\n");
                    return 1;
                    //collision
            }
            updateSnake(snake, xCoor, yCoor, down_dir, old_dir,player, info);
            dir_arg = down_dir;
            //checkFood(snake, down_dir, player, info);
        }
        /*checkFood(snake, down_dir, player, info);
        checkSpeed(snake, player, info);
        checkFreeze(snake, player, info);
        recalc_freeze_times(snake, player,info);
        checkEdwards(snake, player, info);*/
        //PLAY_SOUND(1);
        int check_brick_col = brickCol(snake, player);
        int check_self_col = traverseList(snake, player);
        if(check_brick_col || check_self_col){
            return 1;
        }else{
            return 0;
        }

        //return traverseList(snake);
        //printf("x: %d y: %d\n", xCoor, yCoor);
}



void writeToHW(struct Snake snake[], int dir, int old_dir, int player,
```

```
        int xCoord, int yCoord, struct SnakeInfo *info, int tail_sprite) {

char sprite = 1;

char sprite_second;

/* Figure out head */
if(dir == right_dir)
        sprite = SNAKE_HEAD_RIGHT;
else if(dir == left_dir)
        sprite = SNAKE_HEAD_LEFT;
else if(dir == down_dir)
        sprite = SNAKE_HEAD_DOWN;
else if(dir == up_dir)
        sprite = SNAKE_HEAD_UP;

if( dir == old_dir ){
        sprite_second = sprite + 8; //Trick bc head and body are offset by 8
} else {
        /* Turn piece */
        if( (dir == right_dir && old_dir == up_dir) ||
                        (dir == down_dir && old_dir == left_dir)){
                //printf("old dir:%d new dir:%d", old_dir, dir);
                sprite_second = SNAKE_TURN_UP_RIGHT;
        }
        else if( ((dir == down_dir) && (old_dir == right_dir)) ||
                        ((dir == left_dir) && (old_dir == up_dir))){
                sprite_second = SNAKE_TURN_RIGHT_DOWN;
        }
        else if( ((dir == left_dir) && (old_dir == down_dir)) ||
                        ((dir == up_dir) && (old_dir == right_dir))){
                sprite_second = SNAKE_TURN_DOWN_LEFT;
        }
        else if( ((dir == up_dir) && (old_dir == left_dir)) ||
                        ((dir == right_dir) && (old_dir == down_dir))){
                sprite_second = SNAKE_TURN_LEFT_UP;
        }
}

/* originallly divided x and ys by 16 */
```

```
	addSnakePiece(player, sprite,  snake[0].xCoord, snake[0].yCoord);
	addSnakePiece(player, sprite_second, snake[1].xCoord, snake[1].yCoord);
	addSnakePiece(player, tail_sprite, snake[info->tail].xCoord,  snake[info-
>tail].yCoord);
}

void check_powerup_col(struct Snake snake[], struct Snake other_snake[], int dir_arg, int
player, struct SnakeInfo * info){
	checkFood(snake, other_snake, dir_arg, player, info);
	checkSpeed(snake, other_snake, player, info);
	checkFreeze(snake,other_snake, player, info);
	recalc_freeze_times(snake, player,info);
	checkEdwards(snake, other_snake, player, info);
}



void check_powerup_buttons(int pressed,struct Snake snake[], struct Snake other_snake[],
struct Freeze freeze[], int player, struct SnakeInfo * info){

	int button = get_button_from_pressed(pressed);
	if( button == A_CODE ){
		apply_freeze(snake, player, info);
	}
	if( button == B_CODE ){
		apply_edwards(snake, other_snake, player, info);
	}

}

void wait_for_continue(){
	while(1) {
		//printf("HIT ANY BUTTON TO RETURN TO MAIN MENU!\n");
		int pressed_player1          = getPlayer1Controller();
		int pressed_player2          = getPlayer2Controller();
		if (pressed_player1 != 0 || pressed_player2 != 0)
			break;
		seed++;
	}
}
```

```
void draw_P1_wins(){
	short y = 10;
	addTilePiece(P_CODE, 16, y);
	addTilePiece(ONE_CODE, 17, y);
	addTilePiece(W_CODE, 19, y);
	addTilePiece(I_CODE, 20, y);
	addTilePiece(N_CODE, 21, y);
	addTilePiece(S_CODE, 22, y);
	addTilePiece(EXC_CODE, 23,y);
}

void draw_P2_wins(){
	short y = 10;
	addTilePiece(P_CODE, 16, y);
	addTilePiece(TWO_CODE, 17, y);
	addTilePiece(W_CODE, 19, y);
	addTilePiece(I_CODE, 20, y);
	addTilePiece(N_CODE, 21, y);
	addTilePiece(S_CODE - 32, 22, y);
	addTilePiece(EXC_CODE, 23,y);
}

void draw_tie(){
	short y = 10;
	addTilePiece(T_CODE, 19, y);
	addTilePiece(I_CODE, 20, y);
	addTilePiece(E_CODE, 21, y);
	addTilePiece(EXC_CODE, 22, y);
}

void draw_winner(int winner_id){
	short i;
	short y = 10;
	for(i = 15; i < 25; i ++){
		removeSnakePiece(PLAYER1, i, y);
		removeSnakePiece(PLAYER2, i, y);
		removeTilePiece(i, y);
	}
	if(winner_id == 1 && PLAYER1 == 1){
		draw_P1_wins();
```

```
        }else if(winner_id == 1 && PLAYER1 == 2){
                draw_P2_wins();
        }else if(winner_id == 2 && PLAYER2 == 2){
                draw_P2_wins();
        }else if(winner_id == 2 && PLAYER2 == 1){
                draw_P1_wins();
        }else if(winner_id == 3){
                draw_tie();
        }
}

void initPowUps(){
        initFood();
        initSpeed();
        initFreeze();
        initEdwards();
}

void drawStartPowUps(struct Snake snake_player1[], struct Snake snake_player2[]){
        while( !drawFood(snake_player1, snake_player2) ){
                //printf("Attempting to Draw food\n");
        }
        while( !drawFood(snake_player1, snake_player2) ){
                //printf("Attempting to Draw food\n");
        }
        while( !drawFood(snake_player1, snake_player2) ){
                //printf("Attempting to Draw food\n");
        }
        while( !drawSpeed(snake_player1, snake_player2) ){
                //printf("Attempting to Draw speed\n");
        }
        //speed_drawn = 1;
        while( !drawFreeze(snake_player1, snake_player2) ){
                //printf("Attempting to Draw freeze\n");
        }
        //freeze_drawn = 1;
        while( !drawEdwards(snake_player1, snake_player2) ){
                //printf("Attempting to Draw edwards\n");
        }
}
```

```
void reset_software(){
        /* index to know which sprite to draw from power up structs */
        food_index                          = 0;
        speed_index                 = 0;
        freeze_index                = 0;
        edwards_index               = 0;

        freeze_pow_count            = 0;
        freeze_drawn                = 0;
        speed_pow_count             = 0;
        speed_drawn                 = 0;
        edwards_pow_count           = 0;
        edwards_drawn               = 0;
        dir_arg                             = 0;
        switch_snakes               = 0;
        food_count = 0;

        LEFT_BOUND                          = 0;
        RIGHT_BOUND                         = 40;
        BOT_BOUND                           = 30;
        TOP_BOUND                           = 0;
        /* used with edwards power up */
        NEW_BOT                             = 29;
        NEW_TOP                             = 0;
        NEW_LEFT                            = 0;
        NEW_RIGHT                           = 39;
        /*
         * 0 -> nobody yet
         * 1 -> snake1
         * 2 -> snake2
         * 3 -> draw
         */
        game_winner                         = 0;
        PLAYER1                             = 1;
        PLAYER2                             = 2;
        /* reinit border */
        //initBorder();
        /* reinit power up board */
        initPowBoard(board, seed);
```

```
}

void fancy_splash(){
	int i;
	int j;
	for(i = 10; i < X_LEN - 10; i++){
		for(j = 1; j < Y_LEN - 1; j++){
			addTilePiece(SPLASH_SNAKE_CODE,i,j);
			addTilePiece(SPLASH_SNAKE_CODE,i+1,j);
			addTilePiece(SPLASH_SNAKE_CODE,i+2,j);
		}
		usleep(2*25000);
		for(j = 1; j < Y_LEN - 1; j++){
			removeTilePiece(i,j);
			removeTilePiece(i+1,j);
			removeTilePiece(i+2,j);
		}
	}
	for(i = X_LEN - 10; i > 10; i--){
		for(j = Y_LEN - 1; j > 1; j--){
			addTilePiece(SPLASH_SNAKE_CODE,i+2,j);
			addTilePiece(SPLASH_SNAKE_CODE,i+1,j);
			addTilePiece(SPLASH_SNAKE_CODE,i,j);
		}
		usleep(2*25000);
		for(j = Y_LEN -1; j > 1; j--){
			removeTilePiece(i+2,j);
			removeTilePiece(i+1,j);
			removeTilePiece(i,j);
		}
	}
}

int main(){

	//printf("Press start to begin the game\n");
	int set = 1;
	while(1) {

		/* Reset display */
```

```
reset_hardware();
reset_software();
//initBorder();
if(set){
	fancy_splash();
	enable_splash_screen();
}

play_splash_sound();
play_sound();
//printf("Press any button to start!\n");

if(set){
	wait_for_continue();
	disable_splash_screen();

}

/* init border */
//initPowBoard(board, seed);
/* Reset snakes and display them */
//PLAYER1 = 1;
//PLAYER2 = 2;
struct Snake snake_player1[SNAKE_SIZE];
struct SnakeInfo info1;
initSnake(snake_player1, 256/16, 256/16, PLAYER1, &info1);
struct Snake snake_player2[SNAKE_SIZE];
struct SnakeInfo info2;
initSnake(snake_player2, 256/16, 320/16, PLAYER2, &info2);

PLAYER1_SLEEP_CYCLES = DEFAULT_SLEEP_CYCLE / SLEEP_TIME;
PLAYER2_SLEEP_CYCLES = DEFAULT_SLEEP_CYCLE / SLEEP_TIME;


//printf("Game is starting");


int player1_dir[4];
int player2_dir[4];
```

```
player1_dir[right_dir] = 1;
player1_dir[left_dir] = 0;
player1_dir[up_dir] = 0;
player1_dir[down_dir] = 0;

player2_dir[right_dir] = 1;
player2_dir[left_dir] = 0;
player2_dir[up_dir] = 0;
player2_dir[down_dir] = 0;

initBorder();
initPowUps();
drawStartPowUps(snake_player1, snake_player2);

if(!set){
       wait_for_continue();
}

set = 0;

int paused = 0;

unsigned char code;
int count_player1                   = 0;
int count_player2                   = 0;

int pressed_player1          = getPlayer1Controller();
int pressed_player2          = getPlayer2Controller();
int potential_pressed1;
int potential_pressed2;
int old_potential1;
int old_potential2;
int game = 1;
int collision = 0;
int p1_move_collision = 0;
int p2_move_collision = 0;
int moved = 0;
while(1) {
```

```
                        /*Check if paused button pushed*/
                        if( (count_player1 >= PLAYER1_SLEEP_CYCLES &&
(check_paused(pressed_player1) != 0) ) ||
                                        (count_player2 >= PLAYER2_SLEEP_CYCLES &&
(check_paused(pressed_player2) != 0) ) ){
                                pressed_player1 = 0;
                                pressed_player2 = 0;
                                count_player1 = 0;
                                count_player2 = 0;
                                paused = !paused;
                                //printf("Paused is now %d\n", paused);
                        }

                        /* Move player 1 */
                        if(count_player1 >= PLAYER1_SLEEP_CYCLES && !paused ){
                                p1_move_collision = movement(code, snake_player1, player1_dir,
pressed_player1, PLAYER1, &info1);
                                if(p1_move_collision)
                                        game_winner = 2;
                                check_powerup_col(snake_player1, snake_player2, player1_dir,
PLAYER1, &info1);
                                //movement(code, snake_player2, player2_dir, food,
pressed_player1, PLAYER2);
                                check_powerup_buttons(pressed_player1, snake_player1,
snake_player2, freeze, PLAYER1, &info1);
                                count_player1 = 0;
                                pressed_player1 = 0;
                                moved = 1;
                                //printf("Moving player1");
                        }

                        /* Move player 2 */
                        if(count_player2 >= PLAYER2_SLEEP_CYCLES && !paused){
                                p2_move_collision = movement(code, snake_player2, player2_dir,
pressed_player2, PLAYER2, &info2);
                                if(p2_move_collision)
                                        game_winner = 1;
                                check_powerup_col(snake_player2, snake_player1, player2_dir,
PLAYER2, &info2);
```

```
                        check_powerup_buttons(pressed_player2, snake_player2,
snake_player1,freeze, PLAYER2, &info2);
                        count_player2 = 0;
                        pressed_player2 = 0;
                        moved = 1;
                }

                /* Get controls from player1 everytime and store control for later if
pressed */
                old_potential1 = potential_pressed1;
                potential_pressed1 = getPlayer1Controller();
                if( potential_pressed1 != 0 && old_potential1 != potential_pressed1){
                        pressed_player1 = potential_pressed1;
                        //printf("pressed player1: %d\n", pressed_player1);
                }

                /* Get controls from player2 everytime and store control for later if
pressed */
                old_potential2 = potential_pressed2;
                potential_pressed2 = getPlayer2Controller();
                if( potential_pressed2 != 0 && old_potential2 != potential_pressed2){
                        pressed_player2 = potential_pressed2;
                }


                if( moved ){
                        collision = snakeCol(snake_player1, snake_player2);
                        play_move_sound();
                }

                if (collision || p1_move_collision || p2_move_collision){
                        //printf("breaking");
                        play_gameover_sound();
                        play_sound();
                        draw_winner(game_winner);
                        usleep(SLEEP_TIME*250000);
                        //wait_for_continue();
                        break;
                }
```

```
                play_sound();

                //printf("Random: %d", PRNG(40));
                moved = 0;
                count_player1++;
                count_player2++;
                usleep(SLEEP_TIME*1000);
            }
            //printf("New Game starting\n");


        }

        //printf("GAME OVER\n");

        return 0;
```

```
*********************************************************************************************
snake_io.h

#ifndef _SNAKE_IO_H_
#define _SNAKE_IO_H_

#include "../snake_software_bsp/system.h"

#define WRITE_SPRITE(select,data) \
IOWR_32DIRECT(DE2_VGA_CONTROLLER_0_BASE, select * 4, data)

#define READ_SNAKE() \
IORD_32DIRECT(DE2_VGA_CONTROLLER_0_BASE, 0 * 4)
//
//#define READ_SNAKE1_TAIL() \
//IORD_32DIRECT(DE2_VGA_CONTROLLER_0_BASE, 1 * 4)
//
//#define READ_SNAKE1_LENGTH() \
//IORD_32DIRECT(DE2_VGA_CONTROLLER_0_BASE, 3 * 4)

#define SOFT_RESET() \
IOWR_32DIRECT(DE2_VGA_CONTROLLER_0_BASE, 3 * 4, 0);
```

```
#define ENABLE_SPLASH_SCREEN() \
IOWR_32DIRECT(DE2_VGA_CONTROLLER_0_BASE, 4 * 4, 0);

#define DISABLE_SPLASH_SCREEN() \
IOWR_32DIRECT(DE2_VGA_CONTROLLER_0_BASE, 5 * 4, 0);

#define READ_PLAYER_CONTROLLER(player) \
IORD_32DIRECT(NES_CONTROLLER_BASE, player * 4);

#define PLAY_SOUND(sound_id) \
IORD_32DIRECT(DE2_AUDIO_CONTROLLER_0_BASE, sound_id * 4);

#define CHANGE_DIVIDER(divider) \
IOWR_32DIRECT(DE2_AUDIO_CONTROLLER_0_BASE, 0, divider);



/* Player/Tile/Address codes */
const char SNAKE_ADDR        = 1;
const char TILES_ADDR        = 2;


char PLAYER1   = 1;
char PLAYER2   = 2;


/* Snake sprite codes */
const char SNAKE_HEAD_RIGHT         = 1;
const char SNAKE_HEAD_LEFT                  = 2;
const char SNAKE_HEAD_UP                    = 3;
const char SNAKE_HEAD_DOWN                  = 4;
const char SNAKE_TAIL_RIGHT                 = 5;
const char SNAKE_TAIL_LEFT                  = 6;
const char SNAKE_TAIL_UP                    = 7;
const char SNAKE_TAIL_DOWN                  = 8;
const char SNAKE_BODY_RIGHT                 = 9;
const char SNAKE_BODY_LEFT                  = 10;
const char SNAKE_BODY_UP                    = 11;
```

```
const char SNAKE_BODY_DOWN                = 12;
const char SNAKE_TURN_UP_RIGHT            = 13;
const char SNAKE_TURN_RIGHT_DOWN    = 14;
const char SNAKE_TURN_DOWN_LEFT           = 15;
const char SNAKE_TURN_LEFT_UP       = 16;

/* Tile sprite codes */
const char RABBIT_CODE                          = 1;
const char MOUSE_CODE                     = 2;
const char EDWARDS_CODE                   = 3;
const char WALL_CODE                      = 4;
const char SPEED_CODE                           = 5;
const char FREEZE_CODE                          = 6;
const char GROWTH_CODE                          = 7;
const char ONE_CODE                             = 8;
const char TWO_CODE                             = 9;
const char P_CODE                               = 10;
const char W_CODE                               = 11;
const char I_CODE                               = 12;
const char N_CODE                               = 13;
const char S_CODE                               = 14;
const char EXC_CODE                             = 15;
const char PLAY_CODE                      = 16;
const char PAUSE_CODE                     = 17;
const char SPLASH_SNAKE_CODE        = 18;
const char T_CODE                               = 19;
const char E_CODE                               = 20;

/* Segment codes */
const char SEG_HEAD                             = 0;
const char SEG_SECOND_HEAD                = 1;
const char SEG_SECOND_TAIL                = 2;
const char SEG_TAIL                             = 3;

/* Add/Remove codes */
const char REMOVE_CODE                          = 0;
const char ADD_CODE                             = 1;

/* Button codes */
const int RIGHT_CODE                      =     (0x00000001);
```

```
const int LEFT_CODE                         =       (0x00000002);
const int DOWN_CODE                         =       (0x00000004);
const int UP_CODE                                   =       (0x00000008);
const int START_CODE                        =       (0x00000010);
const int SELECT_CODE                               =       (0x00000020);
const int B_CODE                                    =       (0x00000040);
const int A_CODE                                    =       (0x00000080);



/*
*       Player : 1 = player1, 2 = player2d
*       Sprite: Choose sprite constant
* NOTE: The snake_x and snake_y are NOT x and y coordinatees.
*               They are the tile numbers (X:0-40, Y:0-30).
*               Please keep track of their actual x,y locations in software
*/
void inline addSnakePiece(int player, char sprite, short snake_x, short snake_y){
        char unused = 0;
        char add_remove = ADD_CODE;
        player = player - 1; // For controller, it expects 0 or 1
        int code = (unused << 27) | (player << 26) | (add_remove << 25) | (sprite << 20) |
(unused << 11) | ((snake_x & 0x03F) << 5) | (snake_y & 0x01F);
        WRITE_SPRITE(SNAKE_ADDR,code);
}

void inline removeSnakePiece(int player, short tile_x, short tile_y){
        char unused = 0;
        char add_remove = REMOVE_CODE;
        char sprite = 0; // Dont care
        player = player - 1; // For controller, it expects 0 or 1
        int code = (unused << 27) | (player << 26) | (add_remove << 25) | (sprite << 20) |
(unused << 11) | ((tile_x & 0x03F) << 5) | (tile_y & 0x01F);
        WRITE_SPRITE(SNAKE_ADDR,code);
}



/*
* NOTE: The tile_x and tile_y are NOT x and y coordinatees.
*               They are the tile numbers (X:0-40, Y:0-30).
*               Please keep track of their actual x,y locations in software
```

```
*/
void inline addTilePiece(char sprite, short tile_x, short tile_y){
	char unused = 0;
	char segment = 0; // Dont care
	char add_remove = ADD_CODE;
	int code = (unused << 28) | (segment << 26) | (add_remove << 25) | (sprite << 20) |
(unused << 11) | ((tile_x & 0x03F) << 5) | (tile_y & 0x01F);;
	WRITE_SPRITE(TILES_ADDR,code);
}

void inline removeTilePiece(short tile_x, short tile_y){
	char unused = 0;
	char segment = 0; // Dont care
	char add_remove = REMOVE_CODE;
	char sprite = 0; // Dont care
	int code = (unused << 28) | (segment << 26) | (add_remove << 25) | (sprite << 20) |
(unused << 11) | ((tile_x & 0x03F) << 5) | (tile_y & 0x01F);
	WRITE_SPRITE(TILES_ADDR,code);
}

int inline getController(player){

	int controls = READ_PLAYER_CONTROLLER(player);
//	int right = controls  & (0x00000001);
//	int left = controls   & (0x00000002);
//	int down = controls   & (0x00000004);
//	int up = controls             & (0x00000008);
//	int start = controls  & (0x00000010);
//	int select = controls         & (0x00000020);
//	int b = controls              & (0x00000040);
//	int a = controls              & (0x00000080);
//	printf("%d-%d-%d-%d-%d-%d-%d-%d\n",a,b,select,start,up,down,left,right);
//	if( right )
//		return 0;
//	if( left )
//		return 1;
//	if( up )
//		return 2;
//	if( down )
//		return 3;
```

```
	return controls;
}

int inline getPlayer1Controller(){

	int c = getController(PLAYER1);
	return c;
}

int inline getPlayer2Controller(){

	int c = getController(PLAYER2);
	return c;
}

int get_dir_from_pressed(int pressed){
	int right	= pressed	& RIGHT_CODE;
	int left	= pressed	& LEFT_CODE;
	int down	= pressed	& DOWN_CODE;
	int up		= pressed	& UP_CODE;
	if( right )
		return 0;
	if( left )
		return 1;
	if( up )
		return 2;
	if( down )
		return 3;

	return -1;
}

int get_button_from_pressed(int pressed){
	int a		= pressed	& A_CODE;
	int b		= pressed	& B_CODE;
	if( a )
		return A_CODE;
	if( b )
```

```
                return B_CODE;
        return -1;
}

int check_paused(int pressed){

        return pressed & SELECT_CODE;
}

void enable_splash_screen(){

        ENABLE_SPLASH_SCREEN();
}

void disable_splash_screen(){

        DISABLE_SPLASH_SCREEN();
}

void reset_hardware(){
        int x = 0;
        int y = 0;
        for(x = 0; x < 40; x++){
                for(y = 0; y < 50; y++){
                        addSnakePiece(0, 0, x, y);
                        addTilePiece(0, x, y);
                }
        }
        int zero = 0;
        int code = (zero << 28) | ((50 & 0x03F) << 5) | (50 & 0x01F);;
        WRITE_SPRITE(TILES_ADDR,code);
        SOFT_RESET();
}


#endif
```

**********

speed.h

```
#ifndef _SPEED_H_
#define _SPEED_H_
#include "snake_io.h"
#include "powboard.h"
#include "constants.h"
#include "audio.h"

void initSpeed(){
	//printf("Initializing speed\n");
	int i;
	int j;
	int count = 0;
	for(i = 0; i < X_LEN; i++){
		for(j = 0; j < Y_LEN; j++){
			if(board[i][j] == 2){
				speed[count].enable = 0;
				speed[count].xCoord = i;
				speed[count].yCoord = j;
				count++;
			}
		}
	}

	shuffle_speed(MAX_POWERUP_SIZE);
}

int checkSpeed(struct Snake snake[], struct Snake other_snake[], int player, struct SnakeInfo
* info){
	int j = speed_index - 1;
	//for(j = 0; j < MAX_POWERUP_SIZE; j++){
		if(speed[j].enable){
			//int xDiff = abs(snake[0].xCoord - speed[j].xCoord*16);
			//int yDiff = abs(snake[0].yCoord - speed[j].yCoord*16);
			//printf("snake x: %d y: %d\n",snake[0].xCoord, snake[0].yCoord);
			//printf("food x: %d y: %d\n",food[j].xCoord, food[j].yCoord);
			if(snake[0].xCoord == speed[j].xCoord && snake[0].yCoord ==
speed[j].yCoord){//if(xDiff <= col_offset && yDiff <= col_offset){
				//printf("Eating Speed!\n");
```

```
                        play_powerup_sound();
                        removeSpeed(j);
                        info->speed_enabled = 1;
                        info->speed_count = 0;
                        //addEnd(snake, dir, player, info);
                        if(player == PLAYER1){
                                PLAYER1_SLEEP_CYCLES  = SPEED_SLEEP_CYCLE / SLEEP_TIME;
                        }else{
                                PLAYER2_SLEEP_CYCLES  = SPEED_SLEEP_CYCLE / SLEEP_TIME;
                        }
                        //                          if(speed_index == MAX_POWERUP_SIZE){
                        //                                  speed_index = 0;
                        //                          }
                        //while( !drawSpeed(speed_index++) );
                        //break;
                }
        }
//}
if(speed_pow_count == 250 && !speed_drawn){
        int i;
        for(i = 0 ; i < 50; i++){
                if(drawSpeed(snake, other_snake)){
                        break;
                }
        }
        //speed_drawn = 1;
        speed_pow_count  = 0;
}
if(speed_pow_count == 300){
        speed_pow_count  = 0;
}
speed_pow_count++;
//printf("speed_count: %d\n", info->speed_count);
if(info->speed_enabled){
        info->speed_count++;
        if(info->speed_count >=  SPEED_TIME){
                //printf("reseting speed\n");
                info->speed_enabled = 0;
                info->speed_count = 0;
                if(player == PLAYER1){
                        PLAYER1_SLEEP_CYCLES  = DEFAULT_SLEEP_CYCLE / SLEEP_TIME;
```

```
                        }else{
                                PLAYER2_SLEEP_CYCLES  = DEFAULT_SLEEP_CYCLE / SLEEP_TIME;
                        }
                }
        }
        return 0;
}

int drawSpeed(struct Snake snake[], struct Snake other_snake[]){
        if(speed_index == MAX_POWERUP_SIZE){
                speed_index = 0;
        }
        if((speed[speed_index].xCoord <= 2 || speed[speed_index].xCoord >= X_LEN-1)
                        || (speed[speed_index].yCoord <= 2 || speed[speed_index].yCoord >=
Y_LEN-1)){
                speed_index++;
                return 0;
        }

        short t_xCoord = speed[speed_index].xCoord;
        short t_yCoord = speed[speed_index].yCoord;
        int i;
        for(i = 0; i < SNAKE_SIZE; i++){
                if(!(snake[i].enable || other_snake[i].enable)){
                        break;
                }
                if(snake[i].enable){
                        if(snake[i].xCoord == t_xCoord && snake[i].yCoord == t_yCoord){
                                speed_index++;
                                return 0;
                        }
                }
                if(other_snake[i].enable){
                        if(other_snake[i].xCoord == t_xCoord && other_snake[i].yCoord ==
t_yCoord){
                                speed_index++;
                                return 0;
                        }
                }
        }
```

```
        if(brick_tiles[t_xCoord][t_yCoord]){
                speed_index++;
                return 0;
        }

        if(freeze[speed_index].enable || food[speed_index].enable ||
edwards[speed_index].enable){
                speed_index++;
                return 0;
        }

        speed[speed_index].enable = 1;
        speed_drawn = 1;
        addTilePiece(SPEED_CODE,  speed[speed_index].xCoord,  speed[speed_index].yCoord);
        speed_index++;
        return 1;
}

void removeSpeed( int index){
        //printf("Removing speed\n");
        speed[index].enable = 0;
        speed_drawn = 0;
        removeTilePiece( speed[index].xCoord,  speed[index].yCoord);
}

void shuffle_speed(int n){
        int i;
        for(i = 0; i < n; i++){
                int index = PRNG(n);
                struct Speed temp = speed[index];
                speed[index] = speed[i];
                speed[i] = temp;
        }
}
#endif
```

**********

powboard.h

```
#ifndef _POWBOARD_H_
#define _POWBOARD_H_
#include "constants.h"
int x[X_LEN];
int y[Y_LEN];

int PRNG(int n){
	n_seed = (8253729 * n_seed + 2396403);
	return n_seed % n;
}

void shuffle(int arr[], int n){
	int i;
	for(i = 0; i < n; i++){
		int index = PRNG(n);
		int temp = arr[index];
		arr[index] = arr[i];
		arr[i] = temp;
	}
}

void shuffle2d(int board[X_LEN][Y_LEN]){
	int arr[1200];
	int i;
	int j;
	int count = 0;
	for(i = 0; i < X_LEN; i++){
		for(j = 0; j < Y_LEN; j++){
			arr[count] = board[i][j];
			count++;
		}
	}
	shuffle(arr, 1200);
	count = 0;
	for(i = 0; i < X_LEN; i++){
		for(j = 0; j < Y_LEN; j++){
			board[i][j]= arr[count];
			count++;
		}
	}
```

```
}

void initPowBoard(int board[X_LEN][Y_LEN], unsigned int seed){
	int i;
	n_seed = seed;
	for (i = 0; i < X_LEN*4; i++){
		x[i] = i;
	}
	shuffle(x, X_LEN);
	for (i = 0; i < Y_LEN*4; i++){
		y[i] = i;
	}
	shuffle(y, Y_LEN);

	int count = 0;
	i = 0;
	int k;
	int j;
	for(k = 0; k < X_LEN; k++){
		for(j = 0; j < Y_LEN; j++){
			if(i == 4){
				i = 0;
			}
			//printf("%d type: %d at loc: %d,%d\n", count, i, x[k], y[j]);
			board[x[k]][y[j]] = i;
			count++;
			i++;
		}
	}
	shuffle2d(board);
}

#endif

*****************************************************************************************
llist.h

#ifndef _LLIST_H_
#define _LLIST_H_
```

```
#include "snake_io.h"
#include "constants.h"


int offset = 16;

void initSnake(struct Snake snake[], int xCoord, int yCoord, int player, struct SnakeInfo *
info){
	info->head = 0;
	info->tail = 2;
	snake[0].xCoord = xCoord;//16;
	snake[0].yCoord = yCoord;//8;
	snake[0].enable = 1;
	//printf("IN INIT xCoord: %d yCoord: %d\n", xCoord, yCoord);
	////printf("IN INIT x: %d y: %d\n",snake[0].xCoord, snake[0].yCoord);
	snake[1].xCoord = snake[0].xCoord - 1; //hardcoded to move right -16
	snake[1].yCoord = snake[0].yCoord;//8;
	snake[1].enable = 1;
	snake[info->tail].xCoord = snake[1].xCoord - 1; //hardcoded to move right - 16
	snake[info->tail].yCoord = snake[1].yCoord;//8;
	snake[info->tail].enable = 1;
	int i;
	for(i = info->tail + 1; i < SNAKE_SIZE; i++){
		snake[i].enable = 0;
	}
	info->has_edwards = 0;
	info->has_freeze = 0;
	info->freeze_enabled = 0;
	info->speed_enabled = 0;
	/* originally had x and ys divided by 16 */
	addSnakePiece(player, SNAKE_HEAD_RIGHT, snake[0].xCoord,  snake[0].yCoord);
	addSnakePiece(player, SNAKE_BODY_RIGHT, snake[1].xCoord, snake[1].yCoord);
	addSnakePiece(player, SNAKE_TAIL_RIGHT, snake[info->tail].xCoord, snake[info-
>tail].yCoord);

	//print snake - for testing
	/*for(i = 0; i < SNAKE_SIZE; i++){
		if(snake[i].enable == 1){
			//printf("snake part at x:%d, y%d\n", snake[i].xCoord, snake[i].yCoord);
		}
```

```
	}*/
}

int abs(int n)
{
	if (n < 0)
		n = -n;

	return n;
}

/*
* check collision between head and other parts of body
*/
int traverseList(struct Snake snake[], int player)
{
	int count = 1;
	while(snake[count].enable)
	{
		//int xDiff = abs(snake[0].xCoord - snake[count].xCoord);
		//int yDiff = abs(snake[0].yCoord - snake[count].yCoord);
		//printf("traverse x:%d y:%d\n", snake[count].xCoord, snake[count].yCoord);
		//printf("diff xdiff:%d ydiff:%d\n", xDiff, yDiff);
		//printf("traverse x:%d y:%d\n", snake[0].xCoord, snake[0].yCoord);
		if(snake[0].xCoord == snake[count].xCoord && snake[0].yCoord ==
snake[count].yCoord){//if(xDiff <= col_offset && yDiff <= col_offset){
			//printf("self Collision!");
			if(player == PLAYER1){
				/* snake1 collided with itself so snake2 wins */
				game_winner = 2;
			}else{
				/* snake2 collided with itself so snake1 wins */
				game_winner = 1;
			}
			////printf("traverse x:%d y:%d\n", snake[count].xCoord,
snake[count].yCoord);
			return 1;
		}
		count++;
	}
```

```
	return 0;
}

int brickCol(struct Snake snake[], int player){
	short x = snake[0].xCoord;///16;
	short y = snake[0].yCoord;///16;
	//printf("brick enable: %d", brick_tiles[x][y]);
	if(brick_tiles[x][y]){
		//printf("brick col");
		if(player == PLAYER1){
			/* snake1 collided with itself so snake2 wins */
			game_winner = 2;
		}else{
			/* snake2 collided with itself so snake1 wins */
			game_winner = 1;
		}
		return 1;
	}
	return 0;
}

int snakeCol(struct Snake snake1[], struct Snake snake2[]){
	//	int i;
	//	int j;
	//	for(i = 0; i < SNAKE_SIZE; i++){
	//		if(!snake1[i].enable)
	//			break;
	//		for(j = 0; j < SNAKE_SIZE; j++){
	//			if(!snake2[j].enable)
	//				break;
	//			//int xDiff = abs(snake1[i].xCoord - snake2[j].xCoord);
	//			//int yDiff = abs(snake1[i].yCoord - snake2[j].yCoord);
	//			if(snake1[i].xCoord == snake2[j].xCoord && snake1[i].yCoord ==
snake2[j].yCoord){//if(xDiff < offset && yDiff < offset){
	//				//printf("2 snake Collision!");
	//				if(i == 0 && j == 0){
	//					/* head on collision, draw */
	//					game_winner = 3;
	//				}else if(i == 0 && j != 0){
	//					/* snake1 head collided with snake2's body*/
```

```
	//				game_winner = 2;
	//			}else if(i != 0 && j == 0){
	//				/* snake2 head collided with snake1's body*/
	//				game_winner = 1;
	//			}
	//			return 1;
	//		}
	//	}
	//	return 0;
	if(snake1[0].xCoord == snake2[0].xCoord && snake1[0].yCoord == snake2[0].yCoord){
		/* head collided draw*/
		game_winner = 3;
		return 1;
	}
	int i;
	for(i = 1; i < SNAKE_SIZE; i++){
		if( !snake1[i].enable && !snake2[i].enable ){
			break;
		}

		if(snake2[i].enable && snake1[0].xCoord == snake2[i].xCoord && snake1[0].yCoord
== snake2[i].yCoord){
			/* snake1 collided with snake2 body */
			game_winner = 2;
			return 1;
		}

		if(snake1[i].enable && snake2[0].xCoord == snake1[i].xCoord && snake2[0].yCoord
== snake1[i].yCoord){
			/* snake2 collided with snake1 body */
			game_winner = 1;
			return 1;
		}
	}
	return 0;
}

void addEnd(struct Snake snake[], int dir, int player, struct SnakeInfo * info)
{
```

```
	//printf("Adding to the end of the snake\n");
	//printf("original tail at x:%d, y%d\n", snake[tail].xCoord, snake[tail].yCoord);

	info->tail = info->tail + 1;
	snake[info->tail].enable = 1;

	if(dir == 0){//left
		snake[info->tail].xCoord = snake[info->tail-1].xCoord + 1;//offset;
		snake[info->tail].yCoord = snake[info->tail-1].yCoord;
	}else if(dir == 1){//right
		snake[info->tail].xCoord = snake[info->tail-1].xCoord - 1;//offset;
		snake[info->tail].yCoord = snake[info->tail-1].yCoord;
	}else if(dir == 2){//up
		snake[info->tail].xCoord = snake[info->tail-1].xCoord;
		snake[info->tail].yCoord = snake[info->tail-1].yCoord + 1;//offset;
	}else if(dir == 3){//down
		snake[info->tail].xCoord = snake[info->tail-1].xCoord;
		snake[info->tail].yCoord = snake[info->tail-1].yCoord - 1;//offset;
	}

}

void updateBody(struct Snake snake[], struct SnakeInfo *info){
	int i;
	for(i = info->tail; i >= 1; i--){
		if(snake[i].enable==1){
			//printf("snake before '=' at x:%d, y%d\n", snake[i].xCoord,
snake[i].yCoord);
			snake[i] = snake[i-1];
			//printf("snake after '=' at x:%d, y%d\n", snake[i].xCoord,
snake[i].yCoord);
		}
	}
}

void updateSnake(struct Snake snake[], int xCoord, int yCoord, int dir, int old_dir,
		int player, struct SnakeInfo * info){
	//printf("Updating snake\n");

	// Remove tail first
```

```
        removeSnakePiece(player,  snake[info->tail].xCoord,  snake[info->tail].yCoord);

        int tail_old_x = snake[info->tail-1].xCoord;
        int tail_old_y = snake[info->tail-1].yCoord;

        //Then update snake
        updateBody(snake, info);
        snake[0].xCoord = xCoord;
        snake[0].yCoord = yCoord;

        int tail_new_x = snake[info->tail-1].xCoord;
        int tail_new_y = snake[info->tail-1].yCoord;



        int tail_sprite = -1;
        if( tail_old_x != tail_new_x){
                if( tail_old_y > tail_new_y){
                        //turned up
                        tail_sprite = SNAKE_TAIL_UP;
                } else if( tail_old_y < tail_new_y ){
                        //turned down
                        tail_sprite = SNAKE_TAIL_DOWN;
                } else {
                        //same direction, which way?
                        tail_sprite = (tail_old_x < tail_new_x) ? SNAKE_TAIL_RIGHT:
SNAKE_TAIL_LEFT;
                }
        }
        if( tail_old_y != tail_new_y){
                if( tail_old_x > tail_new_x){
                        //turned left
                        tail_sprite = SNAKE_TAIL_LEFT;
                } else if( tail_old_x < tail_new_x ){
                        //turned down
                        tail_sprite = SNAKE_TAIL_RIGHT;
                } else {
                        //same direction, which way?
                        tail_sprite = (tail_old_y < tail_new_y) ? SNAKE_TAIL_DOWN:
SNAKE_TAIL_UP;
                }
```

```
        }

        // Then do new hardware display
        writeToHW(snake, dir, old_dir, player, xCoord, yCoord, info, tail_sprite);


}


#endif /* #ifndef _LIST_H_ */


*****************************************************************************************
freeze.h

#ifndef _FREEZE_H_
#define _FREEZE_H_

#include "snake_io.h"
#include "powboard.h"
#include "constants.h"
#include "audio.h"



void initFreeze(){
        //printf("Initializing freeze\n");
        int i;
        int j;
        int count = 0;
        for(i = 0; i < X_LEN; i++){
                for(j = 0; j < Y_LEN; j++){
                        if(board[i][j] == 1){
                                freeze[count].enable = 0;
                                freeze[count].xCoord = i;
                                freeze[count].yCoord = j;
                                count++;
                        }
                }
```

```
	}

	shuffle_freeze(freeze,MAX_POWERUP_SIZE);
}

int checkFreeze(struct Snake snake[], struct Snake other_snake[], int player, struct SnakeInfo
* info){

	int j = freeze_index - 1;
	//for(j = 0; j < MAX_POWERUP_SIZE; j++){
		if(freeze[j].enable){
			//int xDiff = abs(snake[0].xCoord - freeze[j].xCoord*16);
			//int yDiff = abs(snake[0].yCoord - freeze[j].yCoord*16);
			////printf("snake x: %d y: %d\n",snake[0].xCoord, snake[0].yCoord);
			if(snake[0].xCoord == freeze[j].xCoord && snake[0].yCoord ==
freeze[j].yCoord){//if(xDiff <= col_offset && yDiff <= col_offset){
				//printf("Eating Freeze!\n");
				play_powerup_sound();
				removeFreeze(freeze,j);
				info->has_freeze = 1;
				//				if(freeze_index ==
MAX_POWERUP_SIZE){
				//					freeze_index = 0;
				//				}
				//while( !drawFreeze(freeze) );
				//break;
			}
		}
	//}
	////printf("count: %d drawn:%d", freeze_pow_count, freeze_drawn);
	if(freeze_pow_count == 400 && !freeze_drawn){
		int i;
		for(i = 0 ; i < 50; i++){
			if(drawFreeze(snake, other_snake)){
				break;
			}
		}
		//freeze_drawn = 1;
		freeze_pow_count  = 0;
	}
```

```
        if(freeze_pow_count == 300){
                freeze_pow_count  = 0;
        }
        freeze_pow_count++;
        return 0;
}

void apply_freeze(struct Snake snake[], int player, struct SnakeInfo * info){

        if( !info->has_freeze ){
                //printf("PLAYER%d DOESNT HAVE FREEZE\n", player);
                return;
        }

        //printf("PLAYER%d used his freeze\n",player);
        info->has_freeze = 0;
        info->freeze_enabled = 1;
        info->freeze_count = 0;
        if(player == PLAYER1){
                PLAYER2_SLEEP_CYCLES  = FREEZE_SLEEP_CYCLE / SLEEP_TIME;
        } else{
                PLAYER1_SLEEP_CYCLES  = FREEZE_SLEEP_CYCLE / SLEEP_TIME;
        }

}

int recalc_freeze_times(struct Snake snake[], int player, struct SnakeInfo * info){

        if(info->freeze_enabled){
                info->freeze_count++;
                if(info->freeze_count >=  FREEZE_TIME){
                        info->freeze_enabled = 0;
                        info->freeze_count = 0;
                        if(player == PLAYER1){
                                PLAYER2_SLEEP_CYCLES  = DEFAULT_SLEEP_CYCLE / SLEEP_TIME;
                        }else{
                                PLAYER1_SLEEP_CYCLES  = DEFAULT_SLEEP_CYCLE / SLEEP_TIME;
                        }

                }
```

```
	}
	return 1;
}

int drawFreeze(struct Snake snake[], struct Snake other_snake[]){
	if(freeze_index == MAX_POWERUP_SIZE){
		freeze_index = 0;
	}
	if((freeze[freeze_index].xCoord <= 2 || freeze[freeze_index].xCoord >= X_LEN-1)
			|| (freeze[freeze_index].yCoord <= 2 || freeze[freeze_index].yCoord >=
Y_LEN-1)){
		freeze_index++;
		return 0;
	}

	short t_xCoord = freeze[freeze_index].xCoord;
	short t_yCoord = freeze[freeze_index].yCoord;
	int i;
	for(i = 0; i < SNAKE_SIZE; i++){
		if(!(snake[i].enable || other_snake[i].enable)){
			break;
		}
		if(snake[i].enable){
			if(snake[i].xCoord == t_xCoord && snake[i].yCoord == t_yCoord){
				freeze_index++;
				return 0;
			}
		}
		if(other_snake[i].enable){
			if(other_snake[i].xCoord == t_xCoord && other_snake[i].yCoord ==
t_yCoord){
				freeze_index++;
				return 0;
			}
		}
	}
	if(brick_tiles[t_xCoord][t_yCoord]){
		freeze_index++;
		return 0;
	}
```

```
        if(food[freeze_index].enable || speed[freeze_index].enable ||
edwards[freeze_index].enable){
                freeze_index++;
                return 0;
        }
        freeze_drawn = 1;
        freeze[freeze_index].enable = 1;
        //printf("Freeze ENABLED at x: %d y: %d\n",freeze[freeze_index].xCoord,
freeze[freeze_index].yCoord);
        addTilePiece(FREEZE_CODE,  freeze[freeze_index].xCoord,  freeze[freeze_index].yCoord);

        //printf("Freeze index: %d\n", freeze_index);
        freeze_index++;

        return 1;
}

void removeFreeze(struct Freeze freeze[], int index){
        //printf("Removing freeze\n");
        freeze[index].enable = 0;
        freeze_drawn = 0;
        removeTilePiece( freeze[index].xCoord,  freeze[index].yCoord);
}

void shuffle_freeze(struct Freeze arr[], int n){
        int i;
        for(i = 0; i < n; i++){
                int index = PRNG(n);
                struct Freeze temp = arr[index];
                arr[index] = arr[i];
                arr[i] = temp;
        }
}

#endif
```

**********

food.h

```
#ifndef _FOOD_H_
#define _FOOD_H_
#include "snake_io.h"
#include "powboard.h"
#include "constants.h"
#include "audio.h"


void initFood(){
	//printf("Initializing food\n");
	int t = 0;
	int i;
	int j;
	int count = 0;
	for(i = 0; i < X_LEN; i++){
		for(j = 0; j < Y_LEN; j++){
			if(board[i][j] == 0){
				food[count].enable = 0;
				food[count].type = t;
				food[count].xCoord = i;
				food[count].yCoord = j;
				count++;
				if(t == 0){
					t = 1;
				}
				else {
					t = 0;
				}
			}
		}
	}

	shuffle_food(MAX_POWERUP_SIZE);
}

int checkFood(struct Snake snake[], struct Snake other_snake[], int dir, int player, struct
SnakeInfo * info)
{
	/* if food count == 0 it will try to draw food in the next available location */
```

```
	if(food_count == 0){
		drawFood(snake, other_snake);
	}
	//struct Snake *head = snake[0];
	int j;
	for(j = 0; j < MAX_POWERUP_SIZE; j++){
		if(food[j].enable){
			//int xDiff = abs(snake[0].xCoord - food[j].xCoord*16);
			//int yDiff = abs(snake[0].yCoord - food[j].yCoord*16);
			//////printf("snake x: %d y: %d\n",snake[0].xCoord, snake[0].yCoord);
			//////printf("food x: %d y: %d\n",food[j].xCoord, food[j].yCoord);
			if(snake[0].xCoord == food[j].xCoord && snake[0].yCoord ==
food[j].yCoord){//if(xDiff <= col_offset && yDiff <= col_offset){
				////printf("Eating Food!\n");
				play_powerup_sound();
				removeFood(j);
				addEnd(snake, dir, player, info);
				/*food_index == MAX_POWERUP_SIZE){
					food_index = 0;
				}*/
				//while( !drawFood(snake, other_snake) );
				/* iterate 100 times looking for the next
				 * free element, if not found
				 */
				if(player == PLAYER1){
					player1_food_eaten++;
					if(player1_food_eaten == 5){
						player1_food_eaten = 0;
						PLAYER1_SLEEP_CYCLES -= 1;
					}
				}else{
					if(player2_food_eaten == 5){
						player2_food_eaten = 0;
						PLAYER2_SLEEP_CYCLES -= 1;
					}
				}
				int i;
				for(i = 0 ; i < 50; i++){
					if(drawFood(snake, other_snake)){
						break;
					}
```

```
                        }
                        break;          // Original sleep time/SLEEP_TIME

                }
        }
    }
    return 0;
}

int drawFood(struct Snake snake[], struct Snake other_snake[]){
    if(food_index == MAX_POWERUP_SIZE){
        food_index = 0;
    }
    if((food[food_index].xCoord <= 2 || food[food_index].xCoord >= X_LEN-1)
            || (food[food_index].yCoord <= 2 || food[food_index].yCoord >= Y_LEN-1)
){
        food_index++;
        return 0;
    }
    short f_xCoord = food[food_index].xCoord;
    short f_yCoord = food[food_index].yCoord;
    int i;
    for(i = 0; i < SNAKE_SIZE; i++){
        if(!(snake[i].enable || other_snake[i].enable)){
            break;
        }
        if(snake[i].enable){
            if(snake[i].xCoord == f_xCoord && snake[i].yCoord == f_yCoord){
                food_index++;
                return 0;
            }
        }
        if(other_snake[i].enable){
            if(other_snake[i].xCoord == f_xCoord && other_snake[i].yCoord ==
f_yCoord){
                food_index++;
                return 0;
            }
        }
    }
```

```
	if(brick_tiles[f_xCoord][f_yCoord]){
		food_index++;
		return 0;
	}

	if(freeze[food_index].enable || speed[food_index].enable ||
edwards[food_index].enable){
		food_index++;
		return 0;
	}

	food[food_index].enable = 1;
	if(food[food_index].type){
		addTilePiece(RABBIT_CODE,  food[food_index].xCoord,  food[food_index].yCoord);
	}else{
		addTilePiece(MOUSE_CODE,  food[food_index].xCoord,  food[food_index].yCoord);
	}
	food_index++;
	food_count++;
	return 1;
}

void removeFood(int index){
	////printf("Removing food\n");
	food[index].enable = 0;
	removeTilePiece( food[index].xCoord,  food[index].yCoord);
	food_count--;
}

void shuffle_food(int n){
	int i;
	for(i = 0; i < n; i++){
		int index = PRNG(n);
		struct Food temp = food[index];
		food[index] = food[i];
		food[i] = temp;
	}
}
```

```
#endif


*********************************************************************************************
edwards.h

#ifndef EDWARDS_H_
#define EDWARDS_H_
#include "drop_brick.h"
#include "brick.h"
#include "audio.h"

void initEdwards(){
	initBrickTile();
	//printf("Initializing edwards\n");
	int t = 0;
	int i;
	int j;
	int count = 0;
	/* counters below used to "simulate" probability */
	int bss_count = 0;/* brick/switch snake counter */
	int bc_count = 0; /* border change counter */
	for(i = 0; i < X_LEN; i++){
		for(j = 0; j < Y_LEN; j++){
			if(board[i][j] == 3){
				edwards[count].enable = 0;
				edwards[count].type = t;
				edwards[count].xCoord = i;
				edwards[count].yCoord = j;
				if(bc_count == 20){
					t = 2;
					bc_count = 0;
				}else if(bss_count == 2){
					t = 1;
					bss_count = 0;
				}
				else {
					t = 0;
					bss_count++;
				}
				count++;
```

```
                        bc_count++;
                }
        }
        shuffle_edwards(MAX_POWERUP_SIZE);

}

int drawEdwards(struct Snake snake[], struct Snake other_snake[]){
        if(edwards_index == MAX_POWERUP_SIZE){
                edwards_index = 0;
        }
        if((edwards[edwards_index].xCoord <= 2 || edwards[edwards_index].xCoord >= X_LEN-1)
                || (edwards[edwards_index].yCoord <= 2 || edwards[edwards_index].yCoord
>= Y_LEN-1) ){
                edwards_index++;
                return 0;
        }
        short f_xCoord = edwards[edwards_index].xCoord;
        short f_yCoord = edwards[edwards_index].yCoord;
        int i;
        for(i = 0; i < SNAKE_SIZE; i++){
                if(!(snake[i].enable || other_snake[i].enable)){
                        break;
                }
                if(snake[i].enable){
                        if(snake[i].xCoord == f_xCoord && snake[i].yCoord == f_yCoord){
                                edwards_index++;
                                return 0;
                        }
                }
                if(other_snake[i].enable){
                        if(other_snake[i].xCoord == f_xCoord && other_snake[i].yCoord ==
f_yCoord){
                                edwards_index++;
                                return 0;
                        }
                }
        }
        if(brick_tiles[f_xCoord][f_yCoord]){
```

```
                edwards_index++;
                return 0;
        }

        if(freeze[edwards_index].enable || speed[edwards_index].enable ||
food[edwards_index].enable){
                edwards_index++;
                return 0;
        }

        edwards[edwards_index].enable = 1;
        //if(edwards[edwards_index].type){
        //printf("added edwards at %d", edwards_index);
        addTilePiece(EDWARDS_CODE,  edwards[edwards_index].xCoord,
edwards[edwards_index].yCoord);
        //}else{
        //      addTilePiece(MOUSE_CODE,  edwards[index].xCoord,  edwards[index].yCoord);
        //}
        edwards_index++;
        edwards_drawn = 1;
        return 1;
}

void removeEdwards(int index){
        //printf("Removing edwards\n");
        //edwards[index].enable = 0;
        //removeTilePiece( edwards[index].xCoord,  edwards[index].yCoord);
        edwards[index].enable = 0;
        //printf("removing at %d x:%d, y:%d", index, edwards[index].xCoord,
edwards[index].yCoord);
        removeTilePiece( edwards[index].xCoord,  edwards[index].yCoord);
        edwards_drawn = 0;
}

void shuffle_edwards(int n){
        int i;
        for(i = 0; i < n; i++){
                int index = PRNG(n);
                struct Edwards temp = edwards[index];
                edwards[index] = edwards[i];
```

```
                edwards[i] = temp;
        }
}

int checkEdwards(struct Snake snake[], struct Snake other_snake[], int player, struct
SnakeInfo * info)
{
        int j = edwards_index - 1;
        //for(j = 0; j < MAX_POWERUP_SIZE; j++){
                if(edwards[j].enable){
                        //int xDiff = abs(snake[0].xCoord - edwards[j].xCoord*16);
                        //int yDiff = abs(snake[0].yCoord - edwards[j].yCoord*16);
                        //printf("snake x: %d y: %d\n",snake[0].xCoord, snake[0].yCoord);
                        if(snake[0].xCoord == edwards[j].xCoord && snake[0].yCoord ==
edwards[j].yCoord){//if(xDiff <= col_offset && yDiff <= col_offset){
                                //printf("Eating Edwards!\n");
                                //printf("x:%d, y:%d", edwards[j].xCoord, edwards[j].yCoord);
                                play_powerup_sound();
                                removeEdwards(j);
                                info->has_edwards = 1;
                                //                              if(edwards_index ==
MAX_POWERUP_SIZE){
                                //                                      edwards_index = 0;
                                //                              }
                                //                              while( !drawEdwards(snake,
other_snake) ){
                                //                                      printf("drawing edwards\n");
                                //                              }
                                //break;
                        }
                }
        //}
        if(edwards_pow_count == 300 && !edwards_drawn){
                int i;
                for(i = 0 ; i < 50; i++){
                        if(drawEdwards(snake, other_snake)){
                                break;
                        }
                }
                //edwards_drawn = 1;
                edwards_pow_count  = 0;
```

```
	}
	if(edwards_pow_count == 300){
		edwards_pow_count  = 0;
	}
	edwards_pow_count++;
	return 0;
}

void apply_edwards(struct Snake snake[], struct Snake other_snake[], int player, struct
SnakeInfo * info){

	if( !info->has_edwards ){
		//printf("PLAYER%d DOESNT HAVE EDWARDS\n", player);
		return;
	}

	//printf("PLAYER%d used his edwards\n",player);
	/*
	 * do this because edwards_index is one ahead
	 * since it is incremented in the draw()
	 */
	int remove_at = 0;
	if(edwards_index == 0){
		remove_at = MAX_POWERUP_SIZE-1;
	}else{
		remove_at = edwards_index - 1;
	}



	if(edwards[remove_at].type == 0){
		setBrickTile(snake, info->tail);
	}else if(edwards[remove_at].type == 1){
		if(PLAYER1 == 1 && PLAYER2 == 2){
			PLAYER1 = 2;
			PLAYER2 = 1;
		}else{
			PLAYER1 = 1;
			PLAYER2 = 2;
		}
	}else{
```

```
                updateBorder();
        }
        info->has_edwards = 0;
}

#endif /* EDWARDS_H_ */


*****************************************************************************************
drop_brick.h

#ifndef _DROP_BRICK_H_
#define _DROP_BRICK_H_

#include "snake_io.h"
#include "powboard.h"
#include "constants.h"

void initBrickTile(){
        short i;
        short j;
        for(i = 0; i < X_LEN; i++){
                for(j = 0; j < Y_LEN; j++){
                        brick_tiles[i][j] = 0;
                }
        }

        for(i = 0; i < X_LEN; i++){
                brick_tiles[i][0] = 1;
                brick_tiles[i][Y_LEN-1] = 1;
        }

        for(i = 0; i < Y_LEN; i++){
                brick_tiles[0][i] = 1;
                brick_tiles[X_LEN-1][i] = 1;
        }

}

void setBrickTile(struct Snake snake[], int index){
```

```
        short x_tile = (short)snake[index].xCoord;///16;
        short y_tile = (short)snake[index].yCoord;///16;
        brick_tiles[x_tile][y_tile] = 1;
        addTilePiece(WALL_CODE, x_tile, y_tile);
}
#endif


*********************************************************************************
constants.h

#ifndef _CONSTANTS_H_
#define _CONSTANTS_H_

#define LENGTH        1200
#define col_offset  14
#define left_dir      0
#define right_dir     1
#define up_dir        2
#define down_dir      3
//#define LEFT_BOUND  0
//#define RIGHT_BOUND 40//640
//#define BOT_BOUND   30//480
//#define TOP_BOUND   0

#define X_LEN         40
#define Y_LEN         30
#define MAX_POWERUP_SIZE     300

#define SNAKE_SIZE                          1200
#define SLEEP_TIME                          2       // milliseconds
#define SPEED_TIME                          50
#define FREEZE_TIME                 50
int DEFAULT_SLEEP_CYCLE        =     60;
int SPEED_SLEEP_CYCLE  =     30;
int FREEZE_SLEEP_CYCLE         =     150;

unsigned int n_seed = 5323;
unsigned int seed = 0;
```

```
int PLAYER1_SLEEP_CYCLES;                  // Original sleep time/SLEEP_TIME
int PLAYER2_SLEEP_CYCLES;                  // Original sleep time/SLEEP_TIME

/* index to know which sprite to draw from power up structs */
int food_index                             = 1;
int speed_index                    = 1;
int freeze_index                   = 1;
int edwards_index                  = 1;
int alt_powups                             = 0;

/* keeps track of current power ups spawn locations */
int board[X_LEN][Y_LEN];
/* keeps track of where a brick was placed */
short brick_tiles[X_LEN][Y_LEN];

int freeze_pow_count               = 0;
int freeze_drawn                   = 0;
int speed_pow_count          = 0;
int speed_drawn                    = 0;
int edwards_pow_count              = 0;
int edwards_drawn                  = 0;
int dir_arg                        = 0;
int switch_snakes                  = 0;
int food_count = 0;
int player1_food_eaten = 0;
int player2_food_eaten = 0;



int LEFT_BOUND =0;
int RIGHT_BOUND       =40;
int BOT_BOUND  =30;
int TOP_BOUND  =0;
/* used with edwards power up */
int NEW_BOT = 29;
int NEW_TOP = 0;
int NEW_LEFT = 0;
int NEW_RIGHT = 39;
/*
* 0 -> nobody yet
* 1 -> snake1
```

```
* 2 -> snake2
* 3 -> draw
*/
int game_winner                         = 0;

/* struct defs for snake and power ups */
struct Snake{
        short xCoord;
        short yCoord;
        short enable;
};

struct SnakeInfo{
        short head;
        short tail;
        short speed_count;
        short speed_enabled;
        short freeze_count;
        short freeze_enabled;
        short has_edwards;
        short has_freeze;
};

struct Food{
        short xCoord;
        short yCoord;
        short enable;
        short type;
};

struct Speed{
        short xCoord;
        short yCoord;
        short enable;
};

struct Freeze{
        short xCoord;
        short yCoord;
        short enable;
```

```
};

struct Edwards{
	short xCoord;
	short yCoord;
	short enable;
	short type;
};

/* gloabl structs */
struct Food food[MAX_POWERUP_SIZE];
struct Speed speed[MAX_POWERUP_SIZE];
struct Freeze freeze[MAX_POWERUP_SIZE];
struct Edwards edwards[MAX_POWERUP_SIZE];

#endif /* #ifndef _CONSTANTS_H_ */


*********************************************************************************************
brick.h

#ifndef _BRICK_H_
#define _BRICK_H_

#include "snake_io.h"
#include "powboard.h"

#define MAX    136
#define X_BORDER      40
#define Y_BORDER      30

void initBorder(){
	int x;
	int y;
	for(x = 0; x < X_BORDER; x++){
		addTilePiece(WALL_CODE,    x,    0);
	}
	for(x = 0; x < X_BORDER; x++){
		addTilePiece(WALL_CODE,    x,    29);
	}
```

```
        for(y = 1; y < Y_BORDER; y++){
                addTilePiece(WALL_CODE,   0,   y);
        }
        for(y = 1; y < Y_BORDER; y++){
                addTilePiece(WALL_CODE,   39,   y);
        }
}

void updateBorder(){
        NEW_TOP += 1;
        NEW_BOT -= 1;
        NEW_LEFT += 1;
        NEW_RIGHT -= 1;
        LEFT_BOUND     += 1;
        RIGHT_BOUND    -= 1;
        BOT_BOUND      -= 1;
        TOP_BOUND      += 1;

        int x;
        int y;
        for(x = 0; x < X_BORDER; x++){
                brick_tiles[x][NEW_TOP] = 1;
                checkOverlap(x, NEW_TOP);
                addTilePiece(WALL_CODE,   x,   NEW_TOP);
        }
        for(x = 0; x < X_BORDER; x++){
                brick_tiles[x][NEW_BOT] = 1;
                checkOverlap(x, NEW_BOT);
                addTilePiece(WALL_CODE,   x,   NEW_BOT);
        }
        for(y = 1; y < Y_BORDER; y++){
                brick_tiles[NEW_LEFT][y] = 1;
                checkOverlap(NEW_LEFT, y);
                addTilePiece(WALL_CODE,   NEW_LEFT,   y);
        }
        for(y = 1; y < Y_BORDER; y++){
                brick_tiles[NEW_RIGHT][y] = 1;
                checkOverlap(NEW_RIGHT, y);
                addTilePiece(WALL_CODE,   NEW_RIGHT,   y);
        }
```

```
}

void checkOverlap(int x, int y){
	if(food[food_index-1].enable && food[food_index-1].xCoord == x && food[food_index-1].yCoord == y){
		food[food_index-1].enable = 0;
		food_count--;
		removeTilePiece((short)x, (short)y);
	}else if(speed[speed_index-1].enable && speed[speed_index-1].xCoord == x && speed[speed_index-1].yCoord == y){
		speed[speed_index-1].enable = 0;
		removeTilePiece((short)x, (short)y);
		speed_drawn = 0;
	}else if(freeze[freeze_index-1].enable && freeze[freeze_index-1].xCoord == x && freeze[freeze_index-1].yCoord == y){
		freeze[freeze_index-1].enable = 0;
		removeTilePiece((short)x, (short)y);
		freeze_drawn = 0;
	}if(edwards[edwards_index-1].enable && edwards[edwards_index-1].xCoord == x && edwards[edwards_index-1].yCoord == y){
		edwards[edwards_index-1].enable = 0;
		removeTilePiece((short)x, (short)y);
		edwards_drawn = 0;
	}
}

#endif

*******************************************************************************************
audio.h

#ifndef _AUDIO_H_
#define _AUDIO_H_

int sound_to_be_played		= -1;
int sound_divider		= -1;

#include "../snake_software_bsp/system.h"
```

```
#define PLAY_SOUND(sound_id) \
IORD_32DIRECT(DE2_AUDIO_CONTROLLER_0_BASE, sound_id * 4);

#define CHANGE_DIVIDER(divider) \
IOWR_32DIRECT(DE2_AUDIO_CONTROLLER_0_BASE, 0, divider);

const int move_sound_code = 0;
const int move_sound_divider = 3125;

const int powerup_sound_code = 1;
const int powerup_sound_divider = 3125;

const int gameover_sound_code = 2;
const int gameover_sound_divider = 3125;

const int splash_sound_code = 3;
const int splash_sound_divider = 3125;

void inline play_move_sound() {
	if (sound_to_be_played == -1) {
		sound_to_be_played = move_sound_code;
		sound_divider = move_sound_divider;
	}
}

void inline play_powerup_sound() {
	sound_to_be_played = powerup_sound_code;
	sound_divider = powerup_sound_divider;
}

void inline play_gameover_sound() {
	sound_to_be_played = gameover_sound_code;
	sound_divider = gameover_sound_divider;
}

void inline play_splash_sound() {
	sound_to_be_played = splash_sound_code;
	sound_divider = splash_sound_divider;
}
```

```
void play_sound(){

	if( sound_to_be_played == -1)
		return;

	PLAY_SOUND(sound_to_be_played);
	PLAY_SOUND(sound_to_be_played);

	sound_to_be_played = -1;
}

#endif



*********************************************************************************************
snakeplus.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity snakeplus is
	port (
		signal CLOCK_50 : in std_logic;
		signal LEDR : out std_logic_vector(17 downto 0);
		SRAM_DQ : inout std_logic_vector(15 downto 0);
		SRAM_ADDR : out std_logic_vector(17 downto 0);
		SRAM_UB_N,
		SRAM_LB_N,
		SRAM_WE_N,
		SRAM_CE_N,
		SRAM_OE_N : out std_logic;

		-- GPIO
		GPIO_0,                                    -- GPIO Connection 0
		GPIO_1 : inout std_logic_vector(35 downto 0); -- GPIO Connection 1
```

```
        -- PS/2 port

        PS2_DAT,                    -- Data
        PS2_CLK : in std_logic;     -- Clock

        SW    : in std_logic_vector(17 downto 0);

        -- VGA output

        VGA_CLK,                                        -- Clock
        VGA_HS,                                         -- H_SYNC
        VGA_VS,                                         -- V_SYNC
        VGA_BLANK,                                      -- BLANK
        VGA_SYNC : out std_logic;                       -- SYNC
        VGA_R,                                          -- Red[9:0]
        VGA_G,                                          -- Green[9:0]
        VGA_B : out std_logic_vector(9 downto 0);       -- Blue[9:0]


        -- Audio Signals --

        AUD_ADCAT : in std_logic;
    -- Audio Codec ADC Data
        AUD_BCLK : inout std_logic;                                                 --
Audio Codec Bit-Stream Clock
        AUD_ADCLRCK,
        -- Audio Codec ADC LR Clock
        AUD_DACDAT,
            -- Audio Codec DAC Data
        AUD_DACLRCK,
        -- Audio Codec DAC LR Clock
        AUD_XCK : out std_logic;
    -- Chip Clock

        I2C_SCLK : out std_logic;
        I2C_SDAT : inout std_logic;
        iCLK,
        iRST_N : in std_logic
);
```

```
end snakeplus;


architecture rtl of snakeplus is
	signal counter : unsigned(15 downto 0);
	signal reset_n : std_logic;
	signal was_reset: std_logic := '0';
begin
		LEDR(17) <= '1';
		LEDR(16) <= '1';

		process (CLOCK_50)
		begin
			if rising_edge(CLOCK_50) then
			if was_reset = '0' then
				reset_n <= '0';
				was_reset <= '1';
			else
				reset_n <= '1';
			end if;
		end if;
		end process;









		nios : entity work.snake_system port map (
			clk_0 => CLOCK_50,
			reset_n => reset_n,


			SRAM_ADDR_from_the_sram => SRAM_ADDR,
			SRAM_CE_N_from_the_sram => SRAM_CE_N,
			SRAM_DQ_to_and_from_the_sram => SRAM_DQ,
			SRAM_LB_N_from_the_sram => SRAM_LB_N,
			SRAM_OE_N_from_the_sram => SRAM_OE_N,
			SRAM_UB_N_from_the_sram => SRAM_UB_N,
```

```
                SRAM_WE_N_from_the_sram => SRAM_WE_N,

                latch1_from_the_nes_controller      => GPIO_0(2),
                pulse1_from_the_nes_controller      => GPIO_0(4),
                data1_to_the_nes_controller         => GPIO_0(6),
                latch2_from_the_nes_controller      => GPIO_0(28),
                pulse2_from_the_nes_controller      => GPIO_0(30),
                data2_to_the_nes_controller         => GPIO_0(32),

                --PS2_Clk_to_the_ps2 => PS2_CLK,
                --PS2_Data_to_the_ps2 => PS2_DAT,

                VGA_BLANK_from_the_de2_vga_controller_0 => VGA_BLANK,
                VGA_B_from_the_de2_vga_controller_0 => VGA_B,
                VGA_CLK_from_the_de2_vga_controller_0 => VGA_CLK,
                VGA_G_from_the_de2_vga_controller_0 => VGA_G,
                VGA_HS_from_the_de2_vga_controller_0 => VGA_HS,
                VGA_R_from_the_de2_vga_controller_0 => VGA_R,
                VGA_SYNC_from_the_de2_vga_controller_0 => VGA_SYNC,
                VGA_VS_from_the_de2_vga_controller_0 => VGA_VS,
                sw_to_the_de2_vga_controller_0 => SW(17 downto 0),

                AUD_ADCDAT_to_the_de2_audio_controller_0 => AUD_ADCAT,
                AUD_ADCLRCK_from_the_de2_audio_controller_0 => AUD_ADCLRCK,
                AUD_BCLK_to_and_from_the_de2_audio_controller_0 => AUD_BCLK,
                AUD_DACDAT_from_the_de2_audio_controller_0 => AUD_DACDAT,
                AUD_DACLRCK_from_the_de2_audio_controller_0 => AUD_DACLRCK,
                AUD_XCK_from_the_de2_audio_controller_0 => AUD_XCK,
                I2C_SCLK_from_the_de2_audio_controller_0 => I2C_SCLK,
                I2C_SDAT_to_and_from_the_de2_audio_controller_0 => I2C_SDAT,
                iCLK_to_the_de2_audio_controller_0 => iCLK,
                iRST_N_to_the_de2_audio_controller_0 => iRST_N,
                leds_from_the_de2_audio_controller_0 => LEDR(15 downto 0)
        );


end rtl;
```

```
********************************************************************************
de2_audio_controller.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

entity de2_audio_controller is

        port (

                clk                     : in  std_logic;
                reset_n         : in  std_logic;
                read                    : in  std_logic;
                write                   : in  std_logic;
                chipselect      : in  std_logic;
                address         : in  std_logic_vector(3 downto 0);
                readdata                : out std_logic_vector(31 downto 0);
                writedata       : in  std_logic_vector(31 downto 0);


                        --      Audio interface signals
                AUD_ADCLRCK     : out std_logic;        --    Audio CODEC ADC LR Clock
                AUD_ADCDAT      : in  std_logic;        --    Audio CODEC ADC Data
                AUD_DACLRCK     : out std_logic;        --    Audio CODEC DAC LR Clock
                AUD_DACDAT      : out std_logic;        --    Audio CODEC DAC Data
                AUD_BCLK                : inout std_logic;      --    Audio CODEC Bit-Stream Clock
                AUD_XCK         : out std_logic;        --      Chip Clock


                iCLK                    : in std_logic;
                iRST_N          : in std_logic;
                I2C_SCLK                : out std_logic;
                I2C_SDAT                : inout std_logic;

                leds                    : out std_logic_vector(15 downto 0)
        );

end de2_audio_controller;
```

```
architecture datapath of de2_audio_controller is

component de2_wm8731_audio is
	port (
		clk				: in std_logic;
			--		Audio CODEC Chip Clock AUD_XCK
		reset_n				: in std_logic;

		start_sound	: in std_logic;
	--		Start sound playback
		select_sound	: in std_logic_vector(3 downto 0);		--		Select
a sound

		change_divider_enable : in std_logic;
	--	Enable divider change
		divider_in		: in std_logic_vector(31 downto 0);		--
	Value to change divider to

		 -- Audio interface signals
		AUD_ADCLRCK	: out std_logic;
	--	Audio CODEC ADC LR Clock
		AUD_ADCDAT	: in  std_logic;
	--	Audio CODEC ADC Data
		AUD_DACLRCK	: out std_logic;
	--	Audio CODEC DAC LR Clock
		AUD_DACDAT	: out std_logic;
	--	Audio CODEC DAC Data
		AUD_BCLK	: inout std_logic;
	--	Audio CODEC Bit-Stream Clock

		leds			: out std_logic_vector(15 downto 0)
	);
end component;

component de2_i2c_av_config is
	port (
		iCLK		: in std_logic;
```

```
                iRST_N : in std_logic;
                I2C_SCLK : out std_logic;
                I2C_SDAT : inout std_logic
        );
end component;


signal audio_clock : unsigned(1 downto 0) := "00";
signal start           : std_logic := '0';
signal change_en       : std_logic := '0';


begin

        process (clk)
        begin
                if rising_edge(clk) then
                        audio_clock <= audio_clock + "1";
                end if;
        end process;

        process (clk)
        begin
                if rising_edge(clk) then
                        if reset_n = '0' then
                                start <= '0';
                                change_en <= '0';
                        elsif chipselect = '1' then
                                if write = '1' then
                                        change_en <= '1';
                                        start <= '0';
                                elsif read = '1' then
                                        start <= '1';
                                        change_en <= '0';
                                else
                                        start <= '0';
                                        change_en <= '0';
                                end if;
                        else
                                start <= '0';
```

```
                        change_en <= '0';
                end if;
        end if;
end process;

AUD_XCK <= audio_clock(1);

i2c: de2_i2c_av_config port map (
        iCLK     => clk,
        iRST_n   => '1',
        I2C_SCLK => I2C_SCLK,
        I2C_SDAT => I2C_SDAT
);

V1: de2_wm8731_audio port map (
        clk                          => audio_clock(1),
        reset_n              => reset_n,
        start_sound          => start,
        select_sound   => address,
        change_divider_enable => change_en,
        divider_in           => writedata,

          --    Audio interface signals
        AUD_ADCLRCK          => AUD_ADCLRCK,
        AUD_ADCDAT           => AUD_ADCDAT,
        AUD_DACLRCK          => AUD_DACLRCK,
        AUD_DACDAT           => AUD_DACDAT,
        AUD_BCLK                     => AUD_BCLK,

        leds                         => leds
);

end datapath;


--------------------------------------------------------------------------------
de2_nes_controller.vhd

library ieee;
```

```
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;


-- Address:           1 = player1, 2= player2
--
-- Writedata:  8 bits of controller value in this order
--                                  A-B-Se-St-Up-Down-Left-Right

entity de2_nes_controller is

        port (

                clk             : in  std_logic;
                reset_n         : in  std_logic;
                read            : in  std_logic;
                write           : in  std_logic;
                chipselect      : in  std_logic;
                address         : in  std_logic_vector(3 downto 0);
                readdata        : out std_logic_vector(31 downto 0);
                writedata       : in  std_logic_vector(31 downto 0);

                latch1          : out std_logic;
                pulse1          : out std_logic;
                data1                 : in std_logic;
                latch2          : out std_logic;
                pulse2          : out std_logic;
                data2                 : in std_logic

        );

end de2_nes_controller;

architecture datapath of de2_nes_controller is

        signal buttons1       : std_logic_vector(7 downto 0);
        signal buttons2       : std_logic_vector(7 downto 0);

        signal reset          : std_logic;
```

```
	component nes_fsm is
		port (
			clk			: in std_logic;
			reset			: in std_logic;
			latch1		: out std_logic;
			pulse1		: out std_logic;
			data1			: in std_logic;
			latch2		: out std_logic;
			pulse2		: out std_logic;
			data2			: in std_logic;
			buttons1_out: out std_logic_vector(7 downto 0);
			buttons2_out: out std_logic_vector(7 downto 0)
		);
	end component;

begin

	reset <= not(reset_n);

	process (clk)
	begin
		if rising_edge(clk) then

			if reset = '1' then
				readdata <= (others => '0');

			else

				if chipselect = '1' then -- This chip is right one
					-- Read --
					if read = '1' then
						--leds(15) <= '1';
						--leds(14 downto 11) <= address;
						--leds(7 downto 0) <= buttons1;
						readdata(31 downto 8) <= (others => '0');
						if address = "0001" then -- First player
							readdata(7 downto 0) <= buttons1;
```

```
                                        elsif address = "0010" then -- Second player
                                                readdata(7 downto 0) <= buttons2;
                                        end if; -- end address
                                end if; -- end read
                        end if; --end chipselect

                end if; -- reset
        end if;-- end rising edge
    end process;

    NES2: nes_fsm port map(

        clk                             => clk,
        reset                           => reset,
        latch1                  => latch1,
        pulse1                  => pulse1,
        data1                           => data1,
        latch2                  => latch2,
        pulse2                  => pulse2,
        data2                           => data2,
        buttons1_out        => buttons1,
        buttons2_out        => buttons2
    );

end datapath;


--------------------------------------------------------------------------------
de2_sram_controller.vhd

library ieee;
use ieee.std_logic_1164.all;

entity de2_sram_controller is

  port (
    signal chipselect : in std_logic;
    signal write, read : in std_logic;
    signal address  :  in std_logic_vector(17 downto 0);
```

```
    signal readdata : out std_logic_vector(15 downto 0);
    signal writedata : in std_logic_vector(15 downto 0);
    signal byteenable : in std_logic_vector(1 downto 0);

    signal SRAM_DQ : inout std_logic_vector(15 downto 0);
    signal SRAM_ADDR : out std_logic_vector(17 downto 0);
    signal SRAM_UB_N, SRAM_LB_N : out std_logic;
    signal SRAM_WE_N, SRAM_CE_N : out std_logic;
    signal SRAM_OE_N            : out std_logic
    );

end de2_sram_controller;

architecture dp of de2_sram_controller is
begin

  SRAM_DQ <= writedata when write = '1'
             else (others => 'Z');
  readdata <= SRAM_DQ;
  SRAM_ADDR <= address;
  SRAM_UB_N <= not byteenable(1);
  SRAM_LB_N <= not byteenable(0);
  SRAM_WE_N <= not write;
  SRAM_CE_N <= not chipselect;
  SRAM_OE_N <= not read;

end dp;



--------------------------------------------------------------------------------
de2_vga_controller.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.definitions.all;
--
---- PROTOCOL:
--             -- ADDRESS
```

```
--                    -- 0001 = SNAKE1
--                    -- 0010 = TILES
--                    -- 0011 = RESET
                      -- 0100 = SPLASH SCREEN
--
--              --WRITEDATA
--                    --4-0: Y (LSB) -- must be ONLY 5 bits
--                    --10-5: X             -- must be at least 6 bits
                      --11-19: unused
--                    --24-20: SPRITE SELECT
--                    --25: 1=ADD, 0=REMOVE
--                    --26: 0=PLAYER1, 1= PLAYER2
--                    --27-31: UNUSED (MSB)
--
--              --Tiles/Snakes protocol (in tiles_ram, from controller to raster)
--                    -- 8 bits
--                    -- 4-0: Sprite select
--                    -- 5    : Player select (0 for player1, 1 for player2)
--                    -- 6   : unused
--                    -- 7    : Enabled/Active signal
--
---- On Add:
----    Only to tail
----           NEED to send second change for second to head
---- On Remove:
----           Can only happen on tail
----           Must send second change to second to tail, with correct tail
----
--
entity snake_plus_vga is

  port (
    clk        : in  std_logic;
    reset_n    : in  std_logic;
    read       : in  std_logic;
    write      : in  std_logic;
    chipselect : in  std_logic;
    address    : in  std_logic_vector(3 downto 0);
    readdata   : out std_logic_vector(31 downto 0);
    writedata  : in  std_logic_vector(31 downto 0);
```

```
        VGA_CLK,                            -- Clock
    VGA_HS,                                 -- H_SYNC
    VGA_VS,                                 -- V_SYNC
    VGA_BLANK,                              -- BLANK
    VGA_SYNC   : out std_logic;             -- SYNC
    VGA_R,                                  -- Red[9:0]
    VGA_G,                                  -- Green[9:0]
    VGA_B             : out std_logic_vector(9 downto 0);  -- Blue[9:0]

    --leds       : out std_logic_vector(15 downto 0);
        sw                  : in std_logic_vector(17 downto 0)
    );

end snake_plus_vga;
--
architecture rtl of snake_plus_vga is

       signal reset                         : std_logic;
       signal soft_reset                    : std_logic                          := '0';

  -- Main memory elements
  -- To/From manage_tiles, add_remove_snake
       signal tiles_write_data                              : std_logic_vector(7 downto 0);
       signal snake_write_data                              : std_logic_vector(7 downto 0);
       signal tiles_write_address                 : std_logic_vector(10 downto 0);
       signal snake_write_address                 : std_logic_vector(10 downto 0);
       signal tiles_write_enable                  : std_logic          := '0';
       signal snake_write_enable                  : std_logic          := '0';

       -- To/From vga_raster
       signal tiles_read_address                  : std_logic_vector(10 downto 0);
       signal snake_read_address                  : std_logic_vector(10 downto 0);
       signal tiles_read_data                               : std_logic_vector(7 downto 0);
       signal snake_read_data                               : std_logic_vector(7 downto 0);
       signal enable_splash_screen          : std_logic := '0';

  --Tile signals
       signal tiles_enabled         : std_logic                          := '0';
       signal snake_enabled         : std_logic                          := '0';
```

```
  -- For VGA
  signal clk25                              : std_logic                 := '0';

        --- Constants ---
              -- Write --
  constant W_SNAKE_SELECT                                 : std_logic_vector    := "0001"; --
Write only
  constant W_TILES_SELECT                                 : std_logic_vector    := "0010"; --
Write only
  constant W_SOFT_RESET                                           : std_logic_vector    :=
"0011"; -- Write only
  constant W_ENABLE_SPLASH_SCREEN           : std_logic_vector    := "0100"; -- Write only
  constant W_DISABLE_SPLASH_SCREEN          : std_logic_vector    := "0101"; -- Write only


component tiles_ram is
       port (
              clock                 : IN STD_LOGIC  := '1';
              data                  : IN STD_LOGIC_VECTOR (7 DOWNTO 0);
              rdaddress      : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
              wraddress      : IN STD_LOGIC_VECTOR (10 DOWNTO 0);
              wren                  : IN STD_LOGIC  := '0';
              q                             : OUT STD_LOGIC_VECTOR (7 DOWNTO 0)
       );
end component;
--
begin

  reset <= not(reset_n);

  --VGA stuff

  process (clk)
  begin
    if rising_edge(clk) then
      clk25 <= not clk25;
    end if;
  end process;
```

```
-- Memory stuff/ LEDS

process (clk)
begin
  if rising_edge(clk) then

            snake_enabled <= '0';
            tiles_enabled  <= '0';



    if reset = '1' or soft_reset = '1' then
                    readdata <= (others => '0');
                    soft_reset <= '0';
    else

                    if chipselect = '1' then -- This chip is right one

                            -- Write --
                            if write = '1' then

                                    -- Select snake1 --
                                    if address = W_SNAKE_SELECT then
                                            snake_enabled <= '1';
                                    end if;

                                    -- Select tiles --
                                    if address = W_TILES_SELECT then
                                            tiles_enabled <= '1';
                                    end if;

                                    if address = W_SOFT_RESET then
                                            soft_reset <= '1';
                                    end if;

                                    if address = W_ENABLE_SPLASH_SCREEN then
                                            enable_splash_screen <= '1';
                                    end if;
```

```
                                        if address = W_DISABLE_SPLASH_SCREEN then
                                                enable_splash_screen <= '0';
                                        end if;

                                -- Read --
                                elsif read = '1' then

--                                      if address = R_SNAKE1_HEAD then
--                                              readdata <= std_logic_vector(
to_unsigned(snake1_head_index, readdata'length) );
--                                      elsif address = R_SNAKE1_TAIL then
--                                              readdata <= std_logic_vector(
to_unsigned(snake1_tail_index, readdata'length) );
--                                      elsif address = R_SNAKE1_LENGTH then
--                                              readdata <= std_logic_vector(
to_unsigned(snake1_length, readdata'length) );
--                                      elsif address = R_SNAKE2_HEAD then
--                                              readdata <= std_logic_vector(
to_unsigned(snake2_head_index, readdata'length) );
--                                      elsif address = R_SNAKE1_TAIL then
--                                              readdata <= std_logic_vector(
to_unsigned(snake2_head_index, readdata'length) );
--                                      elsif address = R_SNAKE1_LENGTH then
--                                              readdata <= std_logic_vector(
to_unsigned(snake2_length, readdata'length) );
--                                      end if;

                                        readdata <= x"0000000" & "000" & enable_splash_screen;

                                end if; -- end write/read

                        end if; -- end chipselect
                end if; --end reset
        end if; --end rising edge
  end process;
```

```
        snake_store : tiles_ram port map (
                clock                   => clk,
                data                    => snake_write_data,
                rdaddress       => snake_read_address,
                wraddress       => snake_write_address,
                wren                    => snake_write_enable,
                q                               => snake_read_data
        );

        tiles_store : tiles_ram port map (
                clock                   => clk,
                data                    => tiles_write_data,
                rdaddress       => tiles_read_address,
                wraddress       => tiles_write_address,
                wren                    => tiles_write_enable,
                q                               => tiles_read_data
        );


--VGA stuff

V1: entity work.de2_vga_raster port map (
    reset => reset,
    clk => clk25,
    VGA_CLK => VGA_CLK,
    VGA_HS => VGA_HS,
    VGA_VS => VGA_VS,
    VGA_BLANK => VGA_BLANK,
    VGA_SYNC => VGA_SYNC,
    VGA_R => VGA_R,
    VGA_G => VGA_G,
    VGA_B => VGA_B,

         tiles_address => tiles_read_address,
         tiles_data            => tiles_read_data,
         snake_address => snake_read_address,
         snake_data            => snake_read_data,
         controller_enable_splash_screen => enable_splash_screen
```

```
  );



DD1: entity work.manage_tiles port map(

               clk                                 => clk,
               reset                               => (reset or soft_reset),
               enabled                             => tiles_enabled,
               data_in                             => writedata,
               data_out                            => tiles_write_data,
               address_out                 => tiles_write_address,
               enable_out                  => tiles_write_enable
);

AD1: entity work.add_remove_snake_part port map (

               clk                                 => clk,
               reset                               => (reset or soft_reset),
               enabled                             => snake_enabled,
               data_in                             => writedata,
               data_out                            => snake_write_data,
               address_out                 => snake_write_address,
               enable_out                  => snake_write_enable
);



end rtl;
--
--
-------------------------------------------------------------------------------------------
------------------------------ SUBCOMPONENTS FOR VGA CONTROLLER ---------------------------
-------------------------------------------------------------------------------------------
----
----
----
-----------------------------------
--------------- Tiles -------------
-----------------------------------
```

```
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.definitions.all;


entity manage_tiles is

	port(
		clk				: in std_logic;
		reset			: in std_logic;
		enabled			: in std_logic;
		data_in			: in std_logic_vector(31 downto 0);
		data_out		: out std_logic_vector(7 downto 0);
		address_out		: out std_logic_vector(10 downto 0);
		enable_out		: out std_logic
	);


end manage_tiles;

architecture mt of manage_tiles is

signal y_val		: std_logic_vector(4 downto 0);	-- 5 bits --
signal x_val		: std_logic_vector(5 downto 0);	-- 6 bits --
signal sprite_select	: std_logic_vector(4 downto 0);	-- 5 bits ---
signal add_remove	: std_logic;				-- 1
bit  ---

signal y_32			: std_logic_vector(10 downto 0);
signal y_8			: std_logic_vector(10 downto 0);
signal x_11			: std_logic_vector(10 downto 0);

begin


	y_val		<=	data_in(4 downto 0);
	x_val		<=	data_in(10 downto 5);
```

```
        add_remove            <=     data_in(25);
        sprite_select  <=     data_in(24 downto 20);

        y_32                          <=     "0"            & data_in(4 downto 0) & "00000";
        y_8                           <=     "000"   & data_in(4 downto 0) & "000";
        x_11                          <=     "00000"        & data_in(10 downto 5);

        process (clk)
        begin
        if rising_edge(clk) then

                enable_out <= '0';

                -- Reset --
                if reset = '1' then
                        data_out <= (others => '0');
                        address_out <= (others => '0');
                        enable_out <= '0';

                -- Enabled --
                elsif enabled = '1' then
                                data_out <= add_remove & "00" & sprite_select;
                                address_out <= std_logic_vector( unsigned(y_32) + unsigned(y_8) +
unsigned(x_11) );
                                enable_out <= '1';
                end if; -- reset/ enabled



        end if; -- rising edge
        end process; -- end clk


end mt;


--
--
--
```

```
-----------------------------------------
------------ Add Remove Snake Part ----------
-----------------------------------------

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.definitions.all;


entity add_remove_snake_part is

	port(
		clk 				: in std_logic;
		reset 			: in std_logic;
		enabled 			: in std_logic;
		data_in 			: in std_logic_vector(31 downto 0);
		data_out 			: out std_logic_vector(7 downto 0);
		address_out 		: out std_logic_vector(10 downto 0);
		enable_out 		: out std_logic
	);

end add_remove_snake_part;

architecture arsp of add_remove_snake_part is

signal y_val 		: std_logic_vector(4 downto 0);	-- 5 bits --
signal x_val 		: std_logic_vector(5 downto 0);	-- 6 bits --
signal sprite_select  : std_logic_vector(4 downto 0);	-- 5 bits ---
signal add_remove 		: std_logic;					-- 1
bit  ---
signal player_select : std_logic;				-- 1 bit  ---

signal y_32 			: std_logic_vector(10 downto 0);
signal y_8 			: std_logic_vector(10 downto 0);
signal x_11 			: std_logic_vector(10 downto 0);

begin
```

```
        y_val                <=     data_in(4 downto 0);
        x_val                <=     data_in(10 downto 5);
        sprite_select  <=     data_in(24 downto 20);
        add_remove           <=     data_in(25);
        player_select  <=     data_in(26);

        y_32                        <=     "0"           & data_in(4 downto 0) & "00000";
        y_8                         <=     "000"   & data_in(4 downto 0) & "000";
        x_11                        <=     "00000"       & data_in(10 downto 5);

        process (clk)
        begin
        if rising_edge(clk) then

                enable_out <= '0';

                -- Reset --
                if reset = '1' then
                        data_out <= (others => '0');
                        address_out <= (others => '0');
                        enable_out <= '0';

                -- Enabled --
                elsif enabled = '1' then
                                data_out <= add_remove & "0" & player_select & sprite_select;
                                address_out <= std_logic_vector( unsigned(y_32) + unsigned(y_8) +
unsigned(x_11) );
                                enable_out <= '1';
                end if; -- reset/ enabled


        end if; -- rising edge
        end process; -- end clk

end arsp;
```

```
************************************************************************************
de2_vga_raster.vhd

--------------------------------------------------------------------------------

-- PROTOCOL:
                        --9-0: Y (LSB)
                        --19-10: X
                        --24-20: SPRITE SELECT
                        --25: 1=ADD, 0=REMOVE
                        --26-27: Which segment referring to
                        --                    00=head, 01=second to head
                        --                    10=second to tail 11=tail
                        --28: increment flag, move all pieces, move head to new x & y
                        --29-31: UNUSED (MSB)
--
--------------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;
use work.definitions.all;

entity de2_vga_raster is

  port (
    reset                               : in std_logic;
    clk                                 : in std_logic;                    -- Should be
25.125 MHz

        tiles_address               : out std_logic_vector(10 downto 0);
        tiles_data                        : in std_logic_vector(7 downto 0);
        snake_address               : out std_logic_vector(10 downto 0);
        snake_data                        : in std_logic_vector(7 downto 0);

        controller_enable_splash_screen : in std_logic;

    VGA_CLK,                                                    -- Clock
    VGA_HS,                                                     -- H_SYNC
    VGA_VS,                                                     -- V_SYNC
    VGA_BLANK,                                                  -- BLANK
```

```
    VGA_SYNC                        : out std_logic;        -- SYNC
    VGA_R,                                                  -- Red[9:0]
    VGA_G,                                                  -- Green[9:0]
    VGA_B                                   : out std_logic_vector(9 downto 0) -- Blue[9:0]
    );

end de2_vga_raster;

architecture rtl of de2_vga_raster is


        COMPONENT splash_snake_rom IS
                PORT
        (
                address         : IN STD_LOGIC_VECTOR (15 DOWNTO 0);
                clock                   : IN STD_LOGIC  := '1';
                q                               : OUT STD_LOGIC_VECTOR (2 DOWNTO 0)
        );
        END COMPONENT;

        COMPONENT splash_title_rom IS
                PORT
        (
                address         : IN STD_LOGIC_VECTOR (13 DOWNTO 0);
                clock                   : IN STD_LOGIC  := '1';
                q                               : OUT STD_LOGIC_VECTOR (0 DOWNTO 0)
        );
        END COMPONENT;


  -- Video parameters

  constant HTOTAL       : integer := 800;
  constant HSYNC        : integer := 96;
  constant HBACK_PORCH  : integer := 48;
  constant HACTIVE      : integer := 640;
  constant HFRONT_PORCH : integer := 16;

  constant VTOTAL       : integer := 525;
```

```
constant VSYNC        : integer := 2;
constant VBACK_PORCH  : integer := 33;
constant VACTIVE      : integer := 480;
constant VFRONT_PORCH : integer := 10;

constant SPLASH_SNAKE_START_H : integer := 180;
constant SPLASH_SNAKE_START_V : integer := 120;
constant SPLASH_SNAKE_SIZE  : integer := 256;

constant SPLASH_TITLE_START_H : integer := 180;
constant SPLASH_TITLE_START_V : integer := 50;
constant SPLASH_TITLE_WIDTH   : integer := 256;
constant SPLASH_TITLE_HEIGHT   : integer := 64;

-- Signals for the video controller
signal Hcount : unsigned(9 downto 0);  -- Horizontal position (0-800)
signal Vcount : unsigned(9 downto 0);  -- Vertical position (0-524)
signal EndOfLine, EndOfField : std_logic;

signal vga_hblank, vga_hsync,
  vga_vblank, vga_vsync : std_logic;  -- Sync. signals



    signal inner_tile_h_pos                   : integer;
    signal inner_tile_v_pos                   : integer;
    signal tiles_h_pos                        : unsigned(5 downto 0);
    signal tiles_v_pos                        : unsigned(4 downto 0);
    signal splash_sprite_h_pos          : unsigned(9 downto 0);
    signal splash_sprite_v_pos          : unsigned(9 downto 0);

    signal splash_snake_address               : std_logic_vector(15 downto 0);
    signal splash_snake_address_enable  : std_logic;
    signal splash_snake_data                        : std_logic_vector(2 downto 0);
    signal splash_sprite_h_pos_16             : std_logic_vector(15 downto 0);
    signal splash_sprite_v_pos_16             : std_logic_vector(15 downto 0);

    signal splash_title_address_enable        : std_logic;
    signal splash_title_data                        : std_logic_vector(0 downto 0);
```

```
signal y_32                         : std_logic_vector(10 downto 0);
signal y_8                          : std_logic_vector(10 downto 0);
signal x_11                         : std_logic_vector(10 downto 0);
signal out_bounds           : std_logic    := '0';

signal sprite_select                        : std_logic_vector(7 downto 0);
signal snake_select                         : std_logic_vector(7 downto 0);
signal player_select                        : std_logic;


              -- process signals
signal green, blue, red, black, tan
       : std_logic;
signal white, pink, gray, yellow, brown
       : std_logic;
signal snake_head_orange, snake_head_black
: std_logic;
signal snake_body_black, snake_body_grey,
                            snake_body_orange, snake_body_white
              : std_logic;
signal snake_turn_black, snake_turn_grey,
                            snake_turn_orange, snake_turn_white
              : std_logic;
signal snake_tail_black, snake_tail_orange, snake_tail_yellow           : std_logic;
signal rabbit_y, rabbit_b, rabbit_w, rabbit_p
: std_logic;
signal mouse_y, mouse_p, mouse_l, mouse_b_eye, mouse_w_eye              : std_logic;
signal edwards_t, edwards_br, edwards_bl, edwards_p, ed_b_eye, ed_w_eye  : std_logic;
signal speed, growth_y, growth_r, freeze
       : std_logic;
signal wall
                                                    : std_logic;
signal P, one, two, W, I, N, S, T, E, exclam
       : std_logic;
signal pause, play
                                    : std_logic;
signal splash_snake_black, splash_snake_yellow,
                     splash_snake_green, splash_snake_red
       : std_logic;
```

```
        signal splash_title_green
                                    : std_logic;

                -- sprites
        type array_type_16x16 is array (0 to 15) of unsigned (0 to 15);

-- snake head sprites
        signal sprite_head_right_black      : array_type_16x16;
        signal sprite_head_right_orange  : array_type_16x16;
        signal sprite_head_left_black       : array_type_16x16;
        signal sprite_head_left_orange      : array_type_16x16;
        signal sprite_head_up_black         : array_type_16x16;
        signal sprite_head_up_orange        : array_type_16x16;
        signal sprite_head_down_black       : array_type_16x16;
        signal sprite_head_down_orange      : array_type_16x16;

        -- snake body colorings
        signal sprite_body_right_black      : array_type_16x16;
        signal sprite_body_right_grey       : array_type_16x16;
        signal sprite_body_right_orange     : array_type_16x16;
        signal sprite_body_right_white      : array_type_16x16;
        signal sprite_body_left_black       : array_type_16x16;
        signal sprite_body_left_grey        : array_type_16x16;
        signal sprite_body_left_orange      : array_type_16x16;
        signal sprite_body_left_white       : array_type_16x16;
        signal sprite_body_up_black         : array_type_16x16;
        signal sprite_body_up_grey                  : array_type_16x16;
        signal sprite_body_up_orange        : array_type_16x16;
        signal sprite_body_up_white         : array_type_16x16;
        signal sprite_body_down_black       : array_type_16x16;
        signal sprite_body_down_grey        : array_type_16x16;
        signal sprite_body_down_orange      : array_type_16x16;
        signal sprite_body_down_white       : array_type_16x16;

        --snake turn colorings
        signal sprite_turn_up_right_black   : array_type_16x16;
        signal sprite_turn_up_right_grey            : array_type_16x16;
        signal sprite_turn_up_right_orange  : array_type_16x16;
        signal sprite_turn_up_right_white   : array_type_16x16;
        signal sprite_turn_right_down_black : array_type_16x16;
```

```
        signal sprite_turn_right_down_grey   : array_type_16x16;
        signal sprite_turn_right_down_orange: array_type_16x16;
        signal sprite_turn_right_down_white  : array_type_16x16;
        signal sprite_turn_down_left_black   : array_type_16x16;
        signal sprite_turn_down_left_grey    : array_type_16x16;
        signal sprite_turn_down_left_orange  : array_type_16x16;
        signal sprite_turn_down_left_white   : array_type_16x16;
        signal sprite_turn_left_up_black           : array_type_16x16;
        signal sprite_turn_left_up_grey            : array_type_16x16;
        signal sprite_turn_left_up_orange    : array_type_16x16;
        signal sprite_turn_left_up_white           : array_type_16x16;

-- snake tail colorings
        signal sprite_tail_right_black       : array_type_16x16;
        signal sprite_tail_right_orange      : array_type_16x16;
        signal sprite_tail_right_yellow      : array_type_16x16;
        signal sprite_tail_left_black        : array_type_16x16;
        signal sprite_tail_left_orange       : array_type_16x16;
        signal sprite_tail_left_yellow       : array_type_16x16;
        signal sprite_tail_up_black          : array_type_16x16;
        signal sprite_tail_up_orange         : array_type_16x16;
        signal sprite_tail_up_yellow         : array_type_16x16;
        signal sprite_tail_down_black        : array_type_16x16;
        signal sprite_tail_down_orange       : array_type_16x16;
        signal sprite_tail_down_yellow       : array_type_16x16;

        -- rabbit colorings
        signal sprite_food_rabbit_y  : array_type_16x16;
        signal sprite_food_rabbit_p  : array_type_16x16;
        signal sprite_food_rabbit_b  : array_type_16x16;
        signal sprite_food_rabbit_w  : array_type_16x16;

        -- mouse colorings, rabbit eyes can be used for mouse eyes
        signal sprite_food_mouse_y   : array_type_16x16;
        signal sprite_food_mouse_p   : array_type_16x16;
        signal sprite_food_mouse_l   : array_type_16x16;

        -- edwards colorings
        signal sprite_food_edwards_n : array_type_16x16;
        signal sprite_food_edwards_t : array_type_16x16;
```

```
        signal sprite_food_edwards_l : array_type_16x16;
        signal sprite_food_edwards_p : array_type_16x16;

        -- needle growth coloring
        signal sprite_powup_growth_r : array_type_16x16;
        signal sprite_powup_growth_y : array_type_16x16;

        -- lightning speed coloring
        signal sprite_powup_speed    : array_type_16x16;

        -- ice freeze coloring
        signal sprite_powup_freeze   : array_type_16x16;

        -- wall obstacle coloring
        signal sprite_wall                  : array_type_16x16;

        -- letter colorings
        signal sprite_P                           : array_type_16x16;
        signal sprite_1                           : array_type_16x16;
        signal sprite_2                           : array_type_16x16;
        signal sprite_W                           : array_type_16x16;
        signal sprite_I                           : array_type_16x16;
        signal sprite_N                           : array_type_16x16;
        signal sprite_S                           : array_type_16x16;
        signal sprite_T                           : array_type_16x16;
        signal sprite_E                           : array_type_16x16;
        signal sprite_exclam                : array_type_16x16;

        -- pause and play colorings
        signal sprite_pause                 : array_type_16x16;
        signal sprite_play                  : array_type_16x16;



begin


        SSE: entity work.splash_snake_rom PORT MAP
        (
```

```
                address        => splash_snake_address,
                clock                  => clk,
                q                              => splash_snake_data
        );

    STE: entity work.splash_title_rom PORT MAP
    (
                address        => splash_snake_address(13 downto 0),
                clock                  => clk,
                q                              => splash_title_data
        );


  -- Horizontal and vertical counters

HCounter : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      Hcount <= (others => '0');
    elsif EndOfLine = '1' then
      Hcount <= (others => '0');
    else
      Hcount <= Hcount + 1;
    end if;
  end if;
end process HCounter;

EndOfLine <= '1' when Hcount = HTOTAL - 1 else '0';

VCounter: process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      Vcount <= (others => '0');
    elsif EndOfLine = '1' then
      if EndOfField = '1' then
        Vcount <= (others => '0');
      else
        Vcount <= Vcount + 1;
```

```
      end if;
    end if;
  end if;
end process VCounter;

EndOfField <= '1' when Vcount = VTOTAL - 1 else '0';

-- State machines to generate HSYNC, VSYNC, HBLANK, and VBLANK

HSyncGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' or EndOfLine = '1' then
      vga_hsync <= '1';
    elsif Hcount = HSYNC - 1 then
      vga_hsync <= '0';
    end if;
  end if;
end process HSyncGen;

HBlankGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      vga_hblank <= '1';
    elsif Hcount = HSYNC + HBACK_PORCH then
      vga_hblank <= '0';
    elsif Hcount = HSYNC + HBACK_PORCH + HACTIVE then
      vga_hblank <= '1';
    end if;
  end if;
end process HBlankGen;

VSyncGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      vga_vsync <= '1';
    elsif EndOfLine ='1' then
      if EndOfField = '1' then
```

```
        vga_vsync <= '1';
      elsif Vcount = VSYNC - 1 then
        vga_vsync <= '0';
      end if;
    end if;
  end if;
end process VSyncGen;

VBlankGen : process (clk)
begin
  if rising_edge(clk) then
    if reset = '1' then
      vga_vblank <= '1';
    elsif EndOfLine = '1' then
      if Vcount = VSYNC + VBACK_PORCH - 1 then
        vga_vblank <= '0';
      elsif Vcount = VSYNC + VBACK_PORCH + VACTIVE - 1 then
        vga_vblank <= '1';
      end if;
    end if;
  end if;
end process VBlankGen;
```

```
-------------------------------------------------------------------
------------ Special Snake_Plus Tile Calculation Logic -------------
-------------------------------------------------------------------

InnerTileH : process (clk)
begin
            if rising_edge(clk) then
                  if reset = '1' then
                        inner_tile_h_pos <= 0;
                        tiles_h_pos <= (others => '0');
                  elsif HCount >= HSYNC + HBACK_PORCH + HACTIVE then
                        inner_tile_h_pos <= 0;
                        tiles_h_pos <= "111111"; -- random > 1200
```

```
                        out_bounds <= '1';
                elsif HCount = HSYNC + HBACK_PORCH then
                        tiles_h_pos <= "000000";
                        out_bounds <= '0';
                elsif  HCount > HSYNC + HBACK_PORCH and
                                        HCount < HSYNC + HBACK_PORCH + HACTIVE then
                        inner_tile_h_pos <= inner_tile_h_pos + 1;
                        if inner_tile_h_pos >= 15 then -- 0-15 should be used
                                inner_tile_h_pos <= 0;


                                tiles_h_pos <= tiles_h_pos + 1;
                        end if;--end inner_tile_h_pos
                end if; -- reset/endofline/hcount
        end if; --end clk
end process InnerTileH;



InnerTileV : process (clk)
begin
        if rising_edge(clk) then
                if reset = '1' then
                        inner_tile_v_pos <= 0;
                        tiles_v_pos <= (others => '0');
                elsif VCount >= VSYNC + VBACK_PORCH + VACTIVE - 1 then
                        inner_tile_v_pos <= 0;
                        tiles_v_pos <= "11111"; -- random > 39
                elsif VCount = VSYNC + VBACK_PORCH - 1 then
                        tiles_v_pos <= "00000";
                elsif  VCount > VSYNC + VBACK_PORCH - 1 and
                                        VCount < VSYNC + VBACK_PORCH + VACTIVE - 1 and
                                        EndOfLine = '1' then
                        inner_tile_v_pos <= inner_tile_v_pos + 1;
                        if inner_tile_v_pos >= 15 then -- 0-15 should be used
                                inner_tile_v_pos <= 0;
                                tiles_v_pos <= tiles_v_pos + 1;
                        end if;--end inner_tile_v_pos
                end if; -- reset/endofline/hcount
        end if; --end clk
end process InnerTileV;
```

```
  --TILES_IN(tiles_h_pos,tiles_v_pos);
 --SNAKE_IN(tiles_h_pos,tiles_v_pos);

       y_32                          <= "0" & std_logic_vector(tiles_v_pos) & "00000";
       y_8                           <= "000" & std_logic_vector(tiles_v_pos) & "000";
       x_11                          <= "00000" & std_logic_vector(tiles_h_pos);

       tiles_address  <= std_logic_vector( unsigned(y_32) + unsigned(y_8) + unsigned(x_11))
when out_bounds = '0' else "11111111111";
       sprite_select  <= tiles_data;
       snake_address  <=     std_logic_vector( unsigned(y_32) + unsigned(y_8) +
unsigned(x_11)) when out_bounds = '0' else "11111111111";
       snake_select   <= snake_data;

  -------------------------------------------------------------
  ------------ End Tile Calculation Logic ---------------------
  -------------------------------------------------------------

  -------------------------------------------------------------
  ----------- Begin Sprite Display Logic ----------------------
  -------------------------------------------------------------

       -- snake sprite generation
  SnakeSpriteGen : process (clk)
       variable sprite_h_pos, sprite_v_pos : integer;
  begin
       if rising_edge(clk) then
```

```
if reset = '1' then
        snake_head_black            <= '0';
        snake_head_orange     <= '0';

        snake_body_black            <= '0';
        snake_body_grey             <= '0';
        snake_body_orange           <= '0';
        snake_body_white            <= '0';

        snake_turn_black            <= '0';
        snake_turn_grey             <= '0';
        snake_turn_orange           <= '0';
        snake_turn_white            <= '0';

        snake_tail_black            <= '0';
        snake_tail_orange     <= '0';
        snake_tail_yellow     <= '0';


else -- Not reset

        sprite_h_pos := inner_tile_h_pos;
        sprite_v_pos := inner_tile_v_pos;

        -- Snake head color signals
        snake_head_black            <= '0';
        snake_head_orange     <= '0';

        -- Snake body color signals
        snake_body_black            <= '0';
        snake_body_grey             <= '0';
        snake_body_orange           <= '0';
        snake_body_white            <= '0';

        --Snake body turn
        snake_turn_black            <= '0';
        snake_turn_grey             <= '0';
        snake_turn_orange           <= '0';
        snake_turn_white            <= '0';
```

```
                    -- Snake tail color signals
                    snake_tail_black             <= '0';
                    snake_tail_orange     <= '0';
                    snake_tail_yellow     <= '0';

                    -- Choose which player
                    player_select <= snake_select(5);

                    -- HEAD ORIENTATIONS
                    if snake_select(7) = '1' and snake_select(4 downto 0) = SNAKE_HEAD_RIGHT
then
                            if sprite_head_right_black(sprite_v_pos)(sprite_h_pos) = '1'
then
                                  snake_head_black <= '1';
                            elsif sprite_head_right_orange(sprite_v_pos)(sprite_h_pos) = '1'
then
                                  snake_head_orange <= '1';
                            end if; -- end sprite snake head

                    elsif snake_select(7) = '1' and snake_select(4 downto 0) =
SNAKE_HEAD_LEFT then
                                  if sprite_head_left_black(sprite_v_pos)(sprite_h_pos) =
'1' then
                                        snake_head_black <= '1';
                                  elsif sprite_head_left_orange(sprite_v_pos)(sprite_h_pos)
= '1' then
                                        snake_head_orange <= '1';
                            end if; -- end sprite snake head

                    elsif snake_select(7) = '1' and snake_select(4 downto 0) = SNAKE_HEAD_UP
then
                                  if sprite_head_up_black(sprite_v_pos)(sprite_h_pos) = '1'
then
                                        snake_head_black <= '1';
                                  elsif sprite_head_up_orange(sprite_v_pos)(sprite_h_pos) =
'1' then
                                        snake_head_orange <= '1';
                            end if; -- end sprite snake head
```

```
                    elsif snake_select(7) = '1' and snake_select(4 downto 0) =
SNAKE_HEAD_DOWN then
                                if sprite_head_down_black(sprite_v_pos)(sprite_h_pos) =
'1' then
                                        snake_head_black <= '1';
                                elsif sprite_head_down_orange(sprite_v_pos)(sprite_h_pos)
= '1' then
                                        snake_head_orange <= '1';
                            end if; -- end sprite snake head



                    --BODY ORIENTATIONS
                    elsif snake_select(7) = '1' and snake_select(4 downto 0) =
SNAKE_BODY_RIGHT then
                             if sprite_body_right_black(sprite_v_pos)(sprite_h_pos) = '1'
then
                                        snake_body_black <= '1';
                                elsif sprite_body_right_grey(sprite_v_pos)(sprite_h_pos) =
'1' then
                                        snake_body_grey <= '1';
                                elsif sprite_body_right_orange(sprite_v_pos)(sprite_h_pos)
= '1' then
                                        snake_body_orange <= '1';
                                elsif sprite_body_right_white(sprite_v_pos)(sprite_h_pos)
= '1' then
                                        snake_body_white <= '1';
                            end if; -- end sprite snake tail

                    elsif snake_select(7) = '1' and snake_select(4 downto 0) =
SNAKE_BODY_LEFT then
                             if sprite_body_left_black(sprite_v_pos)(sprite_h_pos) = '1'
then
                                        snake_body_black <= '1';
                                elsif sprite_body_left_grey(sprite_v_pos)(sprite_h_pos) =
'1' then
                                        snake_body_grey <= '1';
                                elsif sprite_body_left_orange(sprite_v_pos)(sprite_h_pos)
= '1' then
                                        snake_body_orange <= '1';
```

```
                                elsif sprite_body_left_white(sprite_v_pos)(sprite_h_pos) =
'1' then
                                        snake_body_white <= '1';
                                end if; -- end sprite snake tail

                        elsif snake_select(7) = '1' and snake_select(4 downto 0) = SNAKE_BODY_UP
then
                                 if sprite_body_up_black(sprite_v_pos)(sprite_h_pos) = '1' then
                                        snake_body_black <= '1';
                                elsif sprite_body_up_grey(sprite_v_pos)(sprite_h_pos) =
'1' then
                                        snake_body_grey <= '1';
                                elsif sprite_body_up_orange(sprite_v_pos)(sprite_h_pos) =
'1' then
                                        snake_body_orange <= '1';
                                elsif sprite_body_up_white(sprite_v_pos)(sprite_h_pos) =
'1' then
                                        snake_body_white <= '1';
                                end if; -- end sprite snake tail

                        elsif snake_select(7) = '1' and snake_select(4 downto 0) =
SNAKE_BODY_DOWN then
                                 if sprite_body_down_black(sprite_v_pos)(sprite_h_pos) = '1'
then
                                        snake_body_black <= '1';
                                elsif sprite_body_down_grey(sprite_v_pos)(sprite_h_pos) =
'1' then
                                        snake_body_grey <= '1';
                                elsif sprite_body_down_orange(sprite_v_pos)(sprite_h_pos)
= '1' then
                                        snake_body_orange <= '1';
                                elsif sprite_body_down_white(sprite_v_pos)(sprite_h_pos) =
'1' then
                                        snake_body_white <= '1';
                                end if; -- end sprite snake tail



                        --TURN ORIENTATIONS
```

```
                    elsif snake_select(7) = '1' and snake_select(4 downto 0) =
SNAKE_TURN_UP_RIGHT then
                            if sprite_turn_up_right_black(sprite_v_pos)(sprite_h_pos) = '1'
then
                                    snake_turn_black <= '1';
                                elsif
sprite_turn_up_right_grey(sprite_v_pos)(sprite_h_pos) = '1' then
                                    snake_turn_grey <= '1';
                                elsif
sprite_turn_up_right_orange(sprite_v_pos)(sprite_h_pos) = '1' then
                                    snake_turn_orange <= '1';
                                elsif
sprite_turn_up_right_white(sprite_v_pos)(sprite_h_pos) = '1' then
                                    snake_body_white <= '1';
                            end if; -- end sprite snake tail


                    elsif snake_select(7) = '1' and snake_select(4 downto 0) =
SNAKE_TURN_RIGHT_DOWN then
                            if sprite_turn_right_down_black(sprite_v_pos)(sprite_h_pos) =
'1' then
                                    snake_turn_black <= '1';
                                elsif
sprite_turn_right_down_grey(sprite_v_pos)(sprite_h_pos) = '1' then
                                    snake_turn_grey <= '1';
                                elsif
sprite_turn_right_down_orange(sprite_v_pos)(sprite_h_pos) = '1' then
                                    snake_turn_orange <= '1';
                                elsif
sprite_turn_right_down_white(sprite_v_pos)(sprite_h_pos) = '1' then
                                    snake_body_white <= '1';
                            end if; -- end sprite snake tail


                    elsif snake_select(7) = '1' and snake_select(4 downto 0) =
SNAKE_TURN_DOWN_LEFT then
                            if sprite_turn_down_left_black(sprite_v_pos)(sprite_h_pos) =
'1' then
                                    snake_turn_black <= '1';
                                elsif
sprite_turn_down_left_grey(sprite_v_pos)(sprite_h_pos) = '1' then
                                    snake_turn_grey <= '1';
```

```
                                    elsif
sprite_turn_down_left_orange(sprite_v_pos)(sprite_h_pos) = '1' then
                                            snake_turn_orange <= '1';
                                    elsif
sprite_turn_down_left_white(sprite_v_pos)(sprite_h_pos) = '1' then
                                            snake_body_white <= '1';
                                end if; -- end sprite snake tail


                            elsif snake_select(7) = '1' and snake_select(4 downto 0) =
SNAKE_TURN_LEFT_UP then
                                 if sprite_turn_left_up_black(sprite_v_pos)(sprite_h_pos) = '1'
then
                                            snake_turn_black <= '1';
                                    elsif sprite_turn_left_up_grey(sprite_v_pos)(sprite_h_pos)
= '1' then
                                            snake_turn_grey <= '1';
                                    elsif
sprite_turn_left_up_orange(sprite_v_pos)(sprite_h_pos) = '1' then
                                            snake_turn_orange <= '1';
                                    elsif
sprite_turn_left_up_white(sprite_v_pos)(sprite_h_pos) = '1' then
                                            snake_body_white <= '1';
                                end if; -- end sprite snake tail




                            --TAIL ORIENTATIONS
                            -- Right orientation for snake tail
                            elsif snake_select(7) = '1' and snake_select(4 downto 0) =
SNAKE_TAIL_RIGHT then
                                    if sprite_tail_right_black(sprite_v_pos)(sprite_h_pos) = '1'
then
                                            snake_tail_black <= '1';
                                    elsif sprite_tail_right_orange(sprite_v_pos)(sprite_h_pos) = '1'
then
                                            snake_tail_orange <= '1';
                                    elsif sprite_tail_right_yellow(sprite_v_pos)(sprite_h_pos) = '1'
then
                                            snake_tail_yellow <= '1';
                                    end if; -- end sprite snake tail
```

```
                    elsif snake_select(7) = '1' and snake_select(4 downto 0) =
SNAKE_TAIL_LEFT then
                            if sprite_tail_left_black(sprite_v_pos)(sprite_h_pos) = '1' then
                                  snake_tail_black <= '1';
                            elsif sprite_tail_left_orange(sprite_v_pos)(sprite_h_pos) = '1'
then
                                  snake_tail_orange <= '1';
                            elsif sprite_tail_left_yellow(sprite_v_pos)(sprite_h_pos) = '1'
then
                                  snake_tail_yellow <= '1';
                            end if; -- end sprite snake tail

                    elsif snake_select(7) = '1' and snake_select(4 downto 0) = SNAKE_TAIL_UP
then
                            if sprite_tail_up_black(sprite_v_pos)(sprite_h_pos) = '1' then
                                  snake_tail_black <= '1';
                            elsif sprite_tail_up_orange(sprite_v_pos)(sprite_h_pos) = '1'
then
                                  snake_tail_orange <= '1';
                            elsif sprite_tail_up_yellow(sprite_v_pos)(sprite_h_pos) = '1'
then
                                  snake_tail_yellow <= '1';
                            end if; -- end sprite snake tail

                    elsif snake_select(7) = '1' and snake_select(4 downto 0) =
SNAKE_TAIL_DOWN then
                            if sprite_tail_down_black(sprite_v_pos)(sprite_h_pos) = '1' then
                                  snake_tail_black <= '1';
                            elsif sprite_tail_down_orange(sprite_v_pos)(sprite_h_pos) = '1'
then
                                  snake_tail_orange <= '1';
                            elsif sprite_tail_down_yellow(sprite_v_pos)(sprite_h_pos) = '1'
then
                                  snake_tail_yellow <= '1';
                            end if; -- end sprite snake tail


                    else
                           snake_head_black             <= '0';
```

```
                        snake_head_orange     <= '0';
                        snake_body_black            <= '0';
                        snake_body_grey             <= '0';
                        snake_body_orange           <= '0';
                        snake_body_white            <= '0';
                        snake_turn_black            <= '0';
                        snake_turn_grey             <= '0';
                        snake_turn_orange           <= '0';
                        snake_turn_white            <= '0';
                        snake_tail_black            <= '0';
                        snake_tail_orange     <= '0';
                        snake_tail_yellow     <= '0';
                end if; -- end sprite select
        end if; -- end reset
    end if; --end clck/rising edge
end process SnakeSpriteGen;




 -- rabbit sprite generation
RabbitSpriteGen : process (clk)
variable sprite_h_pos, sprite_v_pos : integer;
begin
    if rising_edge(clk) then
        if reset = '1' then
                rabbit_y <= '0';
                rabbit_b <= '0';
                rabbit_w <= '0';
                rabbit_p <= '0';

        elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = RABBIT_CODE then

                sprite_h_pos := inner_tile_h_pos;
                sprite_v_pos := inner_tile_v_pos;

                rabbit_y <= '0';
                rabbit_b <= '0';
                rabbit_w <= '0';
                rabbit_p <= '0';
```

```
                if sprite_food_rabbit_y(sprite_v_pos)(sprite_h_pos) = '1' then
                        rabbit_y <= '1';
                elsif sprite_food_rabbit_b(sprite_v_pos)(sprite_h_pos) = '1' then
                        rabbit_b <= '1';
                elsif sprite_food_rabbit_p(sprite_v_pos)(sprite_h_pos) = '1' then
                        rabbit_p <= '1';
                elsif sprite_food_rabbit_w(sprite_v_pos)(sprite_h_pos) = '1' then
                        rabbit_w <= '1';
                end if;
        else
                rabbit_y <= '0';
                rabbit_p <= '0';
                rabbit_w <= '0';
                rabbit_b <= '0';
        end if;
    end if;
end process RabbitSpriteGen;

-- mouse sprite generation
MouseSpriteGen : process (clk)
variable sprite_h_pos, sprite_v_pos : integer;
begin
    if rising_edge(clk) then
        if reset = '1' then
                mouse_y <= '0';
                mouse_l <= '0';
                mouse_p <= '0';
                mouse_b_eye <= '0';
                mouse_w_eye <= '0';
        elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = MOUSE_CODE  then

                sprite_h_pos := inner_tile_h_pos;
                sprite_v_pos := inner_tile_v_pos;

                mouse_y <= '0';
                mouse_l <= '0';
                mouse_p <= '0';
                mouse_b_eye <= '0';
                mouse_w_eye <= '0';
```

```
                if sprite_food_mouse_y(sprite_v_pos)(sprite_h_pos) = '1' then
                        mouse_y <= '1';
                elsif sprite_food_mouse_l(sprite_v_pos)(sprite_h_pos) = '1' then
                        mouse_l <= '1';
                elsif sprite_food_mouse_p(sprite_v_pos)(sprite_h_pos) = '1' then
                        mouse_p <= '1';
                elsif sprite_food_rabbit_b(sprite_v_pos)(sprite_h_pos) = '1' then
                        mouse_b_eye <= '1';
                elsif sprite_food_rabbit_w(sprite_v_pos)(sprite_h_pos) = '1' then
                        mouse_w_eye <= '1';
                end if;
        else
                mouse_y <= '0';
                mouse_l <= '0';
                mouse_p <= '0';
                mouse_b_eye <= '0';
                mouse_w_eye <= '0';
        end if; --
    end if;
end process MouseSpriteGen;

-- edwards sprite generation
EdwardsSpriteGen : process (clk)
variable sprite_h_pos, sprite_v_pos : integer;
begin
    if rising_edge(clk) then
        if reset = '1' then
                edwards_t <= '0';
                edwards_bl <= '0';
                edwards_br <= '0';
                edwards_p <= '0';
                ed_w_eye <= '0';
                ed_b_eye <= '0';
        elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = EDWARDS_CODE then

                        sprite_h_pos := inner_tile_h_pos;
                        sprite_v_pos := inner_tile_v_pos;

                        edwards_t <= '0';
```

```
                        edwards_bl <= '0';
                        edwards_br <= '0';
                        edwards_p <= '0';
                        ed_w_eye <= '0';
                        ed_b_eye <= '0';

                    if sprite_food_edwards_t(sprite_v_pos)(sprite_h_pos) = '1' then
                        edwards_t <= '1';
                    elsif sprite_food_edwards_l(sprite_v_pos)(sprite_h_pos) = '1' then
                        edwards_bl <= '1';
                    elsif sprite_food_edwards_n(sprite_v_pos)(sprite_h_pos) = '1' then
                        edwards_br <= '1';
                    elsif sprite_food_edwards_p(sprite_v_pos)(sprite_h_pos) = '1' then
                        edwards_p <= '1';
                    elsif sprite_food_rabbit_w(sprite_v_pos)(sprite_h_pos) = '1' then
                        ed_w_eye <= '1';
                    elsif sprite_food_rabbit_b(sprite_v_pos)(sprite_h_pos) = '1' then
                        ed_b_eye <= '1';
                    end if;
            else
                    edwards_t <= '0';
                    edwards_bl <= '0';
                    edwards_br <= '0';
                    edwards_p <= '0';
                    ed_w_eye <= '0';
                    ed_b_eye <= '0';
            end if;
        end if;
    end process EdwardsSpriteGen;


    -- brick wall sprite generation
      BrickWallSpriteGen : process (clk)
        variable sprite_h_pos, sprite_v_pos : integer;
        begin
            if rising_edge(clk) then
                if reset = '1' then
                    wall <= '0';
                elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = WALL_CODE
then
```

```
                        sprite_h_pos := inner_tile_h_pos;
                        sprite_v_pos := inner_tile_v_pos;

                         if sprite_wall(sprite_v_pos)(sprite_h_pos) = '1' then
                               wall <= '1';
                         else
                               wall <= '0';
                         end if;
                  else
                        wall <= '0';
                  end if;
            end if;
  end process BrickWallSpriteGen;

   -- speed powerup sprite generation
       SpeedSpriteGen : process (clk)
         variable sprite_h_pos, sprite_v_pos : integer;
         begin
               if rising_edge(clk) then
                     if reset = '1' then
                           speed <= '0';
                     elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = SPEED_CODE
then
                               sprite_h_pos := inner_tile_h_pos;
                               sprite_v_pos := inner_tile_v_pos;
                         if sprite_powup_speed(sprite_v_pos)(sprite_h_pos) = '1' then
                               speed <= '1';
                         else
                               speed <= '0';
                         end if;
                     else
                           speed <= '0';
                     end if;
               end if;
  end process SpeedSpriteGen;

   -- freeze powerup sprite generation
       FreezeSpriteGen : process (clk)
         variable sprite_h_pos, sprite_v_pos : integer;
```

```
        begin
            if rising_edge(clk) then
                if reset = '1' then
                    freeze <= '0';
                elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = FREEZE_CODE
then
                        sprite_h_pos := inner_tile_h_pos;
                        sprite_v_pos := inner_tile_v_pos;
                     if sprite_powup_freeze(sprite_v_pos)(sprite_h_pos) = '1' then
                        freeze <= '1';
                     else
                        freeze <= '0';
                     end if;
                else
                    freeze <= '0';
                end if;
            end if;
  end process FreezeSpriteGen;


  -- growth power up generation
  GrowthSpriteGen : process (clk)
  variable sprite_h_pos, sprite_v_pos : integer;
  begin
      if rising_edge(clk) then
            if reset = '1' then
                growth_y <= '0';
                growth_r <= '0';
            elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = GROWTH_CODE then

                    sprite_h_pos := inner_tile_h_pos;
                    sprite_v_pos := inner_tile_v_pos;

                    growth_y <= '0';
                    growth_r <= '0';

                 if sprite_powup_growth_y(sprite_v_pos)(sprite_h_pos) = '1' then
                    growth_y <= '1';
                 elsif sprite_powup_growth_r(sprite_v_pos)(sprite_h_pos) = '1' then
                    growth_r <= '1';
```

```
                        else
                                growth_y <= '0';
                                growth_r <= '0';
                        end if;
                else
                        growth_y <= '0';
                        growth_r <= '0';
                end if;
        end if;
  end process GrowthSpriteGen;


        -- Number 1 generation
        Number1Gen : process (clk)
          variable sprite_h_pos, sprite_v_pos : integer;
          begin
                if rising_edge(clk) then
                        if reset = '1' then
                                one <= '0';
                        elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = ONE_CODE
then

                                        sprite_h_pos := inner_tile_h_pos;
                                        sprite_v_pos := inner_tile_v_pos;

                                if sprite_1(sprite_v_pos)(sprite_h_pos) = '1' then
                                        one <= '1';
                                else
                                        one <= '0';
                                end if;
                        else
                                one <= '0';
                        end if;
                end if;
  end process Number1Gen;

    -- Number 2 generation
        Number2Gen : process (clk)
          variable sprite_h_pos, sprite_v_pos : integer;
          begin
```

```
            if rising_edge(clk) then
                  if reset = '1' then
                        two <= '0';
                  elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = TWO_CODE
then

                              sprite_h_pos := inner_tile_h_pos;
                              sprite_v_pos := inner_tile_v_pos;

                         if sprite_2(sprite_v_pos)(sprite_h_pos) = '1' then
                              two <= '1';
                         else
                              two <= '0';
                         end if;
                  else
                        two <= '0';
                  end if;
            end if;
  end process Number2Gen;

      -- letter P generation
      LetterPGen : process (clk)
        variable sprite_h_pos, sprite_v_pos : integer;
        begin
            if rising_edge(clk) then
                  if reset = '1' then
                        P <= '0';
                  elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = P_CODE then

                              sprite_h_pos := inner_tile_h_pos;
                              sprite_v_pos := inner_tile_v_pos;

                         if sprite_P(sprite_v_pos)(sprite_h_pos) = '1' then
                              P <= '1';
                         else
                              P <= '0';
                         end if;
                  else
                        P <= '0';
                  end if;
```

```
            end if;
end process LetterPGen;

 -- Letter W generation
     LetterWGen : process (clk)
       variable sprite_h_pos, sprite_v_pos : integer;
       begin
            if rising_edge(clk) then
                   if reset = '1' then
                          W <= '0';
                   elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = W_CODE then

                                 sprite_h_pos := inner_tile_h_pos;
                                 sprite_v_pos := inner_tile_v_pos;

                          if sprite_W(sprite_v_pos)(sprite_h_pos) = '1' then
                                 W <= '1';
                          else
                                 W <= '0';
                          end if;
                   else
                          W <= '0';
                   end if;
            end if;
end process LetterWGen;

 -- Letter I generation
     LetterIGen : process (clk)
       variable sprite_h_pos, sprite_v_pos : integer;
       begin
            if rising_edge(clk) then
                   if reset = '1' then
                          I <= '0';
                   elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = I_CODE then

                                 sprite_h_pos := inner_tile_h_pos;
                                 sprite_v_pos := inner_tile_v_pos;

                          if sprite_I(sprite_v_pos)(sprite_h_pos) = '1' then
                                 I <= '1';
```

```
                        else
                              I <= '0';
                        end if;
                  else
                        I <= '0';
                  end if;
            end if;
end process LetterIGen;

 -- Letter N generation
     LetterNGen : process (clk)
       variable sprite_h_pos, sprite_v_pos : integer;
       begin
            if rising_edge(clk) then
                  if reset = '1' then
                        N <= '0';
                  elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = N_CODE then

                              sprite_h_pos := inner_tile_h_pos;
                              sprite_v_pos := inner_tile_v_pos;

                        if sprite_N(sprite_v_pos)(sprite_h_pos) = '1' then
                              N <= '1';
                        else
                              N <= '0';
                        end if;
                  else
                        N <= '0';
                  end if;
            end if;
end process LetterNGen;

 -- Letter S generation
     LetterSGen : process (clk)
       variable sprite_h_pos, sprite_v_pos : integer;
       begin
            if rising_edge(clk) then
                  if reset = '1' then
                        S <= '0';
                  elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = S_CODE then
```

```
                              sprite_h_pos := inner_tile_h_pos;
                              sprite_v_pos := inner_tile_v_pos;

                        if sprite_S(sprite_v_pos)(sprite_h_pos) = '1' then
                              S <= '1';
                        else
                              S <= '0';
                        end if;
                  else
                        S <= '0';
                  end if;
            end if;
end process LetterSGen;

 -- Letter T generation
     LetterTGen : process (clk)
       variable sprite_h_pos, sprite_v_pos : integer;
       begin
            if rising_edge(clk) then
                  if reset = '1' then
                        T <= '0';
                  elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = T_CODE then

                              sprite_h_pos := inner_tile_h_pos;
                              sprite_v_pos := inner_tile_v_pos;

                        if sprite_T(sprite_v_pos)(sprite_h_pos) = '1' then
                              T <= '1';
                        else
                              T <= '0';
                        end if;
                  else
                        T <= '0';
                  end if;
            end if;
end process LetterTGen;

 -- Letter E generation
     LetterEGen : process (clk)
```

```
        variable sprite_h_pos, sprite_v_pos : integer;
        begin
            if rising_edge(clk) then
                if reset = '1' then
                    E <= '0';
                elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = E_CODE then

                        sprite_h_pos := inner_tile_h_pos;
                        sprite_v_pos := inner_tile_v_pos;

                    if sprite_E(sprite_v_pos)(sprite_h_pos) = '1' then
                        E <= '1';
                    else
                        E <= '0';
                    end if;
                else
                    E <= '0';
                end if;
            end if;
    end process LetterEGen;

    -- Exclamation point generation
        ExclamGen : process (clk)
        variable sprite_h_pos, sprite_v_pos : integer;
        begin
            if rising_edge(clk) then
                if reset = '1' then
                    exclam <= '0';
                elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = EXC_CODE
then

                        sprite_h_pos := inner_tile_h_pos;
                        sprite_v_pos := inner_tile_v_pos;

                    if sprite_exclam(sprite_v_pos)(sprite_h_pos) = '1' then
                        exclam <= '1';
                    else
                        exclam <= '0';
                    end if;
                else
```

```
                        exclam <= '0';
                    end if;
                end if;
    end process ExclamGen;

    -- Play button sprite generation
        PlayButtonSpriteGen : process (clk)
          variable sprite_h_pos, sprite_v_pos : integer;
          begin
                if rising_edge(clk) then
                    if reset = '1' then
                        play <= '0';
                    elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = PLAY_CODE
then

                            sprite_h_pos := inner_tile_h_pos;
                            sprite_v_pos := inner_tile_v_pos;

                         if sprite_play(sprite_v_pos)(sprite_h_pos) = '1' then
                            play <= '1';
                         else
                            play <= '0';
                         end if;
                    else
                        play <= '0';
                    end if;
                end if;
    end process PlayButtonSpriteGen;

    -- Pause button sprite generation
        PauseButtonSpriteGen : process (clk)
          variable sprite_h_pos, sprite_v_pos : integer;
          begin
                if rising_edge(clk) then
                    if reset = '1' then
                        pause <= '0';
                    elsif sprite_select(7) = '1' and sprite_select(4 downto 0) = PAUSE_CODE
then

                            sprite_h_pos := inner_tile_h_pos;
```

```
                                    sprite_v_pos := inner_tile_v_pos;

                                    if sprite_pause(sprite_v_pos)(sprite_h_pos) = '1' then
                                          pause <= '1';
                                    else
                                          pause <= '0';
                                    end if;
                              else
                                    pause <= '0';
                              end if;
                        end if;
  end process PauseButtonSpriteGen;




  ------------------------------------------------------------------------
  ------ SPLASH SCREEN STUFF -------------------------------------------------
  ------------------------------------------------------------------------

       -- Get the point on the screen to fetch from memory
  splash_sprite_h_pos        <= (Hcount) - (HSYNC + HBACK_PORCH + SPLASH_SNAKE_START_H);
  splash_sprite_v_pos        <= (Vcount) - (VSYNC + VBACK_PORCH - 1 + SPLASH_SNAKE_START_V);
  splash_sprite_h_pos_16     <= "00000000" & std_logic_vector(splash_sprite_h_pos(7 downto
0));
  splash_sprite_v_pos_16     <= std_logic_vector(splash_sprite_v_pos(7 downto 0)) &
"00000000";
  splash_snake_address              <= std_logic_vector( unsigned(splash_sprite_v_pos_16) +
unsigned(splash_sprite_h_pos_16) ) ;

  SplashScreenEnableGen : process (clk)
  begin
              if rising_edge(clk) then
                    if reset = '1' then
                          splash_snake_address_enable <= '0';
                          splash_title_address_enable <= '0';
                    else
                          splash_snake_address_enable <= '0';
                          splash_title_address_enable <= '0';
```

```
                        if controller_enable_splash_screen = '1' or (sprite_select(7) =
'1' and sprite_select(4 downto 0) = SPLASH_SNAKE_CODE) then

                                if Hcount >= (HSYNC + HBACK_PORCH + SPLASH_SNAKE_START_H)
and Hcount < (HSYNC + HBACK_PORCH + SPLASH_SNAKE_START_H + SPLASH_SNAKE_SIZE) and
                                        Vcount >= (VSYNC + VBACK_PORCH - 1 +
SPLASH_SNAKE_START_V) and Vcount < (VSYNC + VBACK_PORCH - 1 + SPLASH_SNAKE_START_V +
SPLASH_SNAKE_SIZE) then
                                        splash_snake_address_enable <= '1';
                                end if;

                                if Hcount >= (HSYNC + HBACK_PORCH + SPLASH_TITLE_START_H)
and Hcount < (HSYNC + HBACK_PORCH + SPLASH_TITLE_START_H + SPLASH_TITLE_WIDTH) and
                                        Vcount >= (VSYNC + VBACK_PORCH - 1 +
SPLASH_TITLE_START_V) and Vcount < (VSYNC + VBACK_PORCH - 1 + SPLASH_TITLE_START_V +
SPLASH_TITLE_HEIGHT) then
                                        splash_title_address_enable <= '1';
                                end if;

                        end if;

                end if;-- end if;
        end if;-- end rising edge
    end process SplashScreenEnableGen;

  -- splash screen Snake generation
  SplashScreenSnakeGen : process (clk)
  begin
      if rising_edge(clk) then
            if reset = '1' then
                  splash_snake_black    <= '0';
                  splash_snake_green    <= '0';
                  splash_snake_yellow   <= '0';
                  splash_snake_red            <= '0';
            elsif splash_snake_address_enable = '1' then

                  splash_snake_black    <= '0';
                  splash_snake_green    <= '0';
                  splash_snake_yellow   <= '0';
```

```
                splash_snake_red            <= '0';

                if splash_snake_data = "001" then
                        splash_snake_black <= '1';
                elsif splash_snake_data = "010" then
                        splash_snake_green <= '1';
                elsif splash_snake_data = "011" then
                        splash_snake_yellow <= '1';
                elsif splash_snake_data = "100" then
                        splash_snake_red <= '1';
                end if;
        else
                splash_snake_black    <= '0';
                splash_snake_green    <= '0';
                splash_snake_yellow   <= '0';
                splash_snake_red            <= '0';
        end if;
    end if;
end process SplashScreenSnakeGen;

  -- splash screen Title generation
SplashScreenTitleGen : process (clk)
begin
    if rising_edge(clk) then
        if reset = '1' then
                splash_title_green    <= '0';
        elsif splash_title_address_enable = '1'  then

                splash_title_green <= '0';

                if splash_title_data = "1" then
                        splash_title_green <= '1';
                end if;
        else
                splash_title_green <= '0';
        end if;
    end if;
end process SplashScreenTitleGen;
```

```
  -- Registered video signals going to the video DAC
  VideoOut: process (clk, reset)
  begin
    if reset = '1' then
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "0000000000";
    elsif clk'event and clk = '1' then

              if blue = '1' or rabbit_b = '1'
                                                or mouse_b_eye = '1' then
                VGA_R <= "0000000000";
                VGA_G <= "1011001000";
                VGA_B <= "1110111000";

              --This got changed to orange, was green, now its orange
              elsif green = '1' or ed_b_eye = '1' or
                           (player_select = '0' and (snake_body_orange = '1' or
snake_head_orange = '1' or snake_turn_orange = '1' or snake_tail_orange = '1'))
                           or splash_title_green = '1'
                            then
                VGA_R <= "1111111111";
                VGA_G <= "0010100000";
                VGA_B <= "0000000000";
              elsif freeze = '1' then
                VGA_R <= "1110000000";
                VGA_G <= "1111111111";
                VGA_B <= "1111111111";
              elsif (player_select = '1' and (snake_body_orange = '1' or snake_head_orange =
'1' or snake_turn_orange = '1'or snake_tail_orange = '1')) or
                           splash_snake_green = '1' then
                      VGA_R <= "0000000000";
                      VGA_G <= "1111111111";
                      VGA_B <= "0000000000";
              elsif splash_snake_red = '1' then
                      VGA_R <= "0100100000";
                      VGA_G <= "0100101100";
                      VGA_B <= "0100100000";
              elsif red = '1' or wall = '1' or growth_r = '1'  then
                VGA_R <= "1111111111";
```

```
                VGA_G <= "0000000000";
                VGA_B <= "0000000000";
              elsif black = '1' or snake_head_black = '1' or snake_tail_black = '1' or
snake_body_black ='1'
                                                    or snake_turn_black = '1' or mouse_l
= '1' or edwards_bl = '1' or splash_snake_black = '1' then
                VGA_R <= "0000000000";
                VGA_G <= "0000000000";
                VGA_B <= "0000000000";
              elsif white = '1' or rabbit_w = '1' or snake_body_white = '1' or
snake_turn_white = '1'
                                                    or one = '1' or P = '1' or two = '1'
                                                    or I = '1' or N = '1' or S = '1' or
T = '1' or E = '1'
                                                    or exclam = '1' or pause = '1' or
ed_w_eye = '1'
                                                    or play = '1'  or W = '1' or
mouse_w_eye = '1' then
                VGA_R <= "1111111111";
                VGA_G <= "1111111111";
                VGA_B <= "1111111111";
              elsif pink = '1' or rabbit_p = '1' or mouse_p = '1'
                                                    or edwards_p = '1' then
                VGA_R <= "1111111111";
                VGA_G <= "0110000000";
                VGA_B <= "0110100000";
              elsif gray = '1'  or snake_body_grey = '1' or snake_turn_grey = '1' or rabbit_y
= '1' or growth_y = '1'
                                                    or mouse_y = '1' then
                VGA_R <= "1100000000";
                VGA_G <= "1100000000";
                VGA_B <= "1100000000";
              elsif yellow = '1' or speed = '1' or snake_tail_yellow = '1' or
splash_snake_yellow = '1' then
                VGA_R <= "1111111111";
                VGA_G <= "1111111111";
                VGA_B <= "0000000000";
              elsif brown = '1' or edwards_br = '1' then
                VGA_R <= "1000010010";
                VGA_G <= "0100001000";
                VGA_B <= "0000100110";
```

```
            elsif tan = '1' or edwards_t = '1' then
              VGA_R <= "1111101000";
              VGA_G <= "1110011000";
              VGA_B <= "1000110000";
    elsif vga_hblank = '0' and vga_vblank ='0' then
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "0000000000";
    else -- black
      VGA_R <= "0000000000";
      VGA_G <= "0000000000";
      VGA_B <= "0000000000";
    end if;
  end if;
end process VideoOut;

VGA_CLK <= clk;
VGA_HS <= not vga_hsync;
VGA_VS <= not vga_vsync;
VGA_SYNC <= '0';
VGA_BLANK <= not (vga_hsync or vga_vsync);




-- Sprite Definitions

--Snake head
sprite_head_down_black(0)               <=      "1000000000111000";
sprite_head_down_black(1)               <=      "1000000000001100";
sprite_head_down_black(2)               <=      "1000000000001000";
sprite_head_down_black(3)               <=      "1000000000001000";
sprite_head_down_black(4)               <=      "1000000001001110";
sprite_head_down_black(5)               <=      "1000000000101110";
sprite_head_down_black(6)               <=      "1000001000110000";
sprite_head_down_black(7)               <=      "0100001000111011";
sprite_head_down_black(8)               <=      "0100001000011111";
sprite_head_down_black(9)               <=      "0000001000011101";
sprite_head_down_black(10)              <=      "0010010000001001";
```

```
sprite_head_down_black(11)              <=      "0001000000001101";
sprite_head_down_black(12)              <=      "0000010000100100";
sprite_head_down_black(13)              <=      "0000001000000100";
sprite_head_down_black(14)              <=      "0000000010001000";
sprite_head_down_black(15)              <=      "0000000000000000";

sprite_head_down_orange(0)              <=      "0111111111000100";
sprite_head_down_orange(1)              <=      "0111111111110000";
sprite_head_down_orange(2)              <=      "0111111111110100";
sprite_head_down_orange(3)              <=      "0111111111110100";
sprite_head_down_orange(4)              <=      "0111111110110000";
sprite_head_down_orange(5)              <=      "0111111111010000";
sprite_head_down_orange(6)              <=      "0111110111001110";
sprite_head_down_orange(7)              <=      "0011110111000100";
sprite_head_down_orange(8)              <=      "0011110111100000";
sprite_head_down_orange(9)              <=      "0011110111100010";
sprite_head_down_orange(10)         <=      "0001101111110010";
sprite_head_down_orange(11)         <=      "0000111111110010";
sprite_head_down_orange(12)         <=      "0000001111011010";
sprite_head_down_orange(13)         <=      "0000000111111000";
sprite_head_down_orange(14)         <=      "0000000001110000";
sprite_head_down_orange(15)         <=      "0000000000000000";


sprite_head_right_black(0)              <=      "1111111000000000";
sprite_head_right_black(1)              <=      "0000000110000000";
sprite_head_right_black(2)              <=      "0000000000100000";
sprite_head_right_black(3)              <=      "0000000000010000";
sprite_head_right_black(4)              <=      "0000000000000000";
sprite_head_right_black(5)              <=      "0000000000101000";
sprite_head_right_black(6)              <=      "0000001111000100";
sprite_head_right_black(7)              <=      "0000000000000000";
sprite_head_right_black(8)              <=      "0000000000000010";
sprite_head_right_black(9)              <=      "0000100000000000";
sprite_head_right_black(10)         <=      "1000011100001000";
sprite_head_right_black(11)         <=      "1000001111000000";
sprite_head_right_black(12)         <=      "1111110111110010";
sprite_head_right_black(13)         <=      "0100110011011100";
sprite_head_right_black(14)         <=      "0000110110000000";
sprite_head_right_black(15)         <=      "0000000111110000";
```

```
sprite_head_right_orange(0)         <=      "0000000000000000";
sprite_head_right_orange(1)         <=      "1111111000000000";
sprite_head_right_orange(2)         <=      "1111111111000000";
sprite_head_right_orange(3)         <=      "1111111111100000";
sprite_head_right_orange(4)         <=      "1111111111110000";
sprite_head_right_orange(5)         <=      "1111111111010000";
sprite_head_right_orange(6)         <=      "1111110000111000";
sprite_head_right_orange(7)         <=      "1111111111111100";
sprite_head_right_orange(8)         <=      "1111111111111100";
sprite_head_right_orange(9)         <=      "1111011111111110";
sprite_head_right_orange(10)        <=      "0111100011110110";
sprite_head_right_orange(11)        <=      "0111110000111110";
sprite_head_right_orange(12)        <=      "0000001000001100";
sprite_head_right_orange(13)        <=      "1011001100000000";
sprite_head_right_orange(14)        <=      "0000001001111000";
sprite_head_right_orange(15)        <=      "0000000000000000";

sprite_head_up_black(0)                     <=      "0000000000000000";
sprite_head_up_black(1)                     <=      "0000000010001000";
sprite_head_up_black(2)                     <=      "0000001000000100";
sprite_head_up_black(3)                     <=      "0000010000100100";
sprite_head_up_black(4)                     <=      "0001000000001101";
sprite_head_up_black(5)                     <=      "0010010000001001";
sprite_head_up_black(6)                     <=      "0000001000011101";
sprite_head_up_black(7)                     <=      "0100001000011111";
sprite_head_up_black(8)                     <=      "0100001000111011";
sprite_head_up_black(9)                     <=      "1000001000110000";
sprite_head_up_black(10)                <=      "1000000000101110";
sprite_head_up_black(11)                <=      "1000000001001110";
sprite_head_up_black(12)                <=      "1000000000001000";
sprite_head_up_black(13)                <=      "1000000000001000";
sprite_head_up_black(14)                <=      "1000000000001100";
sprite_head_up_black(15)                <=      "1000000000111000";

sprite_head_up_orange(0)                <=      "0000000000000000";
sprite_head_up_orange(1)                <=      "0000000001110000";
sprite_head_up_orange(2)                <=      "0000000111111000";
sprite_head_up_orange(3)                <=      "0000001111011010";
sprite_head_up_orange(4)                <=      "0000111111110010";
```

```
sprite_head_up_orange(5)                <=      "0001101111110010";
sprite_head_up_orange(6)                <=      "0011110111100010";
sprite_head_up_orange(7)                <=      "0011110111100000";
sprite_head_up_orange(8)                <=      "0011110111000100";
sprite_head_up_orange(9)                <=      "0111110111001110";
sprite_head_up_orange(10)               <=      "0111111111010000";
sprite_head_up_orange(11)               <=      "0111111110110000";
sprite_head_up_orange(12)               <=      "0111111111110100";
sprite_head_up_orange(13)               <=      "0111111111110100";
sprite_head_up_orange(14)               <=      "0111111111110000";
sprite_head_up_orange(15)               <=      "0111111111000100";

sprite_head_left_black(0)               <=      "0000000001111111";
sprite_head_left_black(1)               <=      "0000000110000000";
sprite_head_left_black(2)               <=      "0000010000000000";
sprite_head_left_black(3)               <=      "0000100000000000";
sprite_head_left_black(4)               <=      "0000000000000000";
sprite_head_left_black(5)               <=      "0001010000000000";
sprite_head_left_black(6)               <=      "0010001111000000";
sprite_head_left_black(7)               <=      "0000000000000000";
sprite_head_left_black(8)               <=      "0100000000000000";
sprite_head_left_black(9)               <=      "0000000000010000";
sprite_head_left_black(10)              <=      "0001000011100001";
sprite_head_left_black(11)              <=      "0000001111000001";
sprite_head_left_black(12)              <=      "0100111110111111";
sprite_head_left_black(13)              <=      "0011101100110010";
sprite_head_left_black(14)              <=      "0000000110110000";
sprite_head_left_black(15)              <=      "0000111110000000";

sprite_head_left_orange(0)              <=      "0000000000000000";
sprite_head_left_orange(1)              <=      "0000000001111111";
sprite_head_left_orange(2)              <=      "0000001111111111";
sprite_head_left_orange(3)              <=      "0000011111111111";
sprite_head_left_orange(4)              <=      "0000111111111111";
sprite_head_left_orange(5)              <=      "0000101111111111";
sprite_head_left_orange(6)              <=      "0001110000111111";
sprite_head_left_orange(7)              <=      "0011111111111111";
sprite_head_left_orange(8)              <=      "0011111111111111";
sprite_head_left_orange(9)              <=      "0111111111101111";
sprite_head_left_orange(10)         <=      "0110111100011110";
```

```
sprite_head_left_orange(11)         <=      "0111110000111110";
sprite_head_left_orange(12)         <=      "0011000001000000";
sprite_head_left_orange(13)         <=      "0000000011001101";
sprite_head_left_orange(14)         <=      "0001111001000000";
sprite_head_left_orange(15)         <=      "0000000000000000";
```

```
--sprite body coloring
sprite_body_right_black(0)              <=      "0000000110000000";
sprite_body_right_black(1)              <=      "0000000000000000";
sprite_body_right_black(2)              <=      "0000000000000000";
sprite_body_right_black(3)              <=      "0000000000000000";
sprite_body_right_black(4)              <=      "0000000000000000";
sprite_body_right_black(5)              <=      "0000000000000000";
sprite_body_right_black(6)              <=      "0000000000000000";
sprite_body_right_black(7)              <=      "0000000000000000";
sprite_body_right_black(8)              <=      "0000000000000000";
sprite_body_right_black(9)              <=      "0000000000000000";
sprite_body_right_black(10)         <=      "0000000000000000";
sprite_body_right_black(11)         <=      "0000000000000000";
sprite_body_right_black(12)         <=      "1111111111111111";
sprite_body_right_black(13)         <=      "1111111111111111";
sprite_body_right_black(14)         <=      "0000000000000000";
sprite_body_right_black(15)         <=      "1111111111111111";

sprite_body_right_grey(0)               <=      "1111111000000000";
sprite_body_right_grey(1)               <=      "0000000000000000";
sprite_body_right_grey(2)               <=      "1111111100000000";
sprite_body_right_grey(3)               <=      "1111111100000000";
sprite_body_right_grey(4)               <=      "1111111100000000";
sprite_body_right_grey(5)               <=      "1111111100000000";
sprite_body_right_grey(6)               <=      "1111111100000000";
sprite_body_right_grey(7)               <=      "1111111100000000";
sprite_body_right_grey(8)               <=      "1111111100000000";
sprite_body_right_grey(9)               <=      "1111111100000000";
sprite_body_right_grey(10)              <=      "1111111100000000";
sprite_body_right_grey(11)              <=      "1111111100000000";
```

```
sprite_body_right_grey(12)              <=      "0000000000000000";
sprite_body_right_grey(13)              <=      "0000000000000000";
sprite_body_right_grey(14)              <=      "1111111100000000";
sprite_body_right_grey(15)              <=      "0000000000000000";

sprite_body_right_orange(0)         <=      "0000000001111111";
sprite_body_right_orange(1)         <=      "0000000000000000";
sprite_body_right_orange(2)         <=      "0000000011111111";
sprite_body_right_orange(3)         <=      "0000000011111111";
sprite_body_right_orange(4)         <=      "0000000011111111";
sprite_body_right_orange(5)         <=      "0000000011111111";
sprite_body_right_orange(6)         <=      "0000000011111111";
sprite_body_right_orange(7)         <=      "0000000011111111";
sprite_body_right_orange(8)         <=      "0000000011111111";
sprite_body_right_orange(9)         <=      "0000000011111111";
sprite_body_right_orange(10)        <=      "0000000011111111";
sprite_body_right_orange(11)        <=      "0000000011111111";
sprite_body_right_orange(12)        <=      "0000000000000000";
sprite_body_right_orange(13)        <=      "0000000000000000";
sprite_body_right_orange(14)        <=      "0000000011111111";
sprite_body_right_orange(15)        <=      "0000000000000000";

sprite_body_right_white(0)              <=      "0000000000000000";
sprite_body_right_white(1)              <=      "1111111111111111";
sprite_body_right_white(2)              <=      "0000000000000000";
sprite_body_right_white(3)              <=      "0000000000000000";
sprite_body_right_white(4)              <=      "0000000000000000";
sprite_body_right_white(5)              <=      "0000000000000000";
sprite_body_right_white(6)              <=      "0000000000000000";
sprite_body_right_white(7)              <=      "0000000000000000";
sprite_body_right_white(8)              <=      "0000000000000000";
sprite_body_right_white(9)              <=      "0000000000000000";
sprite_body_right_white(10)         <=      "0000000000000000";
sprite_body_right_white(11)         <=      "0000000000000000";
sprite_body_right_white(12)         <=      "0000000000000000";
sprite_body_right_white(13)         <=      "0000000000000000";
sprite_body_right_white(14)         <=      "0000000000000000";
sprite_body_right_white(15)         <=      "0000000000000000";
```

```
sprite_body_up_black(0)                 <=      "0000000000001101";
sprite_body_up_black(1)                 <=      "0000000000001101";
sprite_body_up_black(2)                 <=      "0000000000001101";
sprite_body_up_black(3)                 <=      "0000000000001101";
sprite_body_up_black(4)                 <=      "0000000000001101";
sprite_body_up_black(5)                 <=      "0000000000001101";
sprite_body_up_black(6)                 <=      "0000000000001101";
sprite_body_up_black(7)                 <=      "1000000000001101";
sprite_body_up_black(8)                 <=      "1000000000001101";
sprite_body_up_black(9)                 <=      "0000000000001101";
sprite_body_up_black(10)            <=      "0000000000001101";
sprite_body_up_black(11)            <=      "0000000000001101";
sprite_body_up_black(12)            <=      "0000000000001101";
sprite_body_up_black(13)            <=      "0000000000001101";
sprite_body_up_black(14)            <=      "0000000000001101";
sprite_body_up_black(15)            <=      "0000000000001101";

sprite_body_up_grey(0)                  <=      "0000000000000000";
sprite_body_up_grey(1)                  <=      "0000000000000000";
sprite_body_up_grey(2)                  <=      "0000000000000000";
sprite_body_up_grey(3)                  <=      "0000000000000000";
sprite_body_up_grey(4)                  <=      "0000000000000000";
sprite_body_up_grey(5)                  <=      "0000000000000000";
sprite_body_up_grey(6)                  <=      "0000000000000000";
sprite_body_up_grey(7)                  <=      "0000000000000000";
sprite_body_up_grey(8)                  <=      "0011111111110010";
sprite_body_up_grey(9)                  <=      "1011111111110010";
sprite_body_up_grey(10)                 <=      "1011111111110010";
sprite_body_up_grey(11)                 <=      "1011111111110010";
sprite_body_up_grey(12)                 <=      "1011111111110010";
sprite_body_up_grey(13)                 <=      "1011111111110010";
sprite_body_up_grey(14)                 <=      "1011111111110010";
sprite_body_up_grey(15)                 <=      "1011111111110010";

sprite_body_up_orange(0)            <=      "1011111111110010";
sprite_body_up_orange(1)            <=      "1011111111110010";
sprite_body_up_orange(2)            <=      "1011111111110010";
sprite_body_up_orange(3)            <=      "1011111111110010";
sprite_body_up_orange(4)            <=      "1011111111110010";
sprite_body_up_orange(5)            <=      "1011111111110010";
```

```
sprite_body_up_orange(6)                <=      "1011111111110010";
sprite_body_up_orange(7)                <=      "0011111111110010";
sprite_body_up_orange(8)                <=      "0000000000000000";
sprite_body_up_orange(9)                <=      "0000000000000000";
sprite_body_up_orange(10)               <=      "0000000000000000";
sprite_body_up_orange(11)               <=      "0000000000000000";
sprite_body_up_orange(12)               <=      "0000000000000000";
sprite_body_up_orange(13)               <=      "0000000000000000";
sprite_body_up_orange(14)               <=      "0000000000000000";
sprite_body_up_orange(15)               <=      "0000000000000000";

sprite_body_up_white(0)                         <=      "0100000000000000";
sprite_body_up_white(1)                         <=      "0100000000000000";
sprite_body_up_white(2)                         <=      "0100000000000000";
sprite_body_up_white(3)                         <=      "0100000000000000";
sprite_body_up_white(4)                         <=      "0100000000000000";
sprite_body_up_white(5)                         <=      "0100000000000000";
sprite_body_up_white(6)                         <=      "0100000000000000";
sprite_body_up_white(7)                         <=      "0100000000000000";
sprite_body_up_white(8)                         <=      "0100000000000000";
sprite_body_up_white(9)                         <=      "0100000000000000";
sprite_body_up_white(10)                <=      "0100000000000000";
sprite_body_up_white(11)                <=      "0100000000000000";
sprite_body_up_white(12)                <=      "0100000000000000";
sprite_body_up_white(13)                <=      "0100000000000000";
sprite_body_up_white(14)                <=      "0100000000000000";
sprite_body_up_white(15)                <=      "0100000000000000";

sprite_body_left_black(0)               <=      "1111111111111111";
sprite_body_left_black(1)               <=      "0000000000000000";
sprite_body_left_black(2)               <=      "1111111111111111";
sprite_body_left_black(3)               <=      "1111111111111111";
sprite_body_left_black(4)               <=      "0000000000000000";
sprite_body_left_black(5)               <=      "0000000000000000";
sprite_body_left_black(6)               <=      "0000000000000000";
sprite_body_left_black(7)               <=      "0000000000000000";
sprite_body_left_black(8)               <=      "0000000000000000";
sprite_body_left_black(9)               <=      "0000000000000000";
sprite_body_left_black(10)              <=      "0000000000000000";
sprite_body_left_black(11)              <=      "0000000000000000";
```

```
sprite_body_left_black(12)              <=      "0000000000000000";
sprite_body_left_black(13)              <=      "0000000000000000";
sprite_body_left_black(14)              <=      "0000000000000000";
sprite_body_left_black(15)              <=      "0000000110000000";

sprite_body_left_grey(0)                <=      "0000000000000000";
sprite_body_left_grey(1)                <=      "0000000011111111";
sprite_body_left_grey(2)                <=      "0000000000000000";
sprite_body_left_grey(3)                <=      "0000000000000000";
sprite_body_left_grey(4)                <=      "0000000011111111";
sprite_body_left_grey(5)                <=      "0000000011111111";
sprite_body_left_grey(6)                <=      "0000000011111111";
sprite_body_left_grey(7)                <=      "0000000011111111";
sprite_body_left_grey(8)                <=      "0000000011111111";
sprite_body_left_grey(9)                <=      "0000000011111111";
sprite_body_left_grey(10)               <=      "0000000011111111";
sprite_body_left_grey(11)               <=      "0000000011111111";
sprite_body_left_grey(12)               <=      "0000000011111111";
sprite_body_left_grey(13)               <=      "0000000011111111";
sprite_body_left_grey(14)               <=      "0000000000000000";
sprite_body_left_grey(15)               <=      "0000000001111111";

sprite_body_left_orange(0)              <=      "0000000000000000";
sprite_body_left_orange(1)              <=      "1111111100000000";
sprite_body_left_orange(2)              <=      "0000000000000000";
sprite_body_left_orange(3)              <=      "0000000000000000";
sprite_body_left_orange(4)              <=      "1111111100000000";
sprite_body_left_orange(5)              <=      "1111111100000000";
sprite_body_left_orange(6)              <=      "1111111100000000";
sprite_body_left_orange(7)              <=      "1111111100000000";
sprite_body_left_orange(8)              <=      "1111111100000000";
sprite_body_left_orange(9)              <=      "1111111100000000";
sprite_body_left_orange(10)         <=      "1111111100000000";
sprite_body_left_orange(11)         <=      "1111111100000000";
sprite_body_left_orange(12)         <=      "1111111100000000";
sprite_body_left_orange(13)         <=      "1111111100000000";
sprite_body_left_orange(14)         <=      "0000000000000000";
sprite_body_left_orange(15)         <=      "1111111000000000";

sprite_body_left_white(0)               <=      "0000000000000000";
```

```
sprite_body_left_white(1)               <=      "0000000000000000";
sprite_body_left_white(2)               <=      "0000000000000000";
sprite_body_left_white(3)               <=      "0000000000000000";
sprite_body_left_white(4)               <=      "0000000000000000";
sprite_body_left_white(5)               <=      "0000000000000000";
sprite_body_left_white(6)               <=      "0000000000000000";
sprite_body_left_white(7)               <=      "0000000000000000";
sprite_body_left_white(8)               <=      "0000000000000000";
sprite_body_left_white(9)               <=      "0000000000000000";
sprite_body_left_white(10)              <=      "0000000000000000";
sprite_body_left_white(11)              <=      "0000000000000000";
sprite_body_left_white(12)              <=      "0000000000000000";
sprite_body_left_white(13)              <=      "0000000000000000";
sprite_body_left_white(14)              <=      "1111111111111111";
sprite_body_left_white(15)              <=      "0000000000000000";

sprite_body_down_black(0)               <=      "1011000000000000";
sprite_body_down_black(1)               <=      "1011000000000000";
sprite_body_down_black(2)               <=      "1011000000000000";
sprite_body_down_black(3)               <=      "1011000000000000";
sprite_body_down_black(4)               <=      "1011000000000000";
sprite_body_down_black(5)               <=      "1011000000000000";
sprite_body_down_black(6)               <=      "1011000000000000";
sprite_body_down_black(7)               <=      "1011000000000001";
sprite_body_down_black(8)               <=      "1011000000000001";
sprite_body_down_black(9)               <=      "1011000000000000";
sprite_body_down_black(10)              <=      "1011000000000000";
sprite_body_down_black(11)              <=      "1011000000000000";
sprite_body_down_black(12)              <=      "1011000000000000";
sprite_body_down_black(13)              <=      "1011000000000000";
sprite_body_down_black(14)              <=      "1011000000000000";
sprite_body_down_black(15)              <=      "1011000000000000";

sprite_body_down_grey(0)                <=      "0100111111111101";
sprite_body_down_grey(1)                <=      "0100111111111101";
sprite_body_down_grey(2)                <=      "0100111111111101";
sprite_body_down_grey(3)                <=      "0100111111111101";
sprite_body_down_grey(4)                <=      "0100111111111101";
sprite_body_down_grey(5)                <=      "0100111111111101";
sprite_body_down_grey(6)                <=      "0100111111111101";
```

```
sprite_body_down_grey(7)                <=      "0100111111111100";
sprite_body_down_grey(8)                <=      "0000000000000000";
sprite_body_down_grey(9)                <=      "0000000000000000";
sprite_body_down_grey(10)               <=      "0000000000000000";
sprite_body_down_grey(11)               <=      "0000000000000000";
sprite_body_down_grey(12)               <=      "0000000000000000";
sprite_body_down_grey(13)               <=      "0000000000000000";
sprite_body_down_grey(14)               <=      "0000000000000000";
sprite_body_down_grey(15)               <=      "0000000000000000";

sprite_body_down_orange(0)              <=      "0000000000000000";
sprite_body_down_orange(1)              <=      "0000000000000000";
sprite_body_down_orange(2)              <=      "0000000000000000";
sprite_body_down_orange(3)              <=      "0000000000000000";
sprite_body_down_orange(4)              <=      "0000000000000000";
sprite_body_down_orange(5)              <=      "0000000000000000";
sprite_body_down_orange(6)              <=      "0000000000000000";
sprite_body_down_orange(7)              <=      "0000000000000000";
sprite_body_down_orange(8)              <=      "0100111111111100";
sprite_body_down_orange(9)              <=      "0100111111111101";
sprite_body_down_orange(10)             <=      "0100111111111101";
sprite_body_down_orange(11)             <=      "0100111111111101";
sprite_body_down_orange(12)             <=      "0100111111111101";
sprite_body_down_orange(13)             <=      "0100111111111101";
sprite_body_down_orange(14)             <=      "0100111111111101";
sprite_body_down_orange(15)             <=      "0100111111111101";

sprite_body_down_white(0)               <=      "0000000000000010";
sprite_body_down_white(1)               <=      "0000000000000010";
sprite_body_down_white(2)               <=      "0000000000000010";
sprite_body_down_white(3)               <=      "0000000000000010";
sprite_body_down_white(4)               <=      "0000000000000010";
sprite_body_down_white(5)               <=      "0000000000000010";
sprite_body_down_white(6)               <=      "0000000000000010";
sprite_body_down_white(7)               <=      "0000000000000010";
sprite_body_down_white(8)               <=      "0000000000000010";
sprite_body_down_white(9)               <=      "0000000000000010";
sprite_body_down_white(10)              <=      "0000000000000010";
sprite_body_down_white(11)              <=      "0000000000000010";
sprite_body_down_white(12)              <=      "0000000000000010";
```

```
sprite_body_down_white(13)              <=      "0000000000000010";
sprite_body_down_white(14)              <=      "0000000000000010";
sprite_body_down_white(15)              <=      "0000000000000010";



--sprite turning coloring
sprite_turn_down_left_black(0)                  <=      "0000000000000110";
sprite_turn_down_left_black(1)                  <=      "0000000000000110";
sprite_turn_down_left_black(2)                  <=      "0000000000000010";
sprite_turn_down_left_black(3)                  <=      "0000000000000010";
sprite_turn_down_left_black(4)                  <=      "0000100000000110";
sprite_turn_down_left_black(5)                  <=      "1111000000000010";
sprite_turn_down_left_black(6)                  <=      "0011000111111111";
sprite_turn_down_left_black(7)                  <=      "0000001000000110";
sprite_turn_down_left_black(8)                  <=      "0000001000000110";
sprite_turn_down_left_black(9)                  <=      "0000001000000010";
sprite_turn_down_left_black(10)                 <=      "0000001000000010";
sprite_turn_down_left_black(11)                 <=      "0000001000000110";
sprite_turn_down_left_black(12)                 <=      "0000001000000111";
sprite_turn_down_left_black(13)                 <=      "0000001000001100";
sprite_turn_down_left_black(14)                 <=      "0000001000001001";
sprite_turn_down_left_black(15)                 <=      "1000001000001111";

sprite_turn_down_left_grey(0)               <=      "0000011111111001";
sprite_turn_down_left_grey(1)               <=      "0000011111111001";
sprite_turn_down_left_grey(2)               <=      "0000011111111101";
sprite_turn_down_left_grey(3)               <=      "0000011111111101";
sprite_turn_down_left_grey(4)               <=      "0000000111111001";
sprite_turn_down_left_grey(5)               <=      "0000000111111101";
sprite_turn_down_left_grey(6)               <=      "0000000000000000";
sprite_turn_down_left_grey(7)               <=      "0000000001000000";
sprite_turn_down_left_grey(8)               <=      "0000000001000000";
sprite_turn_down_left_grey(9)               <=      "0000000110000000";
sprite_turn_down_left_grey(10)                  <=      "0000000111101000";
sprite_turn_down_left_grey(11)                  <=      "0000000111010000";
sprite_turn_down_left_grey(12)                  <=      "0000000111101000";
sprite_turn_down_left_grey(13)                  <=      "0000000111110011";
sprite_turn_down_left_grey(14)                  <=      "0000000000000110";
```

```
sprite_turn_down_left_grey(15)                  <=      "0000000111110000";

sprite_turn_down_left_orange(0)                 <=      "0000000000000000";
sprite_turn_down_left_orange(1)                 <=      "0000000000000000";
sprite_turn_down_left_orange(2)                 <=      "0000000000000000";
sprite_turn_down_left_orange(3)                 <=      "0000000000000000";
sprite_turn_down_left_orange(4)                 <=      "1111011000000000";
sprite_turn_down_left_orange(5)                 <=      "0000111000000000";
sprite_turn_down_left_orange(6)                 <=      "1100111000000000";
sprite_turn_down_left_orange(7)                 <=      "1111110110111001";
sprite_turn_down_left_orange(8)                 <=      "1111110110111001";
sprite_turn_down_left_orange(9)                 <=      "1111110001111101";
sprite_turn_down_left_orange(10)            <=      "1111110000010101";
sprite_turn_down_left_orange(11)            <=      "1111110000101001";
sprite_turn_down_left_orange(12)            <=      "1111110000010000";
sprite_turn_down_left_orange(13)            <=      "1111110000000000";
sprite_turn_down_left_orange(14)            <=      "0000000000000000";
sprite_turn_down_left_orange(15)            <=      "0111110000000000";

sprite_turn_down_left_white(0)                  <=      "0000000000000000";
sprite_turn_down_left_white(1)                  <=      "0000000000000000";
sprite_turn_down_left_white(2)                  <=      "0000000000000000";
sprite_turn_down_left_white(3)                  <=      "0000000000000000";
sprite_turn_down_left_white(4)                  <=      "0000000000000000";
sprite_turn_down_left_white(5)                  <=      "0000000000000000";
sprite_turn_down_left_white(6)                  <=      "0000000000000000";
sprite_turn_down_left_white(7)                  <=      "0000000000000000";
sprite_turn_down_left_white(8)                  <=      "0000000000000000";
sprite_turn_down_left_white(9)                  <=      "0000000000000000";
sprite_turn_down_left_white(10)                 <=      "0000000000000000";
sprite_turn_down_left_white(11)                 <=      "0000000000000000";
sprite_turn_down_left_white(12)                 <=      "0000000000000000";
sprite_turn_down_left_white(13)                 <=      "0000000000000000";
sprite_turn_down_left_white(14)                 <=      "1111110111110000";
sprite_turn_down_left_white(15)                 <=      "0000000000000000";

sprite_turn_left_up_black(0)                <=      "1000000000100000";
sprite_turn_left_up_black(1)                <=      "0000000000100000";
sprite_turn_left_up_black(2)                <=      "0000000001100000";
sprite_turn_left_up_black(3)                <=      "0000000001100000";
```

```
sprite_turn_left_up_black(4)              <=      "0000000000010000";
sprite_turn_left_up_black(5)              <=      "0000000000000000";
sprite_turn_left_up_black(6)              <=      "1111111110000000";
sprite_turn_left_up_black(7)              <=      "0000000001000000";
sprite_turn_left_up_black(8)              <=      "0000000001000000";
sprite_turn_left_up_black(9)              <=      "0000000001000000";
sprite_turn_left_up_black(10)             <=      "0000000001000000";
sprite_turn_left_up_black(11)             <=      "0000000001000000";
sprite_turn_left_up_black(12)             <=      "1110000001000000";
sprite_turn_left_up_black(13)             <=      "1011100111010011";
sprite_turn_left_up_black(14)             <=      "1001111111111111";
sprite_turn_left_up_black(15)             <=      "1101000001000000";

sprite_turn_left_up_grey(0)               <=      "0000000000000000";
sprite_turn_left_up_grey(1)               <=      "0000000000000000";
sprite_turn_left_up_grey(2)               <=      "0000000000000000";
sprite_turn_left_up_grey(3)               <=      "0000000000000000";
sprite_turn_left_up_grey(4)               <=      "0000000000000000";
sprite_turn_left_up_grey(5)               <=      "0000000000001111";
sprite_turn_left_up_grey(6)               <=      "0000000000001111";
sprite_turn_left_up_grey(7)               <=      "1011111000111111";
sprite_turn_left_up_grey(8)               <=      "1011111000111111";
sprite_turn_left_up_grey(9)               <=      "1011110110111111";
sprite_turn_left_up_grey(10)              <=      "1011010000111111";
sprite_turn_left_up_grey(11)              <=      "1010100000111111";
sprite_turn_left_up_grey(12)              <=      "0001010000111111";
sprite_turn_left_up_grey(13)              <=      "0100000000101100";
sprite_turn_left_up_grey(14)              <=      "0110000000000000";
sprite_turn_left_up_grey(15)              <=      "0010000000111111";

sprite_turn_left_up_orange(0)             <=      "0011111111010000";
sprite_turn_left_up_orange(1)             <=      "1011111111010000";
sprite_turn_left_up_orange(2)             <=      "1011111110010000";
sprite_turn_left_up_orange(3)             <=      "1011111110010000";
sprite_turn_left_up_orange(4)             <=      "1011111111100000";
sprite_turn_left_up_orange(5)             <=      "1011111111110000";
sprite_turn_left_up_orange(6)             <=      "0000000001110000";
sprite_turn_left_up_orange(7)             <=      "0000000110000000";
sprite_turn_left_up_orange(8)             <=      "0000000110000000";
sprite_turn_left_up_orange(9)             <=      "0000001000000000";
```

```
sprite_turn_left_up_orange(10)                  <=      "0000101110000000";
sprite_turn_left_up_orange(11)                  <=      "0001011110000000";
sprite_turn_left_up_orange(12)                  <=      "0000101110000000";
sprite_turn_left_up_orange(13)                  <=      "0000011000000000";
sprite_turn_left_up_orange(14)                  <=      "0000000000000000";
sprite_turn_left_up_orange(15)                  <=      "0000111110000000";

sprite_turn_left_up_white(0)                <=      "0100000000000000";
sprite_turn_left_up_white(1)                <=      "0100000000000000";
sprite_turn_left_up_white(2)                <=      "0100000000000000";
sprite_turn_left_up_white(3)                <=      "0100000000000000";
sprite_turn_left_up_white(4)                <=      "0100000000000000";
sprite_turn_left_up_white(5)                <=      "0100000000000000";
sprite_turn_left_up_white(6)                <=      "0000000000000000";
sprite_turn_left_up_white(7)                <=      "0100000000000000";
sprite_turn_left_up_white(8)                <=      "0100000000000000";
sprite_turn_left_up_white(9)                <=      "0100000000000000";
sprite_turn_left_up_white(10)               <=      "0100000000000000";
sprite_turn_left_up_white(11)               <=      "0100000000000000";
sprite_turn_left_up_white(12)               <=      "0000000000000000";
sprite_turn_left_up_white(13)               <=      "0000000000000000";
sprite_turn_left_up_white(14)               <=      "0000000000000000";
sprite_turn_left_up_white(15)               <=      "0000000000000000";

sprite_turn_right_down_black(0)                 <=      "0000001000001011";
sprite_turn_right_down_black(1)                 <=      "1111111111111001";
sprite_turn_right_down_black(2)                 <=      "1100101110011101";
sprite_turn_right_down_black(3)                 <=      "0000001000000111";
sprite_turn_right_down_black(4)                 <=      "0000001000000000";
sprite_turn_right_down_black(5)                 <=      "0000001000000000";
sprite_turn_right_down_black(6)                 <=      "0000001000000000";
sprite_turn_right_down_black(7)                 <=      "0000001000000000";
sprite_turn_right_down_black(8)                 <=      "0000001000000000";
sprite_turn_right_down_black(9)                 <=      "0000000111111111";
sprite_turn_right_down_black(10)            <=      "0000000000000000";
sprite_turn_right_down_black(11)            <=      "0000100000000000";
sprite_turn_right_down_black(12)            <=      "0000011000000000";
sprite_turn_right_down_black(13)            <=      "0000011000000000";
sprite_turn_right_down_black(14)            <=      "0000010000000000";
sprite_turn_right_down_black(15)            <=      "0000010000000001";
```

```
sprite_turn_right_down_grey(0)                  <=      "1111110000000100";
sprite_turn_right_down_grey(1)                  <=      "0000000000000110";
sprite_turn_right_down_grey(2)                  <=      "0011010000000010";
sprite_turn_right_down_grey(3)                  <=      "1111110000101000";
sprite_turn_right_down_grey(4)                  <=      "1111110000010101";
sprite_turn_right_down_grey(5)                  <=      "1111110000101101";
sprite_turn_right_down_grey(6)                  <=      "1111110110111101";
sprite_turn_right_down_grey(7)                  <=      "1111110001111101";
sprite_turn_right_down_grey(8)                  <=      "1111110001111101";
sprite_turn_right_down_grey(9)                  <=      "1111000000000000";
sprite_turn_right_down_grey(10)                 <=      "1111000000000000";
sprite_turn_right_down_grey(11)                 <=      "0000000000000000";
sprite_turn_right_down_grey(12)                 <=      "0000000000000000";
sprite_turn_right_down_grey(13)                 <=      "0000000000000000";
sprite_turn_right_down_grey(14)                 <=      "0000000000000000";
sprite_turn_right_down_grey(15)                 <=      "0000000000000000";

sprite_turn_right_down_orange(0)            <=      "0000000111110000";
sprite_turn_right_down_orange(1)            <=      "0000000000000000";
sprite_turn_right_down_orange(2)            <=      "0000000001100000";
sprite_turn_right_down_orange(3)            <=      "0000000111010000";
sprite_turn_right_down_orange(4)            <=      "0000000111101000";
sprite_turn_right_down_orange(5)            <=      "0000000111010000";
sprite_turn_right_down_orange(6)            <=      "0000000001000000";
sprite_turn_right_down_orange(7)            <=      "0000000110000000";
sprite_turn_right_down_orange(8)            <=      "0000000110000000";
sprite_turn_right_down_orange(9)            <=      "0000111000000000";
sprite_turn_right_down_orange(10)           <=      "0000111111111101";
sprite_turn_right_down_orange(11)           <=      "0000011111111101";
sprite_turn_right_down_orange(12)           <=      "0000100111111101";
sprite_turn_right_down_orange(13)           <=      "0000100111111101";
sprite_turn_right_down_orange(14)           <=      "0000101111111101";
sprite_turn_right_down_orange(15)           <=      "0000101111111100";

sprite_turn_right_down_white(0)                 <=      "0000000000000000";
sprite_turn_right_down_white(1)                 <=      "0000000000000000";
sprite_turn_right_down_white(2)                 <=      "0000000000000000";
sprite_turn_right_down_white(3)                 <=      "0000000000000000";
sprite_turn_right_down_white(4)                 <=      "0000000000000010";
```

```
sprite_turn_right_down_white(5)                 <=      "0000000000000010";
sprite_turn_right_down_white(6)                 <=      "0000000000000010";
sprite_turn_right_down_white(7)                 <=      "0000000000000010";
sprite_turn_right_down_white(8)                 <=      "0000000000000010";
sprite_turn_right_down_white(9)                 <=      "0000000000000000";
sprite_turn_right_down_white(10)            <=      "0000000000000010";
sprite_turn_right_down_white(11)            <=      "0000000000000010";
sprite_turn_right_down_white(12)            <=      "0000000000000010";
sprite_turn_right_down_white(13)            <=      "0000000000000010";
sprite_turn_right_down_white(14)            <=      "0000000000000010";
sprite_turn_right_down_white(15)            <=      "0000000000000010";

sprite_turn_up_right_black(0)               <=      "1111000001000001";
sprite_turn_up_right_black(1)               <=      "1001000001000000";
sprite_turn_up_right_black(2)               <=      "0011000001000000";
sprite_turn_up_right_black(3)               <=      "1110000001000000";
sprite_turn_up_right_black(4)               <=      "0110000001000000";
sprite_turn_up_right_black(5)               <=      "0100000001000000";
sprite_turn_up_right_black(6)               <=      "0100000001000000";
sprite_turn_up_right_black(7)               <=      "0110000001000000";
sprite_turn_up_right_black(8)               <=      "0110000001000000";
sprite_turn_up_right_black(9)               <=      "1111111110001100";
sprite_turn_up_right_black(10)              <=      "0100000000001111";
sprite_turn_up_right_black(11)              <=      "0110000000010000";
sprite_turn_up_right_black(12)              <=      "0100000000000000";
sprite_turn_up_right_black(13)              <=      "0100000000000000";
sprite_turn_up_right_black(14)              <=      "0110000000000000";
sprite_turn_up_right_black(15)              <=      "0110000000000000";

sprite_turn_up_right_grey(0)                <=      "0000111110000000";
sprite_turn_up_right_grey(1)                <=      "0110000000000000";
sprite_turn_up_right_grey(2)                <=      "1100111110000000";
sprite_turn_up_right_grey(3)                <=      "0001011110000000";
sprite_turn_up_right_grey(4)                <=      "0000101110000000";
sprite_turn_up_right_grey(5)                <=      "0001011110000000";
sprite_turn_up_right_grey(6)                <=      "0000000110000000";
sprite_turn_up_right_grey(7)                <=      "0000001000000000";
sprite_turn_up_right_grey(8)                <=      "0000001000000000";
sprite_turn_up_right_grey(9)                <=      "0000000000000000";
sprite_turn_up_right_grey(10)               <=      "1011111110000000";
```

```
sprite_turn_up_right_grey(11)           <=      "1001111110000000";
sprite_turn_up_right_grey(12)           <=      "1011111111100000";
sprite_turn_up_right_grey(13)           <=      "1011111111100000";
sprite_turn_up_right_grey(14)           <=      "1001111111100000";
sprite_turn_up_right_grey(15)           <=      "1001111111100000";

sprite_turn_up_right_orange(0)          <=      "0000000000111110";
sprite_turn_up_right_orange(1)          <=      "0000000000000000";
sprite_turn_up_right_orange(2)          <=      "0000000000111111";
sprite_turn_up_right_orange(3)          <=      "0000100000111111";
sprite_turn_up_right_orange(4)          <=      "1001010000111111";
sprite_turn_up_right_orange(5)          <=      "1010100000111111";
sprite_turn_up_right_orange(6)          <=      "1011111000111111";
sprite_turn_up_right_orange(7)          <=      "1001110110111111";
sprite_turn_up_right_orange(8)          <=      "1001110110111111";
sprite_turn_up_right_orange(9)          <=      "0000000001110011";
sprite_turn_up_right_orange(10)         <=      "0000000001110000";
sprite_turn_up_right_orange(11)         <=      "0000000001101111";
sprite_turn_up_right_orange(12)         <=      "0000000000000000";
sprite_turn_up_right_orange(13)         <=      "0000000000000000";
sprite_turn_up_right_orange(14)         <=      "0000000000000000";
sprite_turn_up_right_orange(15)         <=      "0000000000000000";

sprite_turn_up_right_white(0)           <=      "0000000000000000";
sprite_turn_up_right_white(1)           <=      "0000111110111111";
sprite_turn_up_right_white(2)           <=      "0000000000000000";
sprite_turn_up_right_white(3)           <=      "0000000000000000";
sprite_turn_up_right_white(4)           <=      "0000000000000000";
sprite_turn_up_right_white(5)           <=      "0000000000000000";
sprite_turn_up_right_white(6)           <=      "0000000000000000";
sprite_turn_up_right_white(7)           <=      "0000000000000000";
sprite_turn_up_right_white(8)           <=      "0000000000000000";
sprite_turn_up_right_white(9)           <=      "0000000000000000";
sprite_turn_up_right_white(10)          <=      "0000000000000000";
sprite_turn_up_right_white(11)          <=      "0000000000000000";
sprite_turn_up_right_white(12)          <=      "0000000000000000";
sprite_turn_up_right_white(13)          <=      "0000000000000000";
sprite_turn_up_right_white(14)          <=      "0000000000000000";
sprite_turn_up_right_white(15)          <=      "0000000000000000";
```

```
-- sprite tail coloring
sprite_tail_down_black(0)          <=     "0000000000000000";
sprite_tail_down_black(1)          <=     "0000000000000000";
sprite_tail_down_black(2)          <=     "0000000000000000";
sprite_tail_down_black(3)          <=     "0000000000000000";
sprite_tail_down_black(4)          <=     "0000000000000000";
sprite_tail_down_black(5)          <=     "0000000000000000";
sprite_tail_down_black(6)          <=     "0000000000000000";
sprite_tail_down_black(7)          <=     "0000000000000000";
sprite_tail_down_black(8)          <=     "1110000010011011";
sprite_tail_down_black(9)          <=     "1000000100111010";
sprite_tail_down_black(10)         <=     "1001001101000101";
sprite_tail_down_black(11)         <=     "1010001101000101";
sprite_tail_down_black(12)         <=     "1000000000000110";
sprite_tail_down_black(13)         <=     "1000000000000110";
sprite_tail_down_black(14)         <=     "1000000000000110";
sprite_tail_down_black(15)         <=     "1000000000000111";

sprite_tail_down_orange(0)         <=     "0000000000000000";
sprite_tail_down_orange(1)         <=     "0000000000000000";
sprite_tail_down_orange(2)         <=     "0000000000000000";
sprite_tail_down_orange(3)         <=     "0000000000000000";
sprite_tail_down_orange(4)         <=     "0000000000000000";
sprite_tail_down_orange(5)         <=     "0000000000000000";
sprite_tail_down_orange(6)         <=     "0000000000000000";
sprite_tail_down_orange(7)         <=     "0000000000000000";
sprite_tail_down_orange(8)         <=     "0001111101100100";
sprite_tail_down_orange(9)         <=     "0111111011000101";
sprite_tail_down_orange(10)        <=     "0110110010111010";
sprite_tail_down_orange(11)        <=     "0101110010111010";
sprite_tail_down_orange(12)        <=     "0111111111111001";
sprite_tail_down_orange(13)        <=     "0111111111111001";
sprite_tail_down_orange(14)        <=     "0111111111111001";
sprite_tail_down_orange(15)        <=     "0111111111111000";
```

```
sprite_tail_down_yellow(0)          <=      "0000000000000000";
sprite_tail_down_yellow(1)          <=      "0000000000000000";
sprite_tail_down_yellow(2)          <=      "0000000000000000";
sprite_tail_down_yellow(3)          <=      "0000000000000000";
sprite_tail_down_yellow(4)          <=      "0000000110000000";
sprite_tail_down_yellow(5)          <=      "0000011111100000";
sprite_tail_down_yellow(6)          <=      "0000111111110000";
sprite_tail_down_yellow(7)          <=      "0001111111111000";
sprite_tail_down_yellow(8)          <=      "0000000000000000";
sprite_tail_down_yellow(9)          <=      "0000000000000000";
sprite_tail_down_yellow(10)         <=      "0000000000000000";
sprite_tail_down_yellow(11)         <=      "0000000000000000";
sprite_tail_down_yellow(12)         <=      "0000000000000000";
sprite_tail_down_yellow(13)         <=      "0000000000000000";
sprite_tail_down_yellow(14)         <=      "0000000000000000";
sprite_tail_down_yellow(15)         <=      "0000000000000000";



sprite_tail_left_black(0)           <=      "1111111100000000";
sprite_tail_left_black(1)           <=      "0000000100000000";
sprite_tail_left_black(2)           <=      "0000100100000000";
sprite_tail_left_black(3)           <=      "0000010000000000";
sprite_tail_left_black(4)           <=      "0000000000000000";
sprite_tail_left_black(5)           <=      "0000000000000000";
sprite_tail_left_black(6)           <=      "0000110000000000";
sprite_tail_left_black(7)           <=      "0000111000000000";
sprite_tail_left_black(8)           <=      "0000000100000000";
sprite_tail_left_black(9)           <=      "0000110000000000";
sprite_tail_left_black(10)          <=      "0000001000000000";
sprite_tail_left_black(11)          <=      "0000001100000000";
sprite_tail_left_black(12)          <=      "0000001100000000";
sprite_tail_left_black(13)          <=      "1111110000000000";
sprite_tail_left_black(14)          <=      "1111001100000000";
sprite_tail_left_black(15)          <=      "1000110100000000";

sprite_tail_left_orange(0)          <=      "0000000000000000";
sprite_tail_left_orange(1)          <=      "1111111000000000";
sprite_tail_left_orange(2)          <=      "1111011000000000";
```

```
sprite_tail_left_orange(3)              <=      "1111101100000000";
sprite_tail_left_orange(4)              <=      "1111111100000000";
sprite_tail_left_orange(5)              <=      "1111111100000000";
sprite_tail_left_orange(6)              <=      "1111001100000000";
sprite_tail_left_orange(7)              <=      "1111000100000000";
sprite_tail_left_orange(8)              <=      "1111111000000000";
sprite_tail_left_orange(9)              <=      "1111001100000000";
sprite_tail_left_orange(10)             <=      "1111110100000000";
sprite_tail_left_orange(11)             <=      "1111110000000000";
sprite_tail_left_orange(12)             <=      "1111110000000000";
sprite_tail_left_orange(13)             <=      "0000001100000000";
sprite_tail_left_orange(14)             <=      "0000110000000000";
sprite_tail_left_orange(15)             <=      "0111001000000000";

sprite_tail_left_yellow(0)              <=      "0000000000000000";
sprite_tail_left_yellow(1)              <=      "0000000000000000";
sprite_tail_left_yellow(2)              <=      "0000000000000000";
sprite_tail_left_yellow(3)              <=      "0000000010000000";
sprite_tail_left_yellow(4)              <=      "0000000011000000";
sprite_tail_left_yellow(5)              <=      "0000000011100000";
sprite_tail_left_yellow(6)              <=      "0000000011100000";
sprite_tail_left_yellow(7)              <=      "0000000011110000";
sprite_tail_left_yellow(8)              <=      "0000000011110000";
sprite_tail_left_yellow(9)              <=      "0000000011100000";
sprite_tail_left_yellow(10)             <=      "0000000011100000";
sprite_tail_left_yellow(11)             <=      "0000000011000000";
sprite_tail_left_yellow(12)             <=      "0000000010000000";
sprite_tail_left_yellow(13)             <=      "0000000000000000";
sprite_tail_left_yellow(14)             <=      "0000000000000000";
sprite_tail_left_yellow(15)             <=      "0000000000000000";



sprite_tail_right_black(0)              <=      "0000000010110001";
sprite_tail_right_black(1)              <=      "0000000011001111";
sprite_tail_right_black(2)              <=      "0000000000111111";
sprite_tail_right_black(3)              <=      "0000000011000000";
sprite_tail_right_black(4)              <=      "0000000011000000";
sprite_tail_right_black(5)              <=      "0000000001000000";
```

```
sprite_tail_right_black(6)              <=      "0000000000110000";
sprite_tail_right_black(7)              <=      "0000000010000000";
sprite_tail_right_black(8)              <=      "0000000001110000";
sprite_tail_right_black(9)              <=      "0000000000110000";
sprite_tail_right_black(10)             <=      "0000000000000000";
sprite_tail_right_black(11)             <=      "0000000000000000";
sprite_tail_right_black(12)             <=      "0000000000100000";
sprite_tail_right_black(13)             <=      "0000000010010000";
sprite_tail_right_black(14)             <=      "0000000010000000";
sprite_tail_right_black(15)             <=      "0000000011111111";

sprite_tail_right_orange(0)             <=      "0000000001001110";
sprite_tail_right_orange(1)             <=      "0000000000110000";
sprite_tail_right_orange(2)             <=      "0000000011000000";
sprite_tail_right_orange(3)             <=      "0000000000111111";
sprite_tail_right_orange(4)             <=      "0000000000111111";
sprite_tail_right_orange(5)             <=      "0000000010111111";
sprite_tail_right_orange(6)             <=      "0000000011001111";
sprite_tail_right_orange(7)             <=      "0000000001111111";
sprite_tail_right_orange(8)             <=      "0000000010001111";
sprite_tail_right_orange(9)             <=      "0000000011001111";
sprite_tail_right_orange(10)            <=      "0000000011111111";
sprite_tail_right_orange(11)            <=      "0000000011111111";
sprite_tail_right_orange(12)            <=      "0000000011011111";
sprite_tail_right_orange(13)            <=      "0000000001101111";
sprite_tail_right_orange(14)            <=      "0000000001111111";
sprite_tail_right_orange(15)            <=      "0000000000000000";

sprite_tail_right_yellow(0)             <=      "0000000000000000";
sprite_tail_right_yellow(1)             <=      "0000000000000000";
sprite_tail_right_yellow(2)             <=      "0000000000000000";
sprite_tail_right_yellow(3)             <=      "0000000100000000";
sprite_tail_right_yellow(4)             <=      "0000001100000000";
sprite_tail_right_yellow(5)             <=      "0000011100000000";
sprite_tail_right_yellow(6)             <=      "0000011100000000";
sprite_tail_right_yellow(7)             <=      "0000111100000000";
sprite_tail_right_yellow(8)             <=      "0000111100000000";
sprite_tail_right_yellow(9)             <=      "0000011100000000";
sprite_tail_right_yellow(10)            <=      "0000011100000000";
sprite_tail_right_yellow(11)            <=      "0000001100000000";
```

```
sprite_tail_right_yellow(12)            <=      "0000000100000000";
sprite_tail_right_yellow(13)            <=      "0000000000000000";
sprite_tail_right_yellow(14)            <=      "0000000000000000";
sprite_tail_right_yellow(15)            <=      "0000000000000000";



sprite_tail_up_black(0)                         <=      "1110000000000001";
sprite_tail_up_black(1)                         <=      "0110000000000001";
sprite_tail_up_black(2)                         <=      "0110000000000001";
sprite_tail_up_black(3)                         <=      "0110000000000001";
sprite_tail_up_black(4)                         <=      "1010001011000101";
sprite_tail_up_black(5)                         <=      "1010001011001001";
sprite_tail_up_black(6)                         <=      "0101110010000001";
sprite_tail_up_black(7)                         <=      "1101100100000111";
sprite_tail_up_black(8)                         <=      "0000000000000000";
sprite_tail_up_black(9)                         <=      "0000000000000000";
sprite_tail_up_black(10)                <=      "0000000000000000";
sprite_tail_up_black(11)                <=      "0000000000000000";
sprite_tail_up_black(12)                <=      "0000000000000000";
sprite_tail_up_black(13)                <=      "0000000000000000";
sprite_tail_up_black(14)                <=      "0000000000000000";
sprite_tail_up_black(15)                <=      "0000000000000000";

sprite_tail_up_orange(0)                <=      "0001111111111110";
sprite_tail_up_orange(1)                <=      "1001111111111110";
sprite_tail_up_orange(2)                <=      "1001111111111110";
sprite_tail_up_orange(3)                <=      "1001111111111110";
sprite_tail_up_orange(4)                <=      "0101110100111010";
sprite_tail_up_orange(5)                <=      "0101110100110110";
sprite_tail_up_orange(6)                <=      "1010001101111110";
sprite_tail_up_orange(7)                <=      "0010011011111000";
sprite_tail_up_orange(8)                <=      "0000000000000000";
sprite_tail_up_orange(9)                <=      "0000000000000000";
sprite_tail_up_orange(10)               <=      "0000000000000000";
sprite_tail_up_orange(11)               <=      "0000000000000000";
sprite_tail_up_orange(12)               <=      "0000000000000000";
sprite_tail_up_orange(13)               <=      "0000000000000000";
sprite_tail_up_orange(14)               <=      "0000000000000000";
```

```
sprite_tail_up_orange(15)                   <=      "0000000000000000";

sprite_tail_up_yellow(0)                    <=      "0000000000000000";
sprite_tail_up_yellow(1)                    <=      "0000000000000000";
sprite_tail_up_yellow(2)                    <=      "0000000000000000";
sprite_tail_up_yellow(3)                    <=      "0000000000000000";
sprite_tail_up_yellow(4)                    <=      "0000000000000000";
sprite_tail_up_yellow(5)                    <=      "0000000000000000";
sprite_tail_up_yellow(6)                    <=      "0000000000000000";
sprite_tail_up_yellow(7)                    <=      "0000000000000000";
sprite_tail_up_yellow(8)                    <=      "0001111111111000";
sprite_tail_up_yellow(9)                    <=      "0000111111110000";
sprite_tail_up_yellow(10)                   <=      "0000011111100000";
sprite_tail_up_yellow(11)                   <=      "0000000110000000";
sprite_tail_up_yellow(12)                   <=      "0000000000000000";
sprite_tail_up_yellow(13)                   <=      "0000000000000000";
sprite_tail_up_yellow(14)                   <=      "0000000000000000";
sprite_tail_up_yellow(15)                   <=      "0000000000000000";


  -- sprite rabbit gray body coloring
  sprite_food_rabbit_y(0)           <=      "0000100000010000";
  sprite_food_rabbit_y(1)           <=      "0001110000111000";
  sprite_food_rabbit_y(2)           <=      "0001011001101000";
  sprite_food_rabbit_y(3)           <=      "0001001001001000";
  sprite_food_rabbit_y(4)           <=      "0001001001001000";
  sprite_food_rabbit_y(5)           <=      "0001001001001000";
  sprite_food_rabbit_y(6)           <=      "0001111111111000";
  sprite_food_rabbit_y(7)           <=      "0001100110011000";
  sprite_food_rabbit_y(8)           <=      "0001100110011000";
  sprite_food_rabbit_y(9)           <=      "0001100110011000";
  sprite_food_rabbit_y(10)    <=      "0001111111111000";
  sprite_food_rabbit_y(11)    <=      "0001111111111000";
  sprite_food_rabbit_y(12)    <=      "0001110000111000";
  sprite_food_rabbit_y(13)    <=      "0001111001111000";
  sprite_food_rabbit_y(14)    <=      "0000111111110000";
  sprite_food_rabbit_y(15)    <=      "0000011111100000";

    -- sprite rabbit blue eyeball coloring
  sprite_food_rabbit_b(0)                   <=      "0000000000000000";
```

```
sprite_food_rabbit_b(1)                 <=     "0000000000000000";
sprite_food_rabbit_b(2)                 <=     "0000000000000000";
sprite_food_rabbit_b(3)                 <=     "0000000000000000";
sprite_food_rabbit_b(4)                 <=     "0000000000000000";
sprite_food_rabbit_b(5)                 <=     "0000000000000000";
sprite_food_rabbit_b(6)                 <=     "0000000000000000";
sprite_food_rabbit_b(7)                 <=     "0000000000000000";
sprite_food_rabbit_b(8)                 <=     "0000001000100000";
sprite_food_rabbit_b(9)                 <=     "0000001000100000";
sprite_food_rabbit_b(10)          <=     "0000000000000000";
sprite_food_rabbit_b(11)          <=     "0000000000000000";
sprite_food_rabbit_b(12)      <=   "0000000000000000";
sprite_food_rabbit_b(13)      <=   "0000000000000000";
sprite_food_rabbit_b(14)      <=   "0000000000000000";
sprite_food_rabbit_b(15)      <=   "0000000000000000";

-- sprite rabbit pink nose coloring
sprite_food_rabbit_p(0)                 <=     "0000000000000000";
sprite_food_rabbit_p(1)                 <=     "0000000000000000";
sprite_food_rabbit_p(2)                 <=     "0000100000010000";
sprite_food_rabbit_p(3)                 <=     "0000110000110000";
sprite_food_rabbit_p(4)                 <=     "0000110000110000";
sprite_food_rabbit_p(5)                 <=     "0000110000110000";
sprite_food_rabbit_p(6)                 <=     "0000000000000000";
sprite_food_rabbit_p(7)                 <=     "0000000000000000";
sprite_food_rabbit_p(8)                 <=     "0000000000000000";
sprite_food_rabbit_p(9)                 <=     "0000000000000000";
sprite_food_rabbit_p(10)          <=     "0000000000000000";
sprite_food_rabbit_p(11)          <=     "0000000000000000";
sprite_food_rabbit_p(12)      <=   "0000001111000000";
sprite_food_rabbit_p(13)      <=   "0000000110000000";
sprite_food_rabbit_p(14)      <=   "0000000000000000";
sprite_food_rabbit_p(15)      <=   "0000000000000000";

 -- sprite rabbit eye white coloring
sprite_food_rabbit_w(0)                 <=     "0000000000000000";
sprite_food_rabbit_w(1)                 <=     "0000000000000000";
sprite_food_rabbit_w(2)                 <=     "0000000000000000";
sprite_food_rabbit_w(3)                 <=     "0000000000000000";
sprite_food_rabbit_w(4)                 <=     "0000000000000000";
```

```
sprite_food_rabbit_w(5)                 <=     "0000000000000000";
sprite_food_rabbit_w(6)                 <=     "0000000000000000";
sprite_food_rabbit_w(7)                 <=     "0000011001100000";
sprite_food_rabbit_w(8)                 <=     "0000010001000000";
sprite_food_rabbit_w(9)                 <=     "0000010001000000";
sprite_food_rabbit_w(10)          <=     "0000000000000000";
sprite_food_rabbit_w(11)          <=     "0000000000000000";
sprite_food_rabbit_w(12)      <=   "0000000000000000";
sprite_food_rabbit_w(13)      <=   "0000000000000000";
sprite_food_rabbit_w(14)      <=   "0000000000000000";
sprite_food_rabbit_w(15)      <=   "0000000000000000";

-- sprite mouse gray body coloring
     sprite_food_mouse_y(0)                     <=     "0001100000011000";
     sprite_food_mouse_y(1)                     <=     "0011110000111100";
     sprite_food_mouse_y(2)                     <=     "0110011001100110";
     sprite_food_mouse_y(3)                     <=     "0100001001000010";
     sprite_food_mouse_y(4)                     <=     "0100001001000010";
     sprite_food_mouse_y(5)                     <=     "0110001001000010";
     sprite_food_mouse_y(6)                     <=     "0011111111111100";
     sprite_food_mouse_y(7)                     <=     "0001100110011000";
     sprite_food_mouse_y(8)                     <=     "0001100110011000";
     sprite_food_mouse_y(9)                     <=     "0001100110011000";
     sprite_food_mouse_y(10)              <=     "0001111111111000";
     sprite_food_mouse_y(11)              <=     "0001111111111000";
     sprite_food_mouse_y(12)              <=     "0001000110001000";
     sprite_food_mouse_y(13)         <=   "0001111001111000";
     sprite_food_mouse_y(14)              <=     "0000100110010000";
     sprite_food_mouse_y(15)                     <=     "0000011111100000";

     -- sprite mouse pink coloring
     sprite_food_mouse_p(0)                     <=     "0000000000000000";
     sprite_food_mouse_p(1)                     <=     "0000000000000000";
     sprite_food_mouse_p(2)                     <=     "0001100000011000";
     sprite_food_mouse_p(3)                     <=     "0011110000111100";
     sprite_food_mouse_p(4)                     <=     "0011110000111100";
     sprite_food_mouse_p(5)                     <=     "0011110000111100";
     sprite_food_mouse_p(6)                     <=     "0000000000000000";
     sprite_food_mouse_p(7)                     <=     "0000000000000000";
     sprite_food_mouse_p(8)                     <=     "0000000000000000";
```

```
sprite_food_mouse_p(9)                  <=     "0000000000000000";
sprite_food_mouse_p(10)             <=     "0000000000000000";
sprite_food_mouse_p(11)             <=     "0000000000000000";
sprite_food_mouse_p(12)             <=     "0000000000000000";
sprite_food_mouse_p(13)         <=  "0000000000000000";
sprite_food_mouse_p(14)             <=     "0000000000000000";
sprite_food_mouse_p(15)                 <=     "0000000000000000";

-- sprite mouse whisker black coloring
sprite_food_mouse_l(0)                  <=     "0000000000000000";
sprite_food_mouse_l(1)                  <=     "0000000000000000";
sprite_food_mouse_l(2)                  <=     "0000000000000000";
sprite_food_mouse_l(3)                  <=     "0000000000000000";
sprite_food_mouse_l(4)                  <=     "0000000000000000";
sprite_food_mouse_l(5)                  <=     "0000000000000000";
sprite_food_mouse_l(6)                  <=     "0000000000000000";
sprite_food_mouse_l(7)                  <=     "0000000000000000";
sprite_food_mouse_l(8)                  <=     "0000000000000000";
sprite_food_mouse_l(9)                  <=     "0000000000000000";
sprite_food_mouse_l(10)             <=     "0000000000000000";
sprite_food_mouse_l(11)             <=     "0000000000000000";
sprite_food_mouse_l(12)             <=     "0000111001110000";
sprite_food_mouse_l(13)         <=  "0000000110000000";
sprite_food_mouse_l(14)             <=     "0000011001100000";
sprite_food_mouse_l(15)                 <=     "0000000000000000";

-- sprite food edwards hair coloring
sprite_food_edwards_n(0)                <=     "0000000000000000";
sprite_food_edwards_n(1)                <=     "0000011111100000";
sprite_food_edwards_n(2)                <=     "0000111111110000";
sprite_food_edwards_n(3)                <=     "0001111111111000";
sprite_food_edwards_n(4)                <=     "0001111111111000";
sprite_food_edwards_n(5)                <=     "0001000000001000";
sprite_food_edwards_n(6)                <=     "0000000000000000";
sprite_food_edwards_n(7)                <=     "0000000000000000";
sprite_food_edwards_n(8)                <=     "0000000000000000";
sprite_food_edwards_n(9)                <=     "0000000000000000";
sprite_food_edwards_n(10)               <=     "0000000000000000";
sprite_food_edwards_n(11)               <=     "0000000000000000";
sprite_food_edwards_n(12)               <=     "0000000000000000";
```

```
sprite_food_edwards_n(13)                <=     "0000000000000000";
sprite_food_edwards_n(14)                <=     "0000000000000000";
sprite_food_edwards_n(15)                <=     "0000000000000000";

-- sprite food edwards skin coloring
sprite_food_edwards_t(0)                 <=     "0000000000000000";
sprite_food_edwards_t(1)                 <=     "0000000000000000";
sprite_food_edwards_t(2)                 <=     "0000000000000000";
sprite_food_edwards_t(3)                 <=     "0000000000000000";
sprite_food_edwards_t(4)                 <=     "0000000000000000";
sprite_food_edwards_t(5)                 <=     "0000111111110000";
sprite_food_edwards_t(6)                 <=     "0001100110011000";
sprite_food_edwards_t(7)                 <=     "0010000000000100";
sprite_food_edwards_t(8)                 <=     "0011000000001100";
sprite_food_edwards_t(9)                 <=     "0011000000001100";
sprite_food_edwards_t(10)                <=     "0001100110011000";
sprite_food_edwards_t(11)                <=     "0001111111111000";
sprite_food_edwards_t(12)                <=     "0001110001111000";
sprite_food_edwards_t(13)          <=    "0001110001111000";
sprite_food_edwards_t(14)                <=     "0000111111110000";
sprite_food_edwards_t(15)                <=     "0000011111100000";

-- sprite food edwards lips coloring
sprite_food_edwards_p(0)                 <=     "0000000000000000";
sprite_food_edwards_p(1)                 <=     "0000000000000000";
sprite_food_edwards_p(2)                 <=     "0000000000000000";
sprite_food_edwards_p(3)                 <=     "0000000000000000";
sprite_food_edwards_p(4)                 <=     "0000000000000000";
sprite_food_edwards_p(5)                 <=     "0000000000000000";
sprite_food_edwards_p(6)                 <=     "0000000000000000";
sprite_food_edwards_p(7)                 <=     "0000000000000000";
sprite_food_edwards_p(8)                 <=     "0000000000000000";
sprite_food_edwards_p(9)                 <=     "0000000000000000";
sprite_food_edwards_p(10)                <=     "0000000000000000";
sprite_food_edwards_p(11)                <=     "0000000000000000";
sprite_food_edwards_p(12)                <=     "0000001110000000";
sprite_food_edwards_p(13)          <=    "0000001110000000";
sprite_food_edwards_p(14)                <=     "0000000000000000";
sprite_food_edwards_p(15)                <=     "0000000000000000";
```

```
-- sprite food edwards glasses black coloring
sprite_food_edwards_l(0)                    <=      "0000000000000000";
sprite_food_edwards_l(1)                    <=      "0000000000000000";
sprite_food_edwards_l(2)                    <=      "0000000000000000";
sprite_food_edwards_l(3)                    <=      "0000000000000000";
sprite_food_edwards_l(4)                    <=      "0000000000000000";
sprite_food_edwards_l(5)                    <=      "0000000000000000";
sprite_food_edwards_l(6)                    <=      "0010011001100100";
sprite_food_edwards_l(7)                    <=      "0001100110011000";
sprite_food_edwards_l(8)                    <=      "0000100110010000";
sprite_food_edwards_l(9)                    <=      "0000100110010000";
sprite_food_edwards_l(10)                   <=      "0000011001100000";
sprite_food_edwards_l(11)                   <=      "0000000000000000";
sprite_food_edwards_l(12)                   <=      "0000000000000000";
sprite_food_edwards_l(13)           <=      "0000000000000000";
sprite_food_edwards_l(14)                   <=      "0000000000000000";
sprite_food_edwards_l(15)                   <=      "0000000000000000";


-- sprite needle growth blood coloring
sprite_powup_growth_r(0)                    <=      "0000000000000000";
sprite_powup_growth_r(1)                    <=      "0000000000000000";
sprite_powup_growth_r(2)                    <=      "0000000000000000";
sprite_powup_growth_r(3)                    <=      "0000000000000000";
sprite_powup_growth_r(4)                    <=      "0000000000000000";
sprite_powup_growth_r(5)                    <=      "0000011000000000";
sprite_powup_growth_r(6)                    <=      "0000011100000000";
sprite_powup_growth_r(7)                    <=      "0000001110000000";
sprite_powup_growth_r(8)                    <=      "0000000111000000";
sprite_powup_growth_r(9)                    <=      "0000000011100000";
sprite_powup_growth_r(10)                   <=      "0000000001100000";
sprite_powup_growth_r(11)                   <=      "0000000000000000";
sprite_powup_growth_r(12)                   <=      "0000000000000000";
sprite_powup_growth_r(13)           <=      "0000000000000000";
sprite_powup_growth_r(14)                   <=      "0000000000000000";
sprite_powup_growth_r(15)                   <=      "0000000000000000";

-- sprite needle growth gray coloring
sprite_powup_growth_y(0)                    <=      "1000000000000000";
```

```
sprite_powup_growth_y(1)                <=      "0100000000000000";
sprite_powup_growth_y(2)                <=      "0010000000000000";
sprite_powup_growth_y(3)                <=      "0001000000000000";
sprite_powup_growth_y(4)                <=      "0000111000000000";
sprite_powup_growth_y(5)                <=      "0000100100000000";
sprite_powup_growth_y(6)                <=      "0000100010000000";
sprite_powup_growth_y(7)                <=      "0000010001000000";
sprite_powup_growth_y(8)                <=      "0000001000100000";
sprite_powup_growth_y(9)                <=      "0000000100010100";
sprite_powup_growth_y(10)               <=      "0000000010011000";
sprite_powup_growth_y(11)               <=      "0000000001110000";
sprite_powup_growth_y(12)               <=      "0000000000101000";
sprite_powup_growth_y(13)           <=      "0000000001000111";
sprite_powup_growth_y(14)               <=      "0000000000000110";
sprite_powup_growth_y(15)               <=      "0000000000000100";

-- sprite lightning coloring
sprite_powup_speed(0)                   <=      "0000000011111110";
sprite_powup_speed(1)                   <=      "0000000111111100";
sprite_powup_speed(2)                   <=      "0000001111111000";
sprite_powup_speed(3)                   <=      "0000011111110000";
sprite_powup_speed(4)                   <=      "0000111111100000";
sprite_powup_speed(5)                   <=      "0001111111000000";
sprite_powup_speed(6)                   <=      "0011111111111100";
sprite_powup_speed(7)                   <=      "0111111111111000";
sprite_powup_speed(8)                   <=      "0000001111110000";
sprite_powup_speed(9)                   <=      "0000001111100000";
sprite_powup_speed(10)                  <=      "0000011111000000";
sprite_powup_speed(11)                  <=      "0000011110000000";
sprite_powup_speed(12)                  <=      "0000111100000000";
sprite_powup_speed(13)              <=      "0000111000000000";
sprite_powup_speed(14)                  <=      "0001110000000000";
sprite_powup_speed(15)                  <=      "0001100000000000";

-- sprite ice freeze coloring
sprite_powup_freeze(0)                  <=      "1100010000010011";
sprite_powup_freeze(1)                  <=      "0110110000010110";
sprite_powup_freeze(2)                  <=      "0011100101011100";
sprite_powup_freeze(3)                  <=      "0111100010011110";
sprite_powup_freeze(4)                  <=      "1100110010110011";
```

```
sprite_powup_freeze(5)                  <=     "0000011011100000";
sprite_powup_freeze(6)                  <=     "0010001111000100";
sprite_powup_freeze(7)                  <=     "0001000110001000";
sprite_powup_freeze(8)                  <=     "0111111111111110";
sprite_powup_freeze(9)                  <=     "0001001111001000";
sprite_powup_freeze(10)             <=     "0010011011100100";
sprite_powup_freeze(11)             <=     "0000110010110000";
sprite_powup_freeze(12)             <=     "1111100101011111";
sprite_powup_freeze(13)         <=  "0011100000011100";
sprite_powup_freeze(14)             <=     "0110110000110110";
sprite_powup_freeze(15)                 <=     "1100010000100011";


        -- sprite brick wall coloring
sprite_wall(0)                      <=     "1111101111101111";
sprite_wall(1)                      <=     "1111101111101111";
sprite_wall(2)                      <=     "1111101111101111";
sprite_wall(3)                      <=     "1111101111101111";
sprite_wall(4)                      <=     "0000000000000000";
sprite_wall(5)                      <=     "1011111011111011";
sprite_wall(6)                      <=     "1011111011111011";
sprite_wall(7)                      <=     "1011111011111011";
sprite_wall(8)                      <=     "0000000000000000";
sprite_wall(9)                      <=     "1111011111011111";
sprite_wall(10)                     <=     "1111011111011111";
sprite_wall(11)                     <=     "1111011111011111";
sprite_wall(12)                     <=     "0000000000000000";
sprite_wall(13)                 <=     "1111101111101111";
sprite_wall(14)                     <=     "1111101111101111";
sprite_wall(15)                     <=     "1111101111101111";

-- sprite letter p
sprite_P(0)                     <=     "0000000000000000";
sprite_P(1)                     <=     "0000011111100000";
sprite_P(2)                     <=     "0000111111110000";
sprite_P(3)                     <=     "0000110001110000";
sprite_P(4)                     <=     "0000110000110000";
sprite_P(5)                     <=     "0000110000110000";
sprite_P(6)                     <=     "0000110001110000";
sprite_P(7)                     <=     "0000110011110000";
```

```
sprite_P(8)                 <=     "0000111111100000";
sprite_P(9)                 <=     "0000111111000000";
sprite_P(10)                <=     "0000110000000000";
sprite_P(11)                <=     "0000110000000000";
sprite_P(12)                <=     "0000110000000000";
sprite_P(13)          <=     "0000110000000000";
sprite_P(14)                <=     "0000110000000000";
sprite_P(15)                <=     "0000000000000000";

-- sprite number 1
sprite_1(0)                 <=     "0000000000000000";
sprite_1(1)                 <=     "0000001111000000";
sprite_1(2)                 <=     "0000011111000000";
sprite_1(3)                 <=     "0000111111000000";
sprite_1(4)                 <=     "0000110111000000";
sprite_1(5)                 <=     "0000000111000000";
sprite_1(6)                 <=     "0000000111000000";
sprite_1(7)                 <=     "0000000111000000";
sprite_1(8)                 <=     "0000000111000000";
sprite_1(9)                 <=     "0000000111000000";
sprite_1(10)                <=     "0000000111000000";
sprite_1(11)                <=     "0000000111000000";
sprite_1(12)                <=     "0000011111110000";
sprite_1(13)          <=     "0000011111110000";
sprite_1(14)                <=     "0000011111110000";
sprite_1(15)                <=     "0000000000000000";

-- sprite number 2
sprite_2(0)                 <=     "0000000000000000";
sprite_2(1)                 <=     "0000001111100000";
sprite_2(2)                 <=     "0000011111110000";
sprite_2(3)                 <=     "0000111000110000";
sprite_2(4)                 <=     "0000110000110000";
sprite_2(5)                 <=     "0000000001110000";
sprite_2(6)                 <=     "0000000001110000";
sprite_2(7)                 <=     "0000000011100000";
sprite_2(8)                 <=     "0000000111000000";
sprite_2(9)                 <=     "0000001110000000";
sprite_2(10)                <=     "0000001110000000";
sprite_2(11)                <=     "0000011100000000";
```

```
sprite_2(12)                <=     "0000111000000000";
sprite_2(13)          <=     "0000111111110000";
sprite_2(14)                <=     "0000111111110000";
sprite_2(15)                <=     "0000000000000000";

-- sprite letter W
sprite_W(0)                 <=     "0000000000000000";
sprite_W(1)                 <=     "0010000000001000";
sprite_W(2)                 <=     "0110000000001100";
sprite_W(3)                 <=     "0110000100001100";
sprite_W(4)                 <=     "0110001110001100";
sprite_W(5)                 <=     "0110001110001100";
sprite_W(6)                 <=     "0110001110001100";
sprite_W(7)                 <=     "0110011111001100";
sprite_W(8)                 <=     "0110011011001100";
sprite_W(9)                 <=     "0110011011001100";
sprite_W(10)                <=     "0111011011011100";
sprite_W(11)                <=     "0111111011111100";
sprite_W(12)                <=     "0111111011111100";
sprite_W(13)          <=     "0011110001111000";
sprite_W(14)                <=     "0001100000110000";
sprite_W(15)                <=     "0000000000000000";

-- sprite letter I
sprite_I(0)                 <=     "0000000000000000";
sprite_I(1)                 <=     "0000111111110000";
sprite_I(2)                 <=     "0000111111110000";
sprite_I(3)                 <=     "0000000110000000";
sprite_I(4)                 <=     "0000000110000000";
sprite_I(5)                 <=     "0000000110000000";
sprite_I(6)                 <=     "0000000110000000";
sprite_I(7)                 <=     "0000000110000000";
sprite_I(8)                 <=     "0000000110000000";
sprite_I(9)                 <=     "0000000110000000";
sprite_I(10)                <=     "0000000110000000";
sprite_I(11)                <=     "0000000110000000";
sprite_I(12)                <=     "0000000110000000";
sprite_I(13)          <=     "0000111111110000";
sprite_I(14)                <=     "0000111111110000";
sprite_I(15)                <=     "0000000000000000";
```

```
-- sprite letter N
sprite_N(0)                <=     "0000000000000000";
sprite_N(1)                <=     "0001100000011000";
sprite_N(2)                <=     "0001110000011000";
sprite_N(3)                <=     "0001110000011000";
sprite_N(4)                <=     "0001111000011000";
sprite_N(5)                <=     "0001111100011000";
sprite_N(6)                <=     "0001101110011000";
sprite_N(7)                <=     "0001101110011000";
sprite_N(8)                <=     "0001100111011000";
sprite_N(9)                <=     "0001100011111000";
sprite_N(10)               <=     "0001100011111000";
sprite_N(11)               <=     "0001100001111000";
sprite_N(12)               <=     "0001100000111000";
sprite_N(13)          <=     "0001100000011000";
sprite_N(14)               <=     "0000000000000000";
sprite_N(15)               <=     "0000000000000000";

-- sprite letter S
sprite_S(0)                <=     "0000011111000000";
sprite_S(1)                <=     "0000111111110000";
sprite_S(2)                <=     "0001110011111000";
sprite_S(3)                <=     "0011100000111000";
sprite_S(4)                <=     "0011100000000000";
sprite_S(5)                <=     "0001110000000000";
sprite_S(6)                <=     "0000111000000000";
sprite_S(7)                <=     "0000111110000000";
sprite_S(8)                <=     "0000001111100000";
sprite_S(9)                <=     "0000000001110000";
sprite_S(10)               <=     "0000000000111000";
sprite_S(11)               <=     "0011100000111000";
sprite_S(12)               <=     "0011100001110000";
sprite_S(13)          <=     "0001110011100000";
sprite_S(14)               <=     "0000111111000000";
sprite_S(15)               <=     "0000011110000000";

-- sprite letter T
sprite_T(0)                <=     "0000000000000000";
sprite_T(1)                <=     "0011111111111100";
```

```
sprite_T(2)                 <=     "0011111111111100";
sprite_T(3)                 <=     "0000000110000000";
sprite_T(4)                 <=     "0000000110000000";
sprite_T(5)                 <=     "0000000110000000";
sprite_T(6)                 <=     "0000000110000000";
sprite_T(7)                 <=     "0000000110000000";
sprite_T(8)                 <=     "0000000110000000";
sprite_T(9)                 <=     "0000000110000000";
sprite_T(10)                <=     "0000000110000000";
sprite_T(11)                <=     "0000000110000000";
sprite_T(12)                <=     "0000000110000000";
sprite_T(13)          <=     "0000000110000000";
sprite_T(14)                <=     "0000000110000000";
sprite_T(15)                <=     "0000000000000000";

-- sprite letter E
sprite_E(0)                 <=     "0000000000000000";
sprite_E(1)                 <=     "0011111111111100";
sprite_E(2)                 <=     "0011111111111100";
sprite_E(3)                 <=     "0011000000000000";
sprite_E(4)                 <=     "0011000000000000";
sprite_E(5)                 <=     "0011000000000000";
sprite_E(6)                 <=     "0011000000000000";
sprite_E(7)                 <=     "0011111111111100";
sprite_E(8)                 <=     "0011111111111100";
sprite_E(9)                 <=     "0011000000000000";
sprite_E(10)                <=     "0011000000000000";
sprite_E(11)                <=     "0011000000000000";
sprite_E(12)                <=     "0011000000000000";
sprite_E(13)          <=     "0011111111111100";
sprite_E(14)                <=     "0011111111111100";
sprite_E(15)                <=     "0000000000000000";

-- sprite exclamation point
sprite_exclam(0)                    <=     "0000000110000000";
sprite_exclam(1)                    <=     "0000001111000000";
sprite_exclam(2)                    <=     "0000001111000000";
sprite_exclam(3)                    <=     "0000001111000000";
sprite_exclam(4)                    <=     "0000001111000000";
sprite_exclam(5)                    <=     "0000001111000000";
```

```
sprite_exclam(6)                    <=      "0000001111000000";
sprite_exclam(7)                    <=      "0000001111000000";
sprite_exclam(8)                    <=      "0000001111000000";
sprite_exclam(9)                    <=      "0000000110000000";
sprite_exclam(10)             <=      "0000000110000000";
sprite_exclam(11)             <=      "0000000000000000";
sprite_exclam(12)             <=      "0000000110000000";
sprite_exclam(13)         <=  "0000001111000000";
sprite_exclam(14)             <=      "0000001111000000";
sprite_exclam(15)                   <=      "0000000110000000";

-- sprite pause button
sprite_pause(0)                     <=      "0000000000000000";
sprite_pause(1)                     <=      "0001110000111000";
sprite_pause(2)                     <=      "0001110000111000";
sprite_pause(3)                     <=      "0001110000111000";
sprite_pause(4)                     <=      "0001110000111000";
sprite_pause(5)                     <=      "0001110000111000";
sprite_pause(6)                     <=      "0001110000111000";
sprite_pause(7)                     <=      "0001110000111000";
sprite_pause(8)                     <=      "0001110000111000";
sprite_pause(9)                     <=      "0001110000111000";
sprite_pause(10)                    <=      "0001110000111000";
sprite_pause(11)                    <=      "0001110000111000";
sprite_pause(12)                    <=      "0001110000111000";
sprite_pause(13)              <=      "0001110000111000";
sprite_pause(14)                    <=      "0001110000111000";
sprite_pause(15)                    <=      "0000000000000000";

-- sprite play button
sprite_play(0)                      <=      "0001100000000000";
sprite_play(1)                      <=      "0001110000000000";
sprite_play(2)                      <=      "0001111000000000";
sprite_play(3)                      <=      "0001111100000000";
sprite_play(4)                      <=      "0001111110000000";
sprite_play(5)                      <=      "0001111111000000";
sprite_play(6)                      <=      "0001111111100000";
sprite_play(7)                      <=      "0001111111110000";
sprite_play(8)                      <=      "0001111111110000";
sprite_play(9)                      <=      "0001111111100000";
```

```
        sprite_play(10)                     <=      "0001111111000000";
        sprite_play(11)                     <=      "0001111110000000";
        sprite_play(12)                     <=      "0001111100000000";
        sprite_play(13)             <=      "0001111000000000";
        sprite_play(14)                     <=      "0001110000000000";
        sprite_play(15)                     <=      "0001100000000000";




  end rtl;


*********************************************************************************************
de2_wm8731_audio.vhd

library ieee;
use ieee.std_logic_1164.all;
use ieee.numeric_std.all;

--use work.sounds.all;

entity de2_wm8731_audio is
        port (
                clk                     : in std_logic;
                --              Audio CODEC Chip Clock AUD_XCK
                reset_n         : in std_logic;

                start_sound     : in std_logic;
        --              Start sound playback
                select_sound: in std_logic_vector(3 downto 0);              --              Select
a sound

                change_divider_enable : in std_logic;                                   --
        Enable divider change
                divider_in              : in std_logic_vector(31 downto 0);         --
        Value to change divider to
```

```
                --      Audio interface signals
                AUD_ADCLRCK     : out std_logic;
        --      Audio CODEC ADC LR Clock
                AUD_ADCDAT      : in  std_logic;
        --      Audio CODEC ADC Data
                AUD_DACLRCK     : out std_logic;
        --      Audio CODEC DAC LR Clock
                AUD_DACDAT      : out std_logic;
        --      Audio CODEC DAC Data
                AUD_BCLK                : inout std_logic;
        --      Audio CODEC Bit-Stream Clock

                leds                    : out std_logic_vector(15 downto 0)
        );
end  de2_wm8731_audio;

architecture rtl of de2_wm8731_audio is

COMPONENT move_sound_rom IS
        PORT
        (
                address         : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
                clock                   : IN STD_LOGIC  := '1';
                q                               : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
        );
END COMPONENT;

COMPONENT powerup_sound_rom IS
        PORT
        (
                address         : IN STD_LOGIC_VECTOR (8 DOWNTO 0);
                clock                   : IN STD_LOGIC  := '1';
                q                               : OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
        );
END COMPONENT;

COMPONENT gameover_sound_rom IS
        PORT
        (
```

```
		address		: IN STD_LOGIC_VECTOR (11 DOWNTO 0);
		clock		: IN STD_LOGIC  := '1';
		q			: OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
	);
END COMPONENT;

COMPONENT snakePlus_sound_rom IS
	PORT
	(
		address		: IN STD_LOGIC_VECTOR (11 DOWNTO 0);
		clock		: IN STD_LOGIC  := '1';
		q			: OUT STD_LOGIC_VECTOR (15 DOWNTO 0)
	);
END COMPONENT;

signal lrck : std_logic;
signal bclk : std_logic;
signal xck  : std_logic;

signal lrck_divider : unsigned(11 downto 0);
signal bclk_divider : unsigned(3 downto 0);

signal set_bclk		: std_logic;
signal set_lrck		: std_logic;
signal clr_bclk		: std_logic;
signal lrck_lat		: std_logic;


signal sound_on		: std_logic := '0';
signal sound		: std_logic_vector(3 downto 0) := X"0";
signal sound_end0 : std_logic := '0';
signal sound_end1 : std_logic := '0';
signal sound_end2 : std_logic := '0';
signal sound_end3 : std_logic := '0';


signal sound0_out	: std_logic_vector(15 downto 0);
signal sound1_out	: std_logic_vector(15 downto 0);
signal sound2_out	: std_logic_vector(15 downto 0);
```

```
signal sound3_out     : std_logic_vector(15 downto 0);

signal counter0       : unsigned(8 downto 0);
signal counter1       : unsigned(8 downto 0);
signal counter2       : unsigned(11 downto 0);
signal counter3       : unsigned(11 downto 0);
signal reset_counters : std_logic := '0';
signal unset          : std_logic := '0';

signal shift_out      : std_logic_vector(15 downto 0) := (others => '0');

--constant divider : integer := 561;
--signal divider : unsigned(31 downto 0) := X"00000823";
signal divider : unsigned(31 downto 0);



begin

        -- LRCK divider
        -- Audio chip main clock is 25MHz / Sample rate 12KHz
        -- Divider is 25 MHz / 8KHz = 3125s (X"C34")
        -- Left justify mode set by I2C controller


        process(clk)
        begin
                if rising_edge(clk) then
                        if reset_n = '0' then
                                divider <= X"00000C34";
                                leds(5) <= '0';
                        elsif change_divider_enable = '1' then
                                divider <= unsigned(divider_in);
                                leds(5) <= '1';
                        else
                                leds(5) <= '0';
                        end if;
                end if;
        end process;
```

```
process (clk)
begin
	if rising_edge(clk) then
		if reset_n = '0' then
			lrck_divider <= (others => '0');
		elsif lrck_divider = divider  then
			lrck_divider <= X"000";
		else
			lrck_divider <= lrck_divider + 1;
		end if;
	end if;
end process;

process (clk)
begin
	if rising_edge(clk) then
		if reset_n = '0' then
			bclk_divider <= (others => '0');
		elsif bclk_divider = X"B" or set_lrck = '1'  then
			bclk_divider <= X"0";
		else
			bclk_divider <= bclk_divider + 1;
		end if;
	end if;
end process;

set_lrck <= '1' when lrck_divider = divider else '0';

process (clk)
begin
	if rising_edge(clk) then
		if reset_n = '0' then
			lrck <= '0';
		elsif set_lrck = '1' then
			lrck <= not lrck;
		end if;
	end if;
end process;

-- BCLK divider
```

```
set_bclk <= '1' when bclk_divider(3 downto 0) = "0101" else '0';
clr_bclk <= '1' when bclk_divider(3 downto 0) = "1011" else '0';

process (clk)
begin
	if rising_edge(clk) then
		if reset_n = '0' then
			bclk <= '0';
		elsif set_lrck = '1' or clr_bclk = '1' then
			bclk <= '0';
		elsif set_bclk = '1' then
			bclk <= '1';
		end if;
	end if;
end process;

-- Audio data shift output
process (clk)
begin
	if rising_edge(clk) then
		if reset_n = '0' then
			shift_out <= (others => '0');
		elsif set_lrck = '1' then
			if sound_on = '1' then
				if sound = X"0" then
					shift_out <= sound0_out;
				elsif sound = X"1" then
					shift_out <= sound1_out;
				elsif sound = X"2" then
					shift_out <= sound2_out;
				elsif sound = X"3" then
					shift_out <= sound3_out;
				end if;
			end if;
		elsif clr_bclk = '1' then
			shift_out <= shift_out (14 downto 0) & '0';
		end if;
	end if;
end process;
```

```
-- Audio outputs

AUD_ADCLRCK  <= lrck;
AUD_DACLRCK  <= lrck;
AUD_DACDAT   <= shift_out(15);
AUD_BCLK     <= bclk;

process (clk)
begin
	if rising_edge(clk) then
		if reset_n = '0' then
			sound_on <= '0';
			leds(15) <= '0';
			leds(14) <= '0';
			sound <= X"0";
			reset_counters <= '0';
			unset <= '0';
		elsif start_sound = '1' then
			sound_on <= '1';
			leds(15 downto 14) <= "10";
			sound <= select_sound;
			reset_counters <= '1';
			unset <= '1';
		elsif unset = '1' then
			reset_counters <= '0';
			unset <= '0';
		elsif (sound_end0 or sound_end1 or sound_end2 or sound_end3) = '1' then
			sound_on <= '0';
			leds(15 downto 14) <= "01";
		end if;
	end if;
end process;

leds(9) <= lrck_lat and not lrck;

-- Counter for sound0
process(clk)
begin
	if rising_edge(clk) then
		if reset_n = '0' then
```

```
                counter0 <= (others => '0');
                sound_end0 <= '0';
                leds(12 downto 10) <= "000";
        elsif reset_counters = '1' then
                counter0 <= (others => '0');
        elsif lrck_lat = '1' and lrck = '0'  then
                if sound_on = '1' and sound = X"0" then
                        if (counter0 < X"1F9") then
                                leds(11 downto 10) <= "10";
                                counter0 <= counter0 + 1;
                        else
                                leds(11 downto 10) <= "01";
                                counter0 <= (others => '0');
                                sound_end0 <= '1';
                        end if;
                end if;
        elsif sound_end0 = '1' then
                leds(12) <= '0';
                sound_end0 <= '0';
        end if;
    end if;
end process;

-- Counter for sound1
process(clk)
begin
    if rising_edge(clk) then
        if reset_n = '0' then
                counter1 <= (others => '0');
                sound_end1 <= '0';
        elsif reset_counters = '1' then
                counter1 <= (others => '0');
        elsif lrck_lat = '1' and lrck = '0'  then
                if sound_on = '1' and sound = X"1" then
                        if (counter1 < X"199") then
                                counter1 <= counter1 + 1;
                        else
                                counter1 <= (others => '0');
                                sound_end1 <= '1';
                        end if;
                end if;
```

```
                elsif sound_end1 = '1' then
                        sound_end1 <= '0';
                end if;
        end if;
end process;

-- Counter for sound2
process(clk)
begin
        if rising_edge(clk) then
                if reset_n = '0' then
                        counter2 <= (others => '0');
                        sound_end2 <= '0';
                elsif reset_counters = '1' then
                        counter2 <= (others => '0');
                elsif lrck_lat = '1' and lrck = '0'  then
                        if sound_on = '1' and sound = X"2" then
                                if (counter2 < X"82A") then
                                        counter2 <= counter2 + 1;
                                else
                                        counter2 <= (others => '0');
                                        sound_end2 <= '1';
                                end if;
                        end if;
                elsif sound_end2 = '1' then
                        sound_end2 <= '0';
                end if;
        end if;
end process;

-- Counter for sound3
process(clk)
begin
        if rising_edge(clk) then
                if reset_n = '0' then
                        counter3 <= (others => '0');
                        sound_end3 <= '0';
                elsif reset_counters = '1' then
                        counter3 <= (others => '0');
                elsif lrck_lat = '1' and lrck = '0'  then
```

```
                        if sound_on = '1' and sound = X"3" then
                                if (counter3 < X"961") then
                                        counter3 <= counter3 + 1;
                                else
                                        counter3 <= (others => '0');
                                        sound_end3 <= '1';
                                end if;
                        end if;
                elsif sound_end3 = '1' then
                        sound_end3 <= '0';
                end if;
        end if;
end process;



process(clk)
begin
        if rising_edge(clk) then
                lrck_lat <= lrck;
        end if;
end process;

s0: move_sound_rom port map (
        address         => std_logic_vector(counter0),
        clock                   => clk,
        q                               => sound0_out
);

s1: powerup_sound_rom port map (
        address         => std_logic_vector(counter1),
        clock                   => clk,
        q                               => sound1_out
);

s2: gameover_sound_rom port map (
        address         => std_logic_vector(counter2),
        clock                   => clk,
        q                               => sound2_out
);
```

```
	s3: snakePlus_sound_rom port map (
		address		=> std_logic_vector(counter3),
		clock			=> clk,
		q				=> sound3_out
	);
end architecture;


--------------------------------------------------------------------------------
definitions.vhd

library ieee;
use ieee.std_logic_1164.all;

package DEFINITIONS is

	--Type Definitions
	  type tiles_ram is array(39 downto 0, 29 downto 0) of
						std_logic_vector(7 downto 0);

	   type snake_ram is array(1200 downto 0) of
						std_logic_vector(31 downto 0);

	--Snake constants
		--constant MAX_SNAKE_SIZE    : integer := 10;

	--Sprite Constants
		constant SPRITE_LEN: integer := 15;



	-- Color declarations
		constant BLACK : STD_LOGIC_VECTOR    := "000";
		constant WHITE : STD_LOGIC_VECTOR    := "001";
		constant RED   : STD_LOGIC_VECTOR    := "010";
		constant GREEN : STD_LOGIC_VECTOR    := "011";
		constant BLUE  : STD_LOGIC_VECTOR    := "100";
		constant YELLOW: STD_LOGIC_VECTOR    := "101";
		constant GRAY  : STD_LOGIC_VECTOR    := "110";
		constant PINK  : STD_LOGIC_VECTOR    := "111";
```

```
-- Sprite Select Codes
constant UNUSED_CODE         : STD_LOGIC_VECTOR(4 downto 0)      := "00000";
constant SNAKE_HEAD_RIGHT    : STD_LOGIC_VECTOR(4 downto 0)      := "00001";
constant SNAKE_HEAD_LEFT     : STD_LOGIC_VECTOR(4 downto 0)      := "00010";
constant SNAKE_HEAD_UP       : STD_LOGIC_VECTOR(4 downto 0)      := "00011";
constant SNAKE_HEAD_DOWN     : STD_LOGIC_VECTOR(4 downto 0)      := "00100";
constant SNAKE_TAIL_RIGHT    : STD_LOGIC_VECTOR(4 downto 0)      := "00101";
constant SNAKE_TAIL_LEFT     : STD_LOGIC_VECTOR(4 downto 0)      := "00110";
constant SNAKE_TAIL_UP       : STD_LOGIC_VECTOR(4 downto 0)      := "00111";
constant SNAKE_TAIL_DOWN     : STD_LOGIC_VECTOR(4 downto 0)      := "01000";
constant SNAKE_BODY_RIGHT    : STD_LOGIC_VECTOR(4 downto 0)      := "01001";
constant SNAKE_BODY_LEFT     : STD_LOGIC_VECTOR(4 downto 0)      := "01010";
constant SNAKE_BODY_UP       : STD_LOGIC_VECTOR(4 downto 0)      := "01011";
constant SNAKE_BODY_DOWN     : STD_LOGIC_VECTOR(4 downto 0)      := "01100";

constant SNAKE_TURN_UP_RIGHT: STD_LOGIC_VECTOR(4 downto 0)       := "01101";
constant SNAKE_TURN_RIGHT_DOWN: STD_LOGIC_VECTOR(4 downto 0)     := "01110";
constant SNAKE_TURN_DOWN_LEFT: STD_LOGIC_VECTOR(4 downto 0)      := "01111";
constant SNAKE_TURN_LEFT_UP  : STD_LOGIC_VECTOR(4 downto 0)      := "10000";


-- Reuse codes for other sprites
constant RABBIT_CODE  : STD_LOGIC_VECTOR(4 downto 0)      := "00001";
constant MOUSE_CODE          : STD_LOGIC_VECTOR(4 downto 0)      := "00010";
constant EDWARDS_CODE        : STD_LOGIC_VECTOR(4 downto 0)      := "00011";
constant WALL_CODE           : STD_LOGIC_VECTOR(4 downto 0)      := "00100";
constant SPEED_CODE          : STD_LOGIC_VECTOR(4 downto 0)      := "00101";
constant FREEZE_CODE  : STD_LOGIC_VECTOR(4 downto 0)      := "00110";
constant GROWTH_CODE  : STD_LOGIC_VECTOR(4 downto 0)      := "00111";
constant ONE_CODE            : STD_LOGIC_VECTOR(4 downto 0)      := "01000";
constant TWO_CODE            : STD_LOGIC_VECTOR(4 downto 0)      := "01001";
constant P_CODE              : STD_LOGIC_VECTOR(4 downto 0)      := "01010";
constant W_CODE              : STD_LOGIC_VECTOR(4 downto 0)      := "01011";
constant I_CODE              : STD_LOGIC_VECTOR(4 downto 0)      := "01100";
constant N_CODE              : STD_LOGIC_VECTOR(4 downto 0)      := "01101";
constant S_CODE              : STD_LOGIC_VECTOR(4 downto 0)      := "01110";
constant EXC_CODE            : STD_LOGIC_VECTOR(4 downto 0)      := "01111";
constant PLAY_CODE           : STD_LOGIC_VECTOR(4 downto 0)      := "10000";
```

```
            constant PAUSE_CODE          : STD_LOGIC_VECTOR(4 downto 0)      := "10001";
            constant SPLASH_SNAKE_CODE : STD_LOGIC_VECTOR(4 downto 0) := "10010";
            constant T_CODE              : STD_LOGIC_VECTOR(4 downto 0)      := "10011";
            constant E_CODE              : STD_LOGIC_VECTOR(4 downto 0)      := "10100";

end package DEFINITIONS;


**********************************************************************************************
nes_fsm.vhd

library ieee;
use ieee.std_logic_1164.all;

entity nes_fsm is
    port (
            clk                  : in std_logic;
            reset                : in std_logic;
            latch1        : out std_logic;
            pulse1        : out std_logic;
            data1                : in std_logic;
            latch2        : out std_logic;
            pulse2        : out std_logic;
            data2                : in std_logic;
            buttons1_out: out std_logic_vector(7 downto 0);
            buttons2_out: out std_logic_vector(7 downto 0)
        );


end nes_fsm;

architecture arch of nes_fsm is

       signal latch                              :      std_logic      := '0';
       signal pulse                              :      std_logic      := '0';

       type states is (latch_state, p0, p1, p2, p3, p4, p5, p6, p7,
                                                    10, 11, 12, 13, 14, 15, 16, 17,
done );
```

```
        signal p_s, n_s                         : states;

        signal buttons1                         : std_logic_vector(7 downto 0);
        signal buttons2                         : std_logic_vector(7 downto 0);

        signal counter                          : integer      := 0;                --Clock
for latch/pulse
        signal clock_60hz                   : std_logic := '0';          -- Clock for polling
controller
        signal clock_60hz_counter     : integer      := 0;

        signal in_state_machine             : std_logic    := '0';        -- Flag for state
machine

        constant MICRO_6                        : integer      := 300;
        constant MICRO_12                       : integer      := 600;
        constant MILLI_8P3                  : integer      := 300000;

begin  -- arch

        latch1 <= latch;
        latch2 <= latch;
        pulse1 <= pulse;
        pulse2 <= pulse;

        buttons1_out <= buttons1;
        buttons2_out <= buttons2;



        -- Generate 60HZ clock to poll controllers
        clockprocess_60hz : process(clk)
        begin
                if rising_edge(clk) then

                        if reset = '1' then
                                clock_60hz_counter <= 0;
                                clock_60hz <= '0';
                        else
                                clock_60hz_counter <= clock_60hz_counter + 1;
```

```
                        clock_60hz <= '0'; -- Always zero except for the one cycle it
isnt

                        if clock_60hz_counter >= MILLI_8P3  then
                                clock_60hz_counter <= 0;
                                clock_60hz <= '1'; -- Set clock HIGH here
                        end if;

                end if; -- end reset


        end if; -- rising edge clk
    end process clockprocess_60hz;


        -- Go through state machine to retrieve values
    fsm_like_things:process(clk, clock_60hz, n_s)
    begin
        if rising_edge(clk) then

                latch <= '0';
                pulse <= '0';

                if reset = '1' then
                        counter <= 0;
                        in_state_machine <= '0';
                        n_s <= latch_state;
                        pulse <= '0';
                        latch <= '0';
                        buttons1 <= (others => '0');
                        buttons2 <= (others => '0');

                elsif clock_60hz = '0' and in_state_machine = '0' then
                        -- Do nothing
                else

                        -- Only increment the latch/pulse clock while in the state
machine
```

```
                counter <= counter + 1;

                -- Set signal so next time around, regardless of 60hz clock,
enter in here
                in_state_machine <= '1';

                -- Ask controllers for data nicely with please and thank you
                case p_s is

                        -- Host sending request to controller
                        when latch_state =>
                                if counter > MICRO_12 then
                                        n_s <= l0;
                                        counter <= 0;
                                        latch <= '0';
                                else
                                        n_s <= latch_state;
                                        latch <= '1';
                                end if;



                        -- Get A button
                        -- Wait 6us
                        when l0 =>
                                if counter > MICRO_6 then
                                        n_s <= p0;
                                        counter <= 0;
                                else
                                        n_s <= l0;
                                        buttons1(7) <= not data1;
                                        buttons2(7) <= not data2;
                                end if;

                        -- Pulse
                        when p0 =>
                                if counter > MICRO_6 then
                                        n_s <= l1;
                                        counter <= 0;
                                        pulse <= '0';
                                else
```

```
                n_s <= p0;
                pulse <= '1';
        end if;


-- Get B button
-- Wait 6us
when l1 =>
        if counter > MICRO_6 then
                n_s <= p1;
                counter <= 0;
        else
                n_s <= l1;
                buttons1(6) <= not data1;
                buttons2(6) <= not data2;
        end if;

-- Pulse
when p1 =>
        if counter > MICRO_6 then
                n_s <= l2;
                counter <= 0;
                pulse <= '0';

        else
                n_s <= p1;
                pulse <= '1';
        end if;

-- Get SELECT button
-- Wait 6us
when l2 =>
        if counter > MICRO_6 then
                n_s <= p2;
                counter <= 0;
        else
                n_s <= l2;
                buttons1(5) <= not data1;
                buttons2(5) <= not data2;
        end if;
```

```
-- Pulse
when p2 =>
        if counter > MICRO_6 then
                n_s <= l3;
                counter <= 0;
                pulse <= '0';

        else
                n_s <= p2;
                pulse <= '1';
        end if;

-- Get START button
-- Wait 6us
when l3 =>
        if counter > MICRO_6 then
                n_s <= p3;
                counter <= 0;
        else
                n_s <= l3;
                buttons1(4) <= not data1;
                buttons2(4) <= not data2;
        end if;

-- Pulse
when p3 =>
        if counter > MICRO_6 then
                n_s <= l4;
                counter <= 0;
                pulse <= '0';
        else
                n_s <= p3;
                pulse <= '1';
        end if;

-- Get UP button
-- Wait 6us
when l4 =>
        if counter > MICRO_6 then
```

```
                n_s <= p4;
                counter <= 0;
        else
                n_s <= l4;
                buttons1(3) <= not data1;
                buttons2(3) <= not data2;
        end if;

-- Pulse
when p4 =>
        if counter > MICRO_6 then
                n_s <= l5;
                counter <= 0;
                pulse <= '0';

        else
                n_s <= p4;
                pulse <= '1';
        end if;



-- Get DOWN button
-- Wait 6us
when l5 =>
        if counter > MICRO_6 then
                n_s <= p5;
                counter <= 0;
        else
                n_s <= l5;
                buttons1(2) <= not data1;
                buttons2(2) <= not data2;
        end if;



-- Pulse
when p5 =>
        if counter > MICRO_6 then
                n_s <= l6;
                counter <= 0;
                pulse <= '0';
```

```
        else
                n_s <= p5;
                pulse <= '1';
        end if;


-- Get LEFT button
-- Wait 6us
when l6 =>
        if counter > MICRO_6 then
                n_s <= p6;
                counter <= 0;
        else
                n_s <= l6;
                buttons1(1) <= not data1;
                buttons2(1) <= not data2;
        end if;

-- Pulse
when p6 =>
        if counter > MICRO_6 then
                n_s <= l7;
                counter <= 0;
                pulse <= '0';


        else
                n_s <= p6;
                pulse <= '1';
        end if;



-- Get RIGHT button
-- Wait 6us
when l7 =>
        if counter > MICRO_6 then
                n_s <= p7;
                counter <= 0;
        else
                n_s <= l7;
```

```
                        buttons1(0) <= not data1;
                        buttons2(0) <= not data2;
                end if;

        -- Pulse
        when p7 =>
                if counter > MICRO_6 then
                        n_s <= done;
                        counter <= 0;
                        pulse <= '0';

                else
                        n_s <= p7;
                        pulse <= '1';
                end if;


        when done =>
                if counter > MICRO_6 then
                        n_s <= latch_state;
                        counter <= 0;
                        in_state_machine <= '0'; -- Reset flag
                else
                        n_s <= done;
                end if;


        when others =>
                counter <= 0;
                n_s <= latch_state;
                pulse <= '0';
                latch <= '0';
                in_state_machine <= '0';

        end case;

    end if; -- clock_60hz
end if; -- rising edge clk
```

```
            ---------------------------
            -- Set the new state to p_s
            p_s <= n_s;
            ---------------------------

    end process fsm_like_things;
end arch;
```