**ZJU-UIUC INSTITUTE**

Zhejiang University/University of Illinois at Urbana-Champaign Institute

**ECE385**

Fall 2021

Lab Report

Final Project

# FGPA Implementation of Fireboy and Watergirl


Liu Peiyuan ID: 3190112158

Song Yifei ID: 3190110099

Lab Section: D225-Morning

TA: Wang Lianjie


**January 6, 2022**

# Contents

# 1 Introduction

Our final project was to create a classical game: Fireboy & Watergirl in the Forest Temple. We did this by modifying the SystemVerilog code from Lab 8. We recreated all the elements of the first level, including Watergirl and Fireman controls, buttons, boards, box, and the music of the game. This design allows you to have almost the same experience as the original game. Through this final project, we have further improvement of the use of finite state machine in System Verilog, control of keyboard in NIOS II E and design ideas of a large project.


Figure 1: Start Page of Our game

# 2 Keyboard Control

Fireman's movement is controlled with keys 8, 6, 4, moving him up, left and right. And Watergirl's movement is controlled with keys W, A, D, moving her up, left and right. The original game has the same key control of Watergirl but has keys ↑, ← and → to control the Fireboy. The reason we changed the Fireboy key is that when we used the same key as the original, pressing all four keys at once caused the input key to change to 10101010. We couldn't solve this problem, so we changed the key that controls fireboy to 8,6,4. However, we gave the user instruction at the start screen (see figure below).

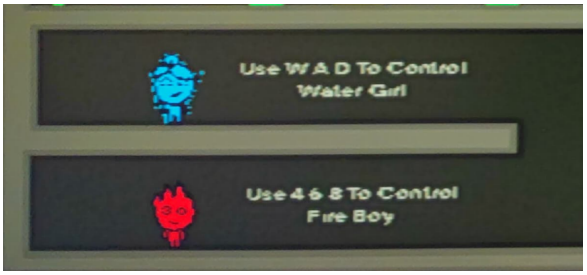


Figure 2: Instructions of keyboard control

## 3 Game Rules

The map of the game is shown below. The game's victory condition is that Fireboy and Watergirl walk to their respective doors in the upper right corner of the screen, while the game's failure condition is that Fireboy enters the water pool or poison pool or Watergirl enters the lava or poison pool. In addition, Fireboy and Watergirl had to cooperatively press buttons to raise and lower the corresponding board and cooperatively push boxes to reach places they couldn't jump.


Figure 3: Map of the Game

## 4 Block Diagram for main modules

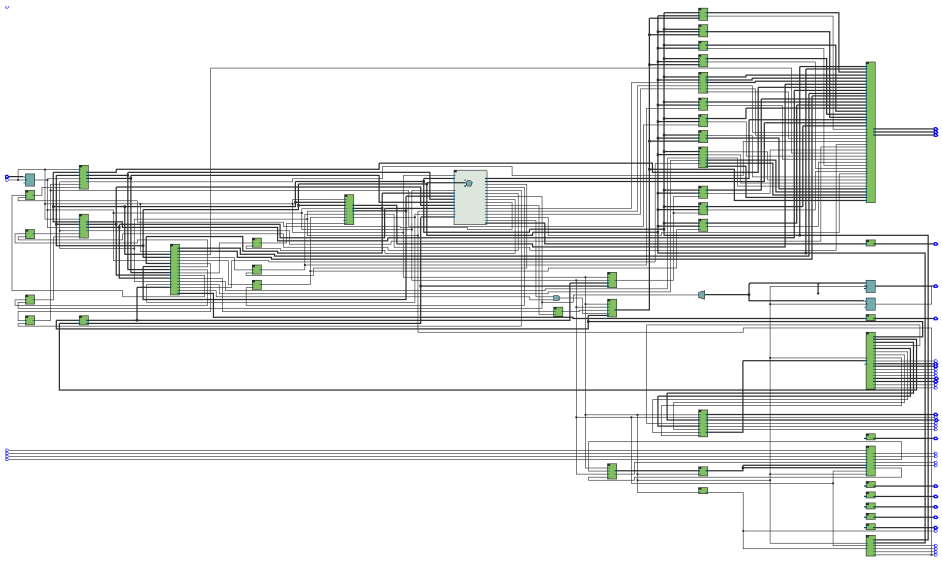

Figure 4: Top level

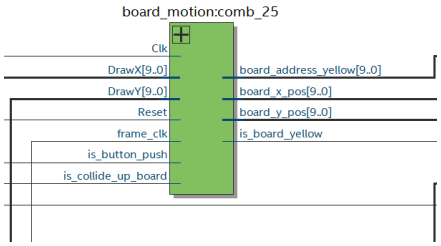


Figure 5: Brown board motion

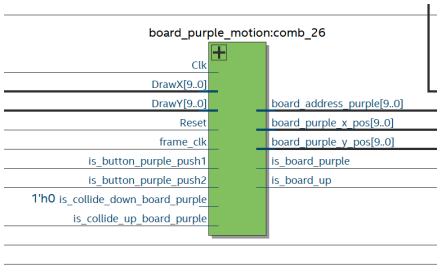


Figure 6: Purple board motion

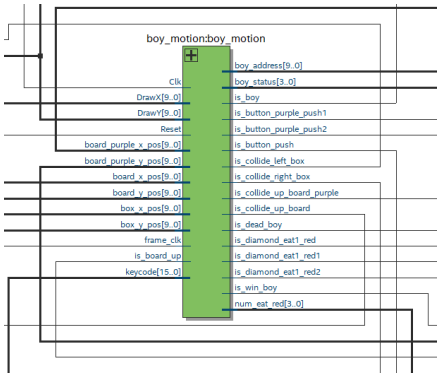


Figure 7: Fireboy motion

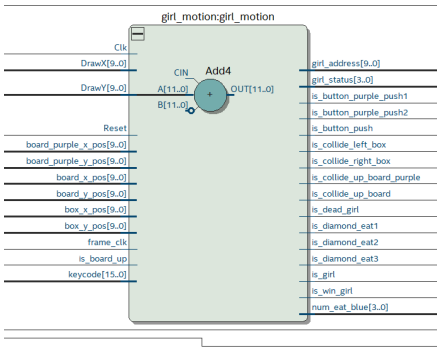


Figure 8: Watergirl motion

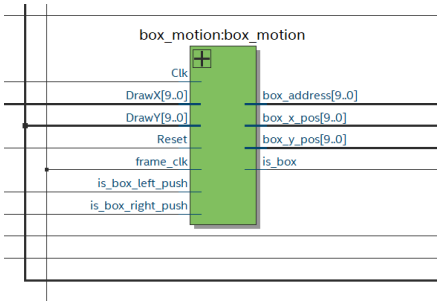


Figure 9: Box motion

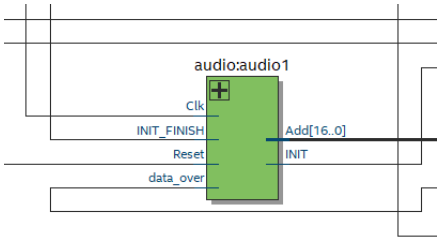


Figure 10: Audio

# 5 Module Descriptions

## 5.1 hpi_io_intf.sv

```
module hpi_io_intf( input         Clk, Reset,
                    input  [1:0]  from_sw_address,
                    output[15:0]  from_sw_data_in,
                    input  [15:0] from_sw_data_out,
                    input         from_sw_r, from_sw_w, from_sw_cs, from_sw_reset, // Active low
                    inout  [15:0] OTG_DATA,
                    output[1:0]   OTG_ADDR,
                    output        OTG_RD_N, OTG_WR_N, OTG_CS_N, OTG_RST_N // Active low
                  );
```

**Description:** The variables containing sw are updated by outside modules, and the otg variables are updated to the sw variables as needed. otg variables represent the USB chip inputs. **Purpose:** Variables in here need to be updated to ensure correct data reads and writes. The module updates the otg signals such that the C code can correctly read and write from the USB keyboard, or be reset.

## 5.2 VGA_controller.sv

```
module  VGA_controller (input             Clk,        // 50 MHz clock
                        input             Reset,      // Active-high reset signal
                        output logic      VGA_HS,     // Horizontal sync pulse.  Active low
                        output logic      VGA_VS,     // Vertical sync pulse.  Active low
                        input             VGA_CLK,    // 25 MHz VGA clock input
                        output logic      VGA_BLANK_N,// Blanking interval indicator.  Active low.
                        output logic      VGA_SYNC_N, // Composite Sync signal.  Active low.  We don't use it in this lab,
                                                      // but the video DAC on the DE2 board requires an input for it.
                        output logic [9:0] DrawX,     // horizontal coordinate
                                           DrawY      // vertical coordinate
                        );
```

**Description:** Takes the data of the ball's current position on screen and updates it to the VGA display with the variables DrawX and DrawY.
**Purpose:** Serves as a compatibility file for the VGA display on the board to the available VGA display. Also serves as an available reset for said display with the reset button of the FPGA.

## 5.3 lab8.sv

```
module lab8( input               CLOCK_50,
             input        [3:0]  KEY,          //bit 0 is set up as Reset
             output logic [6:0]  HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7,
             output logic [7:0]  LEDG,
             // VGA Interface
             output logic [7:0]  VGA_R,        //VGA Red
                                 VGA_G,        //VGA Green
                                 VGA_B,        //VGA Blue
             output logic        VGA_CLK,      //VGA Clock
                                 VGA_SYNC_N,   //VGA Sync signal
                                 VGA_BLANK_N,  //VGA Blank signal
                                 VGA_VS,       //VGA virtical sync signal
                                 VGA_HS,       //VGA horizontal sync signal
             // CY7C67200 Interface
             inout  wire  [15:0] OTG_DATA,     //CY7C67200 Data bus 16 Bits
             output logic [1:0]  OTG_ADDR,     //CY7C67200 Address 2 Bits
             output logic        OTG_CS_N,     //CY7C67200 Chip Select
                                 OTG_RD_N,     //CY7C67200 Write
                                 OTG_WR_N,     //CY7C67200 Read
                                 OTG_RST_N,    //CY7C67200 Reset
             input               OTG_INT,      //CY7C67200 Interrupt
             // SDRAM Interface for Nios II Software
             output logic [12:0] DRAM_ADDR,    //SDRAM Address 13 Bits
             inout  wire  [31:0] DRAM_DQ,      //SDRAM Data 32 Bits
             output logic [1:0]  DRAM_BA,      //SDRAM Bank Address 2 Bits
             output logic [3:0]  DRAM_DQM,     //SDRAM Data Mast 4 Bits
             output logic        DRAM_RAS_N,   //SDRAM Row Address Strobe
                                 DRAM_CAS_N,   //SDRAM Column Address Strobe
                                 DRAM_CKE,     //SDRAM Clock Enable
                                 DRAM_WE_N,    //SDRAM Write Enable
                                 DRAM_CS_N,    //SDRAM Chip Select
                                 DRAM_CLK,      //SDRAM Clock
             // Add for audio part
             input               AUD_ADCDAT,
             input               AUD_DACLRCK,
             input               AUD_ADCLRCK,
             input               AUD_BCLK,
             output logic        I2C_SCLK,
                                 I2C_SDAT,
                                 AUD_XCK,
                                 AUD_DACDAT);
```

**Description:** Serves as the top level module and provides the output data for the C code to assign to the board and the usb cable. Controls logic to update room variables.
**Purpose:** Instantiates all the other System Verilog modules and links together the entire program with the help of Qsys.

## 5.4 background.sv

```
module background
(
    input  logic [3:0]    status,           // Whether now is the background page
    input  logic [9:0] DrawX, DrawY,        // Current pixel coordinates
    output logic is_background,             // Whether current pixel belongs to background
    output logic [16:0] background_address  // address for color mapper to figure out what color the logo pixel should be
);
```

**Description:** Caculate the display address of the background.
**Purpose:** Display the background to the screen.

### 5.5 map1.sv

```
module map1
(
      input  logic [3:0] status,          // Whether now is the map page
      input  logic [9:0] DrawX, DrawY,       // current pixel coordinates
      output logic is_map1,               // Whether current pixel belongs to map
      output logic [16:0] map1_address    // address for color mapper to figure out what color the map pixel should be
);
```

**Description:** Caculate the display address of the map.
**Purpose:** Display the map to the screen.

### 5.6 button.sv

```
module button
(
      input  logic [9:0] DrawX, DrawY,          // Current pixel coordinates
      input  logic is_button_purple_push1,
      output logic is_button,
      output logic [7:0] button_address
);
```

**Description:** Caculate the display address of one of the purple button.
**Purpose:** Display one of the purple button to the screen.

### 5.7 button1.sv

```
module button1
(
      input  logic [9:0] DrawX, DrawY,          // Current pixel coordinates
      input  logic is_button_purple_push2,
      output logic is_button1,
      output logic [7:0] button_address1
);
```

**Description:** Caculate the display address of another purple button.
**Purpose:** Display another purple button to the screen.

### 5.8 button_yellow.sv

```
module button_yellow
(
      input  logic [9:0] DrawX, DrawY,
      input  logic is_button_push,
      output logic is_button_yellow,
      output logic [7:0] button_yellow_address
);
```

**Description:** Caculate the display address of yellow button.
**Purpose:** Display the yellow button to the screen.

### 5.9 girl_word.sv

```
module girl_word
(
      input  logic keyboardpress,
      input  logic [9:0] DrawX, DrawY,
      output logic is_girlword,
      output logic [11:0] girlword_address
);
```

**Description:** Caculate the display address of girl's instruction word.
**Purpose:** Display the girl's instruction word to the screen.

### 5.10 boy_word.sv

```
module boy_word
(
      input  logic keyboardpress,
      input  logic [9:0] DrawX, DrawY,
      output logic is_boyword,
      output logic [11:0] boyword_address
);
```

**Description:** Caculate the display address of boy's instruction word.
**Purpose:** Display the boy's instruction word to the screen.

### 5.11 collision.sv

```
module collision
(
        input    [9:0] x,
        input    [9:0] y,
        input    [6:0] width,
        input    [6:0] height,
        output   is_collide_up,
                 is_collide_down,
                 is_collide_left,
                 is_collide_right,
                 // Modify
                 is_collide_left_end,
                 is_collide_right_end,
                 is_collide_left_top,
                 is_collide_right_top
    );
```

**Description:** This module is used to calculate whether the girl or the boy is collide with the wall based on the color of the map.
**Purpose:** girl_motion or boy_motion will use this module to determine the motion of girl or boy.

### 5.12 collision_board.sv

```
module collision_board
(
        input    [9:0] x,
        input    [9:0] y,
        input    [6:0] width,
        input    [6:0] height,
        input    [9:0] board_x_pos,
        input    [9:0] board_y_pos,
        input    [9:0] board_purple_x_pos,
        input    [9:0] board_purple_y_pos,
        output   is_collide_down_board,
        output   is_collide_up_board,
        output   is_collide_left_board,
        output   is_collide_up_board_purple,
        output   is_collide_down_board_purple,
        output   is_collide_right_board_purple
    );
```

**Description:** This module is used to calculate whether the girl or the boy is collide with the yellow board based on the coordinate.
**Purpose:** girl_motion or boy_motion will use this module to determine the motion of girl or boy.

### 5.13 box_collide.sv

```
module box_collide
(
        input    [9:0] x,
        input    [9:0] y,
        input    [6:0] width,
        input    [6:0] height,
        output   is_collide_down,
                 is_collide_left,
                 is_collide_right

);
```

**Description:** This module is used to calculate whether the box is collide with the right wall, left wall or the down wall.
**Purpose:** box_motion will use this module to determine the motion of box.

### 5.14 collision_box.sv

```
module collision_box
(
         input    [9:0] x,
         input    [9:0] y,
         input    [6:0] width,
         input    [6:0] height,
         input    [9:0] box_x_pos,
         input    [9:0] box_y_pos,
         output   is_collide_left_box,
         output   is_collide_right_box,
         output   is_collide_down_box
      );
```

**Description:** This module is used to calculate whether the girl or the boy is collide with the box based on the coordinate.
**Purpose:** girl_motion or boy_motion will use this module to determine the motion of girl or boy.

### 5.15 red_diamond.sv

```
module red_diamond
(
     input  logic [9:0] DrawX, DrawY,         // Current pixel coordinates
     input  logic is_diamond_eat1_red,
     input  logic is_diamond_eat1_red1,
     input  logic is_diamond_eat1_red2,
     output logic is_red_diamond,             // Whether current pixel belongs to red_diamond
     output logic is_red_diamond1,
     output logic is_red_diamond2,
     output logic [8:0] red_diamond_address,     // address for color mapper to figure out what color the logo pixel should be
     output logic [8:0] red_diamond_address1,
     output logic [8:0] red_diamond_address2
);
```

**Description:** This module is to display all the three red diamonds on the map.
**Purpose:** Determine the address of each red diamond.

### 5.16 blue_diamond.sv

```
module blue_diamond
(
      input  logic [9:0] DrawX, DrawY,          // Current pixel coordinates
      input  logic is_diamond_eat1,
      input  logic is_diamond_eat2,
      input  logic is_diamond_eat3,
      output logic is_blue_diamond,             // Whether current pixel belongs to blue_diamond
      output logic is_blue_diamond1,
      output logic is_blue_diamond2,
      output logic [8:0] blue_diamond_address,
      output logic [8:0] blue_diamond_address1,
      output logic [8:0] blue_diamond_address2
);
```

**Description:** This module is to display all the three blue diamonds on the map.
**Purpose:** Determine the address of each blue diamond.

### 5.17 is_red_diamond_eat.sv

```
module is_red_diamond_eat
(
        input    Clk,
        input    Reset,
        input    [9:0] x,
        input    [9:0] y,
        input    [6:0] width,
        input    [6:0] height,
        output   is_diamond_eat1_red,
        output   is_diamond_eat1_red1,
        output   is_diamond_eat1_red2,
        output [3:0] num_eat_red
    );
```

**Description:** This module uses FSM to determine whether each red diamond is eaten.
**Purpose:** Determine whether each red diamond is eaten.

### 5.18 is_blue_diamond_eat.sv

```
module is_blue_diamond_eat
(
        input    Clk,
        input    Reset,
        input    [9:0] x,
        input    [9:0] y,
        input    [6:0] width,
        input    [6:0] height,
        output   is_diamond_eat1,
        output   is_diamond_eat2,
        output   is_diamond_eat3,
        output [3:0] num_eat_blue
    );
```

**Description:** This module uses FSM to determine whether each blue diamond is eaten.
**Purpose:** Determine whether each blue diamond is eaten.

### 5.19 girl_motion.sv

```
module  girl_motion
(               input         Clk,                // 50 MHz clock
                              Reset,              // Active-high reset signal
                              frame_clk,          // The clock indicating a new frame (~60Hz)
                input [9:0]   DrawX, DrawY,       // Current pixel coordinates
                input [9:0]   board_x_pos, board_y_pos,
                input [9:0]   board_purple_x_pos, board_purple_y_pos,
                input [9:0]   box_x_pos, box_y_pos,
                input [15:0]   keycode,       // added input for keycode
                input         is_board_up,
                output logic is_girl,             // Whether current pixel belongs to ball or background
                output logic [3:0]  girl_status,
                output logic [9:0] girl_address,
                output logic is_dead_girl,
                output logic is_win_girl,
                output logic is_diamond_eat1,
                output logic is_diamond_eat2,
                output logic is_diamond_eat3,
                output logic [3:0] num_eat_blue,
                output logic is_button_push,
                output logic is_button_purple_push1,
                output logic is_button_purple_push2,
                output logic is_collide_up_board,
                output logic is_collide_up_board_purple,
                output logic is_collide_left_box,
                output logic is_collide_right_box
              );
```

**Description:** This module is the essential part of the motion of the girl, consists of left, right, up motion. Particularly, whether the girl is collide with walls or elements, whether the girl is dead etc. This moduel is mainly implemented with FSM.
**Purpose:** Determine the motion of girl.

### 5.20 boy_motion.sv

```
module  boy_motion
(               input           Clk,                // 5c MHz clock
                                Reset,              // Active-high reset signal
                                frame_clk,          // The clock indicating a new frame (~60Hz)
                input [9:0]     DrawX, DrawY,       // Current pixel coordinates
                input [9:0]     board_x_pos, board_y_pos,
                input [9:0]     board_purple_x_pos, board_purple_y_pos,
                input [9:0]     box_x_pos, box_y_pos,
                input [15:0]    keycode,        // added input for keycode
                input           is_board_up,
                output logic is_boy,            // Whether current pixel belongs to ball or background
                output logic [3:0]  boy_status,
                output logic [9:0] boy_address,
                output logic is_dead_boy,
                output logic is_win_boy,
                output logic is_diamond_eat1_red,
                output logic is_diamond_eat1_red1,
                output logic is_diamond_eat1_red2,
                output logic [3:0] num_eat_red,
                output logic is_button_push,
                output logic is_button_purple_push1,
                output logic is_button_purple_push2,
                output logic is_collide_up_board,
                output logic is_collide_up_board_purple,
                output logic is_collide_left_box,
                output logic is_collide_right_box
              );
```

**Description:** This module is the essential part of the motion of the boy, consists of left, right, up motion. Particularly, whether the boy is collide with walls or elements, whether the boy is dead etc. This moduel is mainly implemented with FSM.
**Purpose:** Determine the motion of boy.

### 5.21 board_motion.sv

```
module  board_motion
(               input           Clk,                // 50 MHz clock
                                Reset,              // Active-high reset signal
                                frame_clk,          // The clock indicating a new frame (~60Hz)
                input [9:0]     DrawX, DrawY,       // Current pixel coordinates
                input           is_button_push,
                input           is_collide_up_board,
                output logic    is_board_yellow,
                output logic [9:0] board_address_yellow,
                output logic [9:0] board_x_pos, board_y_pos
              );
```

**Description:** This module is the essential part of the motion of the yellow board. Particularly, whether the board is down, whether the board reaches the right place etc. This moduel is mainly implemented with FSM.
**Purpose:** Determine the motion of the yellow board.

### 5.22 board_purple_motion.sv

```
module  board_purple_motion
(               input           Clk,                // 50 MHz clock
                                Reset,              // Active-high reset signal
                                frame_clk,          // The clock indicating a new frame (~60Hz)
                input [9:0]     DrawX, DrawY,       // Current pixel coordinates
                input           is_button_purple_push1,
                input           is_button_purple_push2,
                input           is_collide_up_board_purple,
                input           is_collide_down_board_purple,
                output logic    is_board_purple,            // Whether current pixel belongs to ball or background
                output logic [9:0] board_address_purple,
                output logic [9:0] board_purple_x_pos, board_purple_y_pos,
                output logic is_board_up
              );
```

**Description:** This module is the essential part of the motion of the purple board. Particularly, whether the board is down or up, whether the board reaches the right place based on the button pushed. This moduel is mainly implemented with FSM.
**Purpose:** Determine the motion of the purple board.

### 5.23 box_motion.sv

```
module  box_motion
(               input           clk,                // 50 MHz clock
                                Reset,              // Active-high reset signal
                                frame_clk,          // The clock indicating a new frame (~60Hz)
                input [9:0]     DrawX, DrawY,       // Current pixel coordinates
                input           is_box_left_push,
                input           is_box_right_push,
                output logic    is_box,
                output logic [9:0] box_address,
                output logic [9:0] box_x_pos, box_y_pos
              );
```

**Description:** This module is the essential part of the motion of the box. Particularly, whether the board is left push, right push or down. This moduel is mainly implemented with FSM.
**Purpose:** Determine the motion of the box.

### 5.24 win_girl.sv

```
module win_girl
(
        input    [9:0] x,
        input    [9:0] y,
        input    [6:0] width,
        input    [6:0] height,
        output   is_win_girl
        );
```

**Description:** This module determines whether the girl reaches the right region to win.
**Purpose:** Determine whether the girl is win.

### 5.25 win_boy.sv

```
module win_boy
(
        input    [9:0] x,
        input    [9:0] y,
        input    [6:0] width,
        input    [6:0] height,
        output   is_win_boy
        );
```

**Description:** This module determines whether the boy reaches the right region to win.
**Purpose:** Determine whether the boy is win.

### 5.26 dead_girl.sv

```
module dead_girl
(
        input    [9:0] x,
        input    [9:0] y,
        input    [6:0] width,
        input    [6:0] height,
        output   is_dead_girl
        );
```

**Description:** This module determines whether the girl touches the lava or poission.
**Purpose:** Determine whether the girl is dead.

### 5.27 dead_boy.sv

```
module dead_boy
(
        input    [9:0] x,
        input    [9:0] y,
        input    [6:0] width,
        input    [6:0] height,
        output   is_dead_boy
        );
```

**Description:** This module determines whether the boy touches the water pool or poission.
**Purpose:** Determine whether the boy is dead.

### 5.28 button_push.sv

```
module button_push
(
        input    Clk,
        input    Reset,
        input    [9:0] x,
        input    [9:0] y,
        input    [6:0] width,
        input    [6:0] height,
        output   is_button_push
    );
```

**Description:** This module determines whether the yellow button is pushed based on the co-ordinate.
**Purpose:** Determine whether the yellow button is pushed.

### 5.29 button_puple_push1.sv

```
module button_purple_push1
(
        input    Clk,
        input    Reset,
        input    [9:0] x,
        input    [9:0] y,
        input    [6:0] width,
        input    [6:0] height,
        output   is_button_purple_push1
    );
```

**Description:** This module determines whether one of the purple button is pushed based on the coordinate.
**Purpose:** Determine whether one of the purple button is pushed.

### 5.30 button_puple_push2.sv

```
module button_purple_push2
(
        input    Clk,
        input    Reset,
        input    [9:0] x,
        input    [9:0] y,
        input    [6:0] width,
        input    [6:0] height,
        output   is_button_purple_push2
    );
```

**Description:** This module determines whether another purple button is pushed based on the coordinate.
**Purpose:** Determine whether another purple button is pushed.

### 5.31 designer.sv

```
module designer
(
      input  logic [9:0] DrawX, DrawY,
      output logic is_designer,
      output logic [12:0] designer_address
);
```

**Description:** Caculate the display address of the designer text.
**Purpose:** Display the designer text to the screen.

### 5.32 color_mapper.sv

```
module  color_mapper
(
                       input  logic [3: 0]   status,
                       input  logic          is_gameover,
                       input  logic          is_girl, is_boy,
                       input  logic          is_girlword, is_boyword,
                       input  logic [3: 0]   girl_status, boy_status,
                       input  logic          is_background,
                       input  logic          is_map1,
                       input  logic          is_blue_diamond,
                       input  logic          is_blue_diamond1,
                       input  logic          is_blue_diamond2,
                       input  logic          is_red_diamond,
                       input  logic          is_red_diamond1,
                       input  logic          is_red_diamond2,
                       input  logic          is_board,
                       input  logic          is_board_yellow,
                       input  logic          is_button,
                       input  logic          is_button1,
                       input  logic          is_button_yellow,
                       input  logic          is_box,
                       input  logic          is_designer,
                       input  logic [16:0]   map1_address,
                       input  logic [16:0]   background_address,
                       input  logic [9:0]    girl_address, boy_address,
                       input  logic [11:0]   girlword_address, boyword_address,
                       input  logic [8:0]    blue_diamond_address,
                       input  logic [8:0]    blue_diamond_address1,
                       input  logic [8:0]    blue_diamond_address2,
                       input  logic [8:0]    red_diamond_address,
                       input  logic [8:0]    red_diamond_address1,
                       input  logic [8:0]    red_diamond_address2,
                       input  logic [9:0]    board_address,
                       input  logic [9:0]    board_address_yellow,
                       input  logic [7:0]    button_address,
                       input  logic [7:0]    button_address1,
                       input  logic [7:0]    button_yellow_address,
                       input  logic [9:0]    box_address,
                       input  logic [12:0]   designer_address,
                       input  logic [11:0]   gameover_address,
                       input  logic [11:0]   gamewin_address,
                       input  logic [9: 0]   DrawX, DrawY,              // Current pixel coordinates
                       output logic [7: 0]   VGA_R, VGA_G, VGA_B        // VGA RGB output
);
```

**Description:** Used to determine what to draw at a given pixel, sending the necessary RGB values to the VGA outputs. It holds modules used to load sprites, and updates inputs into them (such as the enemy's alive status in the door module) to ensure certain sprites are not drawn before we want them to be displayed.
**Purpose:** Used to update VGA RGB values for printing to the screen.

### 5.33 game_logic.sv

```
module game_logic
(
        input  logic         Clk,           // 50 MHz clock
        input  logic         Reset,         // Active-high reset signal
        input  logic         is_dead,
        input  logic         is_win,
        input  logic   [15:0] keycode,         // Input from the keycode
        output logic   [3:0] status         // Status of the game {Background, Begin, Win, Loose}
);
```

**Description:** This module is used to determine the top level logic of the whole game. It store the logic in the "status" logic. "status": 0001 represents background, 0010 represents map1, 0100 represents lose, 1000 represents win.
**Purpose:** Used to determine the total game logic of the game.

### 5.34 HexDriver.sv

```
module HexDriver (input  [3:0]  In0,
                  output logic [6:0]  Out0);
```

**Description:** This is supposed to drive the LEDs on FPGA and display the result.
**Purpose:** We use this module to display the pressed keyboard.

### 5.35 select.sv

```
module select(
            input  logic in0, in1,
            output logic out);
```

**Description:** This is the OR gate implement.
**Purpose:** We use this module to merge some signals.

### 5.36 keycode_select.sv

```
module keycode_select(
            input  logic [15:0] keycode, keycode0,
            output logic [15:0] keycode_girl, keycode_boy);
```

**Description:** This module calculates the real key value received by the girl or boy based on the key value entered by the user. With this module, keys received by our girl or boy are not interfered with by irrelevant keys.
**Purpose:** Used to determine the key value of the girl and the boy.

### 5.37 audio.sv

```
module audio ( input logic Clk, Reset,
               input logic INIT_FINISH,
               input logic data_over,
               output logic INIT,
               output [16:0] Add
);
```

**Description:** This module implement a finite state machine to read the hex values provided in the music file.
**Purpose:** It can control the length of read data and the speed of reading which is extremely important for tuning the music.

### 5.38 music.sv

```
module music (input logic  Clk,
              input logic  [16:0]Add,
              output logic [16:0]music_content);
```

**Description:** This module is implemented for reading the music bytes into the on-chip memory.
**Purpose:** It can control the length of read data the size storage unit.

# 6 baseline and additional features

**Baseline features implemented**:

For baseline edition, we have implemented a single-player version of the first chapter of the classic game fireboy and watergirl. There is a start screen at the beginning of the whole game. The end screen can either be "Game Over" which informs the player of the death or "Success" which congratulates the player for a good game. We have designed three kinds of movement for the player which include moving left, moving right and jump. These movements are controlled by the player through the USB input from the keyboard (A, W, D). For the interface, we used NIOS II to connect the keyboard to determine the movement of the character. Also, we designed finite state machines for the character whihc enables the character automatically push the box when directly touching it and push the button on the ground by standing on it. We set up the frame of the ground, ceil and wall which could stop the character from moving out of the map by judging the color at the detection points around the center of the sprite. Also we used a counter to record the number of diamonds collected by the player and show it on the hex display of the DE2 board. The character starts running from the bottom and ends with success at the top gates. Falling into the toxic liquid and the liquid with different color from the sprite itself is considered as a failure(death). This is also implemented by detecting the color at the detection points around the center of the sprites.

**Additional features implemented**:

After we successfully implemented almost all the baseline features of one player version. The next target becomes implementing two-player version. We draw new sprites and turn them into hex values for the DE2 board to load. Then we design the finite state machine for the other character which is similar to the first one. To recieve six keys at the same time, we also changed the platform designer to add two more instance of keycode and changed all of them to 16bit wise. Dealing with the unordered keyboard input needed us to implementing additional logic which can make correct response after receiving unordered key input. Instead of simply OR the keyword together, we also add logic gates to avoid invalid input infecting the regular game logic. Therefore, theoretically, mis-touching other unrelated keys will not affect the action of the sprites. The last part of the additional feature is adding background sound. We refer to the code provided on Github which could convert the wav audio file into hex file. After connecting the audio interface in the top level design, the background sound could be played normally.

**Additional features not implemented**:

Some of the additional features in our proposal are not implemented due to the time limitation and their high difficulty. The limitation of hardware should also be taken into consideration. Originally, we plan to show the number of collected diamond on the screen, but we end up with showing the numbers on the hex display of DE2 board. Although we successfully added background music, but we did not implement sound effects of collecting diamonds and winning. Based on the sophisticated finite state machine of the game logic, add those sound effects would be more than tedious. Other than that, we did not use the SRAM interface which is more ef-

ficient than using on-chip memory. While, after compressing the original photo of background and sprites, we managed to succeed with on-chip memory, so that is not a severe problem for us.

# 7 Platform Designer and PIO Modules

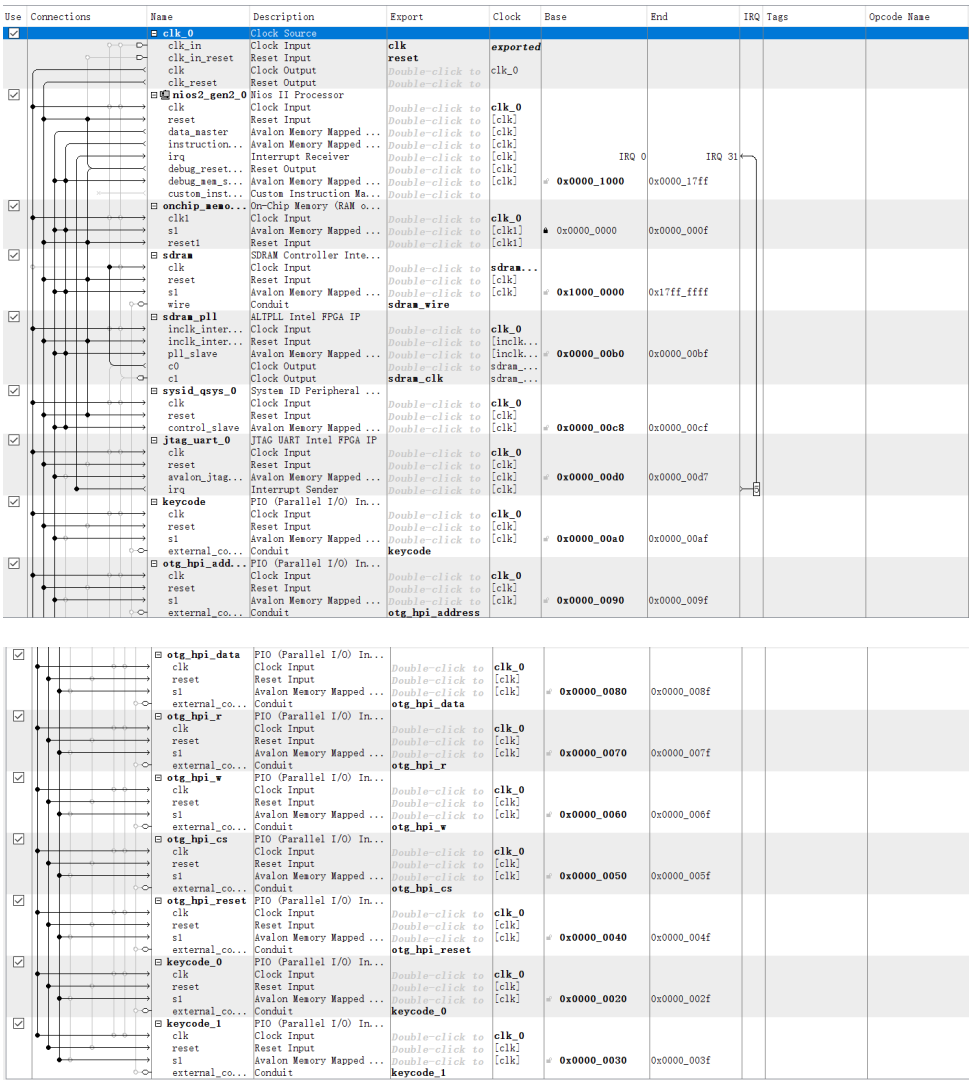


Figure 11: Platform Designer and Qsys Modules

**otg_hpi_data:**
**Input / Output:** [15:0] otg_hpi_data
**Base Address** $0x80 - 0x8f$
**Description:** This is 16 bits data that will be placed in hpi-data register.
**Purpose:** We need this PIO to transfer data between USB chip and NIOS II.

**otg_hpi_address:**
**Output:** [1:0] keycode_export
**Base Address** $0x90 - 0x9f$
**Description:** The signal set the hpi registers we are doing operation on.
**Purpose:** Depending on the operation, the signal will be set to different value.

**otg_hpi_cs:**

**Output:** [15:0] keycode_export
**Base Address** $0x50 - 0x5f$
**Description:** This is set 0 when we want to IOWrite or IORead.
**Purpose:** It is needed to do USBWrite or USBRead.

**keycode:**
**Input:** clk, reset
**Output:** [15:0] keycode_export
**Base Address** $0xa0 - 0xaf$
**Description:** This is the data of the pressed key.
**Purpose:** We will take in the data from keyboard and pass down to the FPGA.

**otg_hpi_r:**
**Output:** otg_hpi_r
**Base Address** $0x70 - 0x7f$
**Description:** This signal is set 1 when you read data from hpi registers.
**Purpose:** This is needed to enable IO_Read.

**otg_hpi_w:**
**Output:** otg_hpi_w
**Base Address** $0x60 - 0x6f$
**Description:** This signal is set 0 when you write data from hpi registers.
**Purpose:** This is needed to enable IO_Write.

**otg_hpi_reset:**
**Output:** otg_hpi_reset
**Base Address** $0x40 - 0x4f$
**Description:** This module is used when resetting the memory of SoC.
**Purpose:** It allows for the reset of the memory of the SoC sent from the software to the FPGA.

**keycode_0:**
**Input:** clk, reset
**Output:** [15:0] keycode_export
**Base Address** $0x20 - 0x2f$
**Description:** This is the data of the pressed key.
**Purpose:** We will take in the data from keyboard and pass down to the FPGA.

**keycode_1:**
**Input:** clk, reset
**Output:** [15:0] keycode_export
**Base Address** $0x30 - 0x3f$
**Description:** This is the data of the pressed key.
**Purpose:** We will take in the data from keyboard and pass down to the FPGA.

# 8 Design and Resource Table

| | |
|---|---|
| LUT | 76028 |
| DSP | 0 |
| Memory(BRAM) | 1611408 |
| Flip-Flop | 2567 |
| Frequency | 122.85MHz |
| Static Power | 110.58 mW |
| Dynamic Power | 0.83 mW |
| Total Power | 192.09 mW |

Table 1: Design resources and statistics

# 9 Timeline Predictions and Results

**Week 1:** We tried to understand and plan what is needed for the game. including collecting the original photos of fireboy, watergirl, background, obstacles, motion, etc. The main achievement is that we learned to use the PNGtoHEX converter code provided on blackboard and we converted the watergirl and background to hex file in order to print them on the screen.

**Week 2:** We started the groundwork for the baseline of the game. We further optimize the resolution of the map and the sprites to accelerate the compiling speed of the project. Then we design the basic logic of the game and the floors and other elements of the game, including the moving logic of the character. Apart from dealing with the game designing, We also started documenting the progress of the whole project. of this project.

**Week 3:** Generally, we implemented the baseline features, including the collision detection of the watergirl which allows the waterfirl to walk on different floors without any bugs. We faced a weird problem that the watergirl can not move left/right and move up at the same time. Therefore, we designed a sophisticated finite state machine to solve that problem and make the motion of the watergirl smoother.

**Week 4:** After the watergirl module became bug-free, we moved on and added the fireboy module to the gamie logic. Then comes the moving boards and box mouduels. Those are the most annoying ones in the whole design process, we keep debugging those two moduels until the last minute before demo.

**Week 5:** The game logic are basically complete. Then we added the audio part into our project and successfully played the original BGM of the game. For better completeness of the whole game, we added welcome interface and ending screens to our game which tell the user what key to press and how to move the character in the game.

# 10 Conclusion

If we had more time, we would want to extend the functionality we perform, such as score display and trigger controlled boards. When we finish the code for the immovable object,including background and walls, we moved on to write other movable modules, such as the watergirl and boards. We were interested in creating modules for immovable objects, but abandoned the idea because of the complexity and didn't want to create new connections among modules. Instead, we use color contrast to detect collision. In the process of designing, we considered

adding triggers for the movement of board, but the implementation of rotation motion is way too complicated. To avoid ending up running out of time, we substitute the trigger with the button. We created implementations of the planned baseline and additional features, but wanted to make the game more collaborative. Therefore, we add a board that the two players must push the button for each other to go to the next floor. We considered implementing a life system that would allow the user to restart the game in 3 attempts before Game Over. Considering the original design of the game, we gave up that idea and decided to give instant death to the character when triggering the death logic. The resolution and quality of sound are also a problem which is caused by the deficiency of memory and compiling time of the whole project. We have to compress the size of the photos and the decrease the bits of the BGM to suit the design into the DE2 board. Although we couldn't finish all the planned features, we have stepped much further than the expected result. We learned how to collaborate when designing a huge project. We learned how to create a game that was complicated and decompose the whole design into small parts and set milestones for ourselves. We did our best to create a game we love using the content we learned from this fantastic class. Thanks to the professors and all the teaching assistants who have helped us a lot in the course ECE 385.

## 11 Appendix: Code and demo video

**Github link:** https://github.com/Hank0626/ECE-385-Final-Project

**Bilibili link:** https://www.bilibili.com/video/BV1Sb4y1Y7nL/