ECE 385

Spring 2020

Final Project

Karate Champ

Simulated Using SOC

Freddy Zhang and Mehul Dugar

Section: ABO (Thu. 3PM)

Gene Shiue

Contents

1 Introduction

The goal of our final project was to implement the game Karate Champ (https://youtu.be/Az308iZZClc) on the FPGA. Our design includes a NIOS II CPU for the purposes of interfacing with the USB keyboard as in lab 8 to emulate movement of the sprites of the game, the VGA interface, and a sound driver. Our goal was to demonstrate our IIe SoC using the USB keyboard and VGA monitor, running our version of Karate Champ with limited yet functional game elements such as a title screen, attacking sequence, etc. We were able to implement this game using the knowledge of the tools that we acquired during the course of this semester as well as helpful guides available on the course website. This project definitely pushed us to think creatively and make important design choices.

2 Written Description

2.1 Game Logic

Our version of Karate Champ allows two players to play against each other. Each user has control over the movements and attacks of one of the two characters on screen with the help of the assigned left/right and attack keys.

In order to win the game, one of the players has to accumulate a total of ten points by landing attacks on the opponent. Each successful attack that lands on the opponent results in being awarded two points. Once an attack lands successfully, the player hit goes down and new round starts by pressing the “enter” key.

2.2 Feature Description

Baseline Features : Before moving on to additional features, we decided to put in some basic features to make the gameplay as close to the real game as possible without complicating it too much. Some of the baseline features we implemented were -

a) Player Movement: We made our first modifications to the ball.sv file that we had been given in lab 8. We constrained the ball (now the player) to move only in terms of x direction at a fixed y position. Moreover, we added another “ball” (again a placeholder for a player) so that we could extend the functionality to two players. We also constrained the motion so that the players would not go out of the screen and instead stop moving past the edges of the screen. Moreover, the movement animations have been smoothed out so it looks more game-like.
b) Two Player compatibility: Next we extended the functionality of the game to now support two players by extending the keycode input reading capabilities of the project. We expanded the size of the keycode reader module from 8 bit to 32 bit so that it could read four keycode inputs at a time.
c) Attack Mechanism: We implemented ways to attach points and attacks to the players. Each attack that is registered on the other player results in the gain of a point, and the game ends when one of the players collects 10 points. Again, the animations that go along with this mechanism were also implemented.
d) Point System: As mentioned in the previous feature, we added a point system that is displayed on the hexdisplay of the FPGA board.

e) Sprites and Animations: We added the animations to go along with all the actions implemented along with the sprites that were stored in on-chip memory. We used a palette to conserve memory.
f) Title Screen: Something we didn't mention in our proposal was having a title screen, which would be a better introduction to our game than just jumping into it. We implemented a Title screen which one has to get through by clicking a key.

Additional Features : We started working on these quite late and thus were unable to see any of them to fruition. This was due to the unprecedented difficulties we had due to the difference in time-zones and mismatched schedule of midterms. We started implementing a sound module but were unable to get it to work as desired. Additionally, we wanted to implement blocking so that the players can defend themselves from the oncoming attacks. However, due to many bugs that appeared while coding, we were not able to successfully implement it.

3 Flow of Implementation

So the starting point of this project is when is_title is high. This leads to the display of the title screen. After pressing a key, the user moves to the gameplay screen wherein there are two character sprites. These character sprites move and perform actions based on the keys mapped to them. There are two keys to move left or right, two to perform attacks. Each of these successful attacks on the opposing player increases the score by 21. These moves are registered by the hit_calculator module. Then these hits are further used to update the scores using the score_counter module. Whichever player reaches a total of 10 points first is

declared a winner. The winner is declared using the win_con module. And hitting the reset button returns to the start screen. All the animations that consist of the gameplay are implemented using the sv modules that are mentioned in the later section.

4 Design Choices

We only used on-chip memory for the sprites because it was the easiest method of implementation and it made it very fast as well. There was no need to wait for the sprites to be loaded onto the board. Most of our big logic was split into different modules on the top level. This way we were able to easily differentiate which module did what. It also reduced our need to repeat the use of the module inside character_one and character_two. It did get pretty tedious to have to add more inputs and outputs based for another module. For example, we first started with two moving characters. When we wanted to add attack moves, we needed to add more inputs to the character_one and character_two modules so that it knows when to attack based on the keypress. Then, if we wanted to identify when a character gets hit, we needed to add more outputs to each character module to send that signal to the hit_calculator module so it understands when hits are being made.

5 SV Module Descriptions

5.1 HexDriver

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

Purpose : This Module was used to create an interface for the hex display on the FPGA board. We used this to display the scores. It converts 4 bit inputs to Hex Display compatible outputs.

5.2 KarateChamp

```
module KarateChamp( 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,
             // 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
            );
```

Purpose : This module acted as the top-level for our project. All inputs and outputs of the FPGA board are connected to this module. All additional modules were instantiated in this module.

5.3 attack_ani

```
module attack_ani(
                        input logic Clk,
                        input logic Reset,
                        input logic keypress,
                        output logic[1:0] ani
                        );
```

Purpose : This module is used to create the animation for a punch by either of the two players. This module holds a state machine to control the animation.

5.4 VGA_controller

```
module  VGA_controller (input              Clk,         // 50 MHz clock
                                           Reset,       // Active-high reset signal
                        output logic       VGA_HS,      // Horizontal sync pulse.  Active low
                                           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.
                                           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
                        );
```

Purpose : This module is used to control which part of the screen is being drawn on. From the documentation, we were able to find out that it works in a raster order which means that it moves left to right in a row and once it reaches the end, it moves on to the next row. Once it reaches the end of the last row which is in the bottom right corner, it moves back to the top left pixel.

5.5 background

```
module background(
                        input Clk, Reset, frame_clk,
                        input is_title,
                        input [9:0] DrawX, DrawY,
                        output logic [2:0] is_back
                        );
```

Purpose : This module is used to draw the background sprites onto the screen. It picks between rocks or blocks based on the coordinates of DrawX and DrawY.

5.6 character_one

```
module  character_one ( input         Clk,                // 50 MHz clock
                            Reset,              // Active-high reset signal
                            frame_clk,          // The clock indicating a new frame (~60Hz)
                                      input             left, right, punch, kick, hurt, enter,
               input [9:0]   DrawX, DrawY,       // Current pixel coordinates
                                      input                     is_title,
               output logic  [1:0] is_char,             // Whether current pixel belongs to ball or background
                                      output logic  punching, kicking, lowblock, highblock,
                                      output logic  [9:0] position
             );
```

Purpose : This module is used to control character 1 (Player 1). This module acts as the interface for all animations as well as all movements for character 1. All modules controlling these aspects of the character are instantiated here.

5.7 character_two

```
module  character_two ( input         Clk,                // 50 MHz clock
                            Reset,              // Active-high reset signal
                            frame_clk,          // The clock indicating a new frame (~60Hz)
                                      input         left, right, punch, kick, hurt, enter,
               input [9:0]   DrawX, DrawY,       // Current pixel coordinates
                                      input                     is_title,
               output logic  [1:0] is_char,             // Whether current pixel belongs to ball or background
                                      output logic  punching, kicking,
                                      output logic  [9:0]  position
             );
```

Purpose : This module is used to control character 2 (Player 2). This module acts as the interface for all animations as well as all movements for character 2. All modules controlling these aspects of the character are instantiated here.

5.8 color_mapper

```
module  color_mapper ( input              [1:0] is_char2,           // Whether current pixel belongs to ball
                                                 input                                [1:0] is_char1,
                                                 input                                is_title,
                                                 input                                [2:0] is_back,
                                                     //   or background (computed in ball.sv)
                       input        [9:0] DrawX, DrawY,       // Current pixel coordinates
                       output logic [7:0] VGA_R, VGA_G, VGA_B // VGA RGB output
                     );
```

Purpose : This module is used to map colors to the components of the game. It decides which color to use to draw depending on the DrawX and DrawY values. Depending on which sprite, it chooses a color from the sprite sheet.

5.9 end_game

```
module end_game(
                input logic [3:0] score1, score2,
                output logic one_win, two_win
                );
```

Purpose : This module is used to check whether the game is over by counting the total points. This module helps in triggering the end of the game sequence.

5.10 hit_calculator

```
module hit_calculator(
                input logic [9:0] x1, x2,
                input logic punch1, punch2, kick1, kick2,
                input logic high1, high2, low1, low2,
                output logic hit1, hit2
                );
```

Purpose : This module is used to determine which of the two players is hit depending on the attack and positions. This outputs whether player 1 or player 2 has been hit, thus updating the points accordingly.

5.11 hpi_to_intf

```
// Interface between NIOS II and EZ-OTG chip
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
                  );
```

Purpose : This module acts as an interface between the NIOS II and the EZ-OTG chip, used to implement the USB Keycode input.

5.12 keycode_reader

```
module keycode_reader(
                                                    input logic [31:0] keycode,

                                                    output logic a_on, d_on, g_on, h_on,                    // Player 1
                                                    output logic l_on, r_on, one_on, two_on,        // Player 2
                                                    output logic enter_on

                                                    );
```

Purpose : This module is used to decode the keycode input by distinguishing between the different keys that are pressed. It checks for the ascii values of the inputs and assigns the high signal to the keys that are pressed.

5.13 score_counter

```
module score_counter(
                                                    input logic Clk,
                                                    input logic [3:0] inscore1,
                                                    input logic [3:0] inscore2,
                                                    input logic Reset,

                                                    input logic hit1,
                                                    input logic hit2,

                                                    output logic [3:0] outscore1,
                                                    output logic [3:0] outscore2,
                                                    output logic complete

                                                    );
```

Purpose : This module is used to keep track of the score of each of the players. It is updated every time a player gets hit (Adds a point to the opponent's score).

5.14 title_screen

```
module title_screen
(
        input Clk, Reset, frame_clk,
        input keypress,
        input [9:0] DrawX, DrawY,
        output logic is_char,
        output logic is_title
        );
```

Purpose : This module is self explanatory, it checks whether the title screen is to be displayed. If yes, the title screen is displayed.

5.15 title_text

```
module  title_text
(
                input [54239:0] read_address,
                input Clk,

                output logic data_Out
);
```

Purpose : This module is a helper of the title_screen module as it helps obtain the bytes for the title sprite stored on a text file.

5.16 walk1

```
module  walk1
(
               input [1535:0] read_address,
               input Clk,

               output logic [1:0] data_Out
);
```

Purpose : This module obtains the bytes for the first frame of the walking animation. The bytes are stored in a text file, and the text file contains numbers ranging from 0 to 3 based on the color of the pixel.

5.17 walk2

```
module  walk2
(
                input [1535:0] read_address,
                input Clk,

                output logic [1:0] data_Out
);
```

Purpose : This module obtains the bytes for the second frame of the walking animation. The bytes are stored in a text file, and the text file contains numbers ranging from 0 to 3 based on the color of the pixel.

5.18 block

```
module  block
(
                input [1024:0] read_address,
                input Clk,

                output logic[2:0] data_Out
);
```

Purpose : This module obtains the bytes for the block that the characters are standing on.

5.19 lowblock

```
module  lowblock
(
                input [1535:0] read_address,
                input Clk,

                output logic [1:0] data_Out
);
```

Purpose : This module obtains the bytes for the low blocking animation. The bytes are stored in a text file, and the text file contains numbers ranging from 0 to 3 based on the color of the pixel. We never used the block animation in the final version because we weren't able to implement the block functionality.

5.20 hit

```
module  hit
(
                input [1535:0] read_address,
                input Clk,

                output logic [1:0] data_Out
);
```

Purpose : This module implements the display of the characters getting hit. The bytes are stored in a text file that the module reads from to obtain the desired address based on the input.

5.21 lowkick

```
module  lowkick
(
                input [1535:0] read_address,
                input Clk,

                output logic [1:0] data_Out
);
```

Purpose : This module implements the display of the characters performing a kick. The bytes are stored in a text file that the module reads from to obtain the desired address based on the input.

5.22 punch

```
module  punch
(
                input [1535:0] read_address,
                input Clk,

                output logic [1:0] data_Out
);
```

Purpose : This module implements the display of the characters performing a punch. The bytes are stored in a text file that the module reads from to obtain the desired address based on the input.

5.23 rock

```
module  rock
(
                input [1024:0] read_address,
                input Clk,

                output logic[1:0] data_Out
);
```

Purpose : Module to display a rock sprite.

5.24 walk3

```
module  walk3
(
                 input [1535:0] read_address,
                 input Clk,

                 output logic [1:0] data_Out
);
```

Purpose : This module is used to implement the animation for transitioning between an attack and the character back to the walk animation. We initially thought it was a third frame of the walk animation which is why it was called walk3.

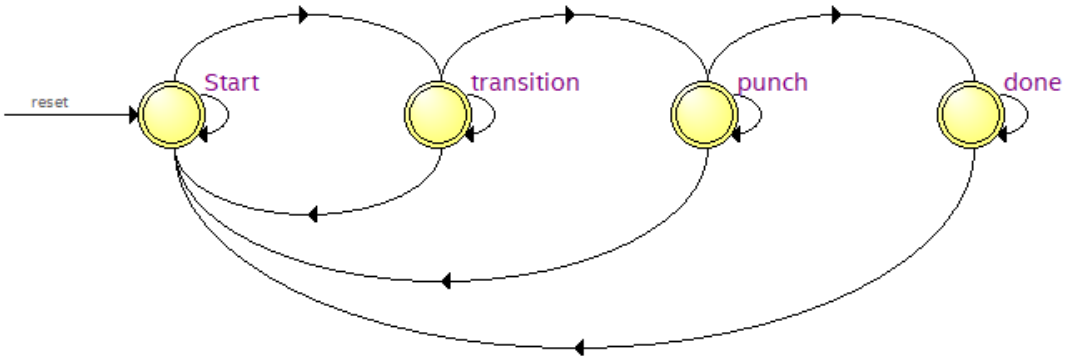
5.25 win_con

```
module win_con
(
        input Clk, Reset, frame_clk,
        input [3:0] score1, score2,
        input [9:0] DrawX, DrawY,
        output logic is_end
        );
```

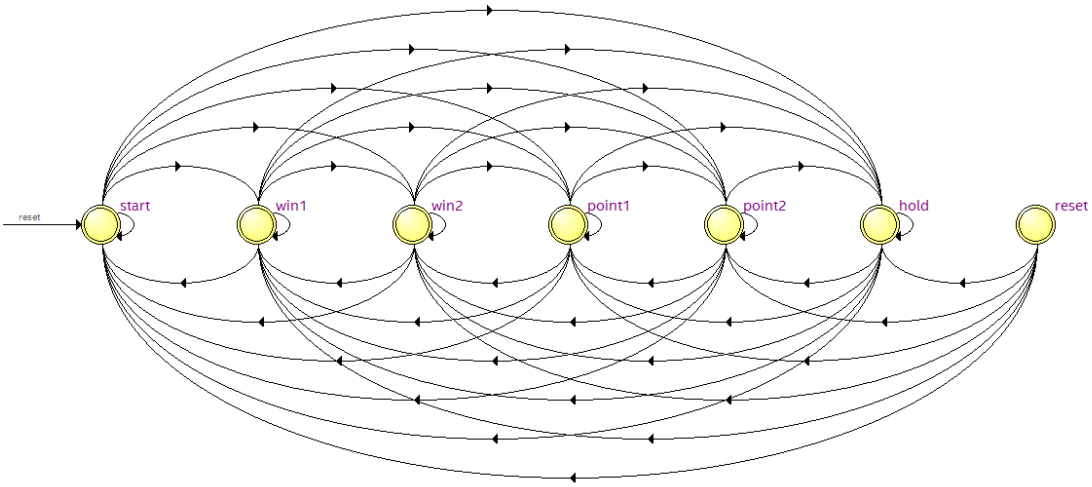
Purpose : This module is used to display the message when the game is won.

6 State Diagrams

6.1 Attack Animation :

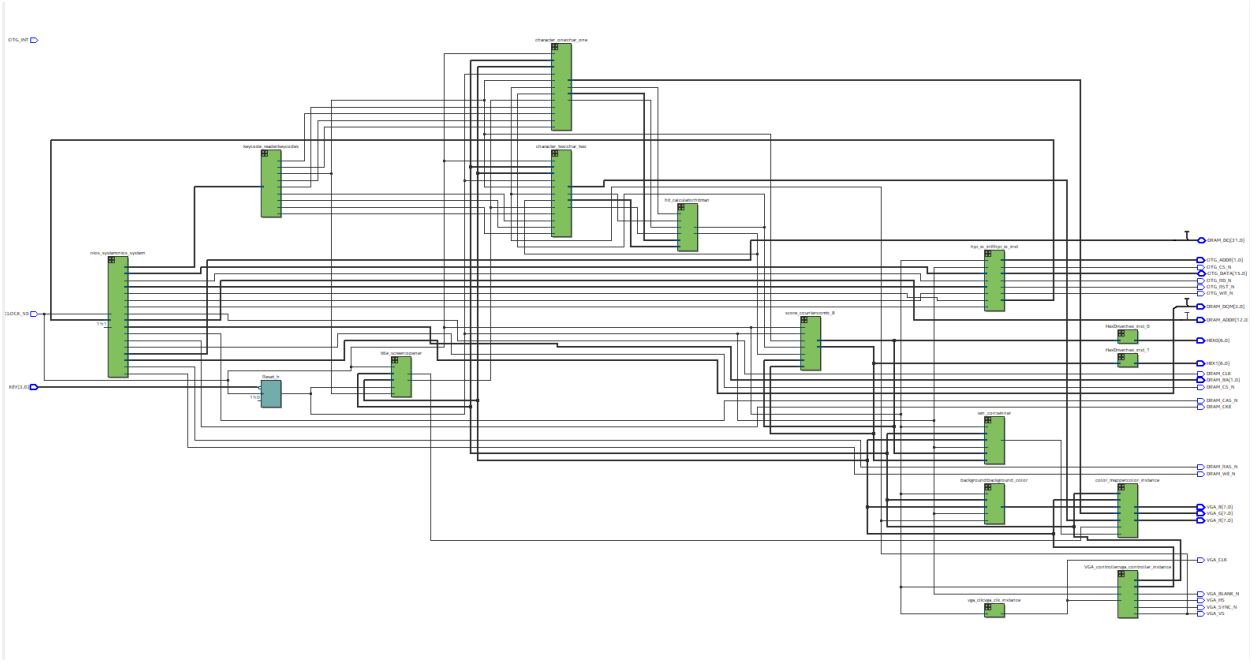


6.2 Score Counter :

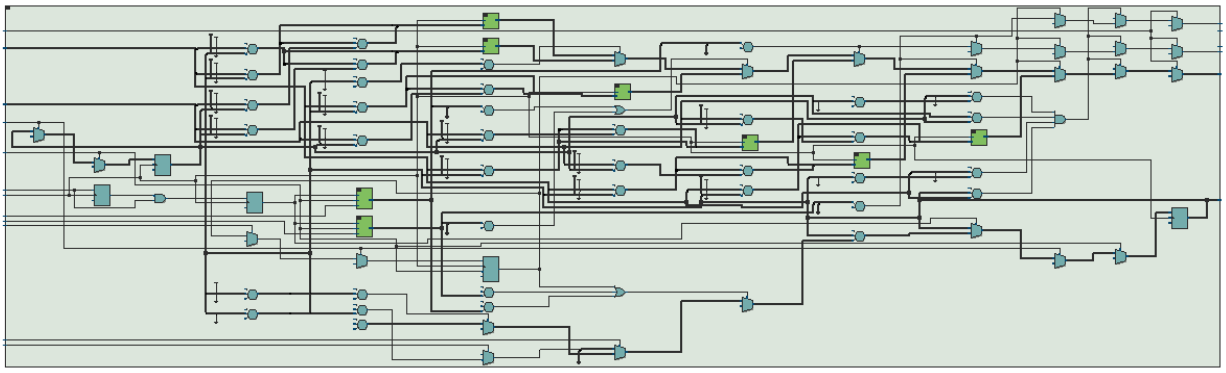


7 Block Diagrams

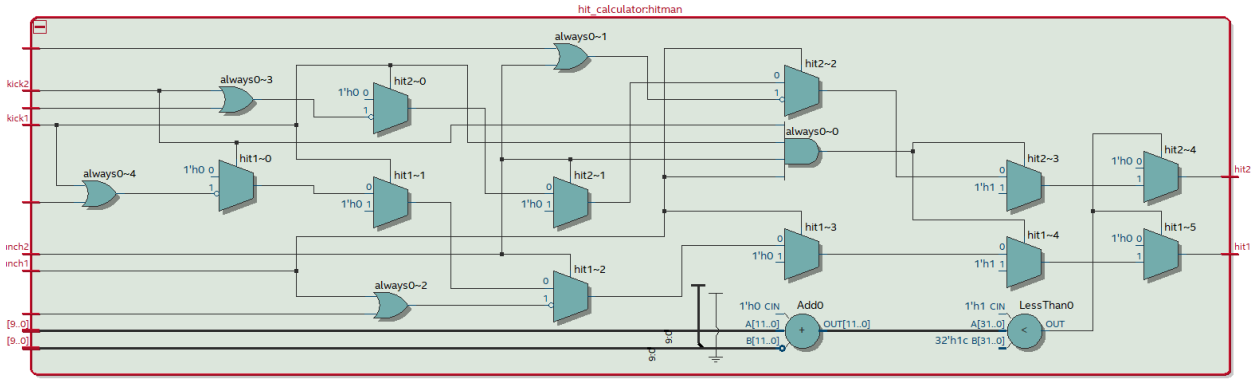
Karatechamp.sv (top-level)



Character_one.sv



hit_calculator.sv



8 Gameplay Images

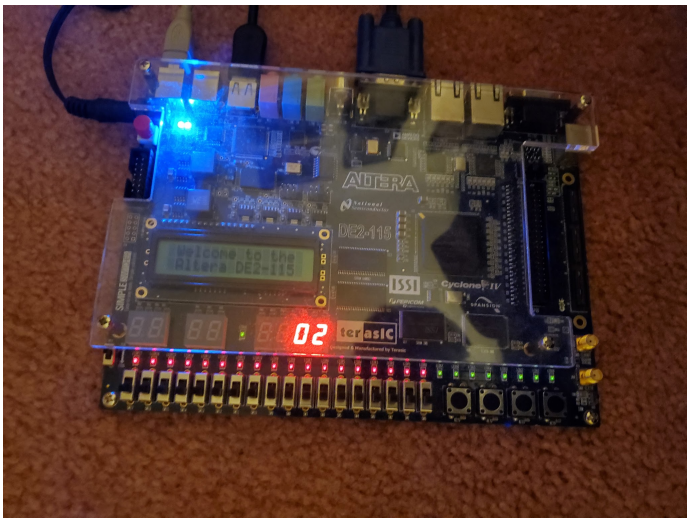
< Title Screen >

< Both characters on screen >

< Red player performing Low Kick >

< White player punching >

<Red player goes down after hit>



<Points displayed on Hex Drivers>

<Win screen when 10 points obtained>

9 Design Resources and Statistics

LUT	3373
DSP	10
BRAM	221,184
Flip-Flop	504
Frequency	75.4 MHz
Static Power	105.17 mW
Dynamic Power	.73 mW
Total Power	173.93 mW

10 Problems Faced

One of the first problems that we faced while trying to implement Karate Champ was being unable to read four keycodes at a time. We were however able to get it fixed by expanding the width of the keycode input and adjusting the Keyboard IO functionality in the C code using Eclipse.

Another issue we faced was being able to display sprites as earlier we were trying to upload the sprites sheet directly onto the board. We quickly realised that it didn't work and referred to the tutorial given on the course wiki to solve the problem.

We also came across an issue where some of the newer moves were not being registered due to low memory space. We were able to fix that by restructuring our moves and saving up space which was otherwise not in use.

We also had a glitch where our game altogether stopped working, reverting back to an older version of the code helped us recover from that. We were able to solve this problem by removing our hex displays. We are still not sure why having less hex displays allowed the characters to move again.

11 Conclusion

We believe that this final project allowed us to demonstrate our understanding of all the skills we acquired in using the FPGA board and other skills that we acquired during this process (such as using sprite sheets, etc.). Some of the skills we were able to demonstrate include the ability to create an interface between the USB devices and the FPGA (using a USB Keyboard), using Finite State Machines for implementing some of the features of this game and creating a VGA interface to help create a visual output from the FPGA board. Some of our understanding of the hardware versus software in terms of speed helped us make important decisions in regards to design. We chose to base a large part of our project on hardware as we knew that hardware was faster in comparison to software owing to Lab 9. Our use of the NIOS II was limited to about the same as Lab 8 as the only changes in the platform designer we made was to increase the width of the keycode reader. This project really allowed us to explore the FPGA board further by giving us the creative freedom that allowed us to have a different experience in comparison to other labs. We say different because we were allowed to create something more fun and tangible such as a game.

12 Acknowledgements

We would like to acknowledge all resources that we were pointed to by the course wiki as we found the documentation and code extremely helpful in implementing features that we lacked knowledge of. Additionally, we would also like to acknowledge KTTECH website for useful guides.