ECE385

Fall 2021
Final Project Report

Battle City

Kunle Li
3190112150
Zifei Han
3190110304

Section: L1 D225
TA: Lianjie Wang

Contents

1 Written description

1.1 Overview

The game we design is based on a famous game in the 1990s called *Battle City*. The overview of our game can be seen at Fig.1.

In this game, we have two human players and one computer player. Each human player is a tank with 5 HP bars, while the computer player (NPC), is a ghost with infinite HP bars. The player will lose his HP if he is shot by the other player, or he is touched by the ghost. If one player loses all his 5 HP, then the other player will win.

Two players cannot go through the walls, while the ghost can. The bullets of the players cannot go through the walls as well.

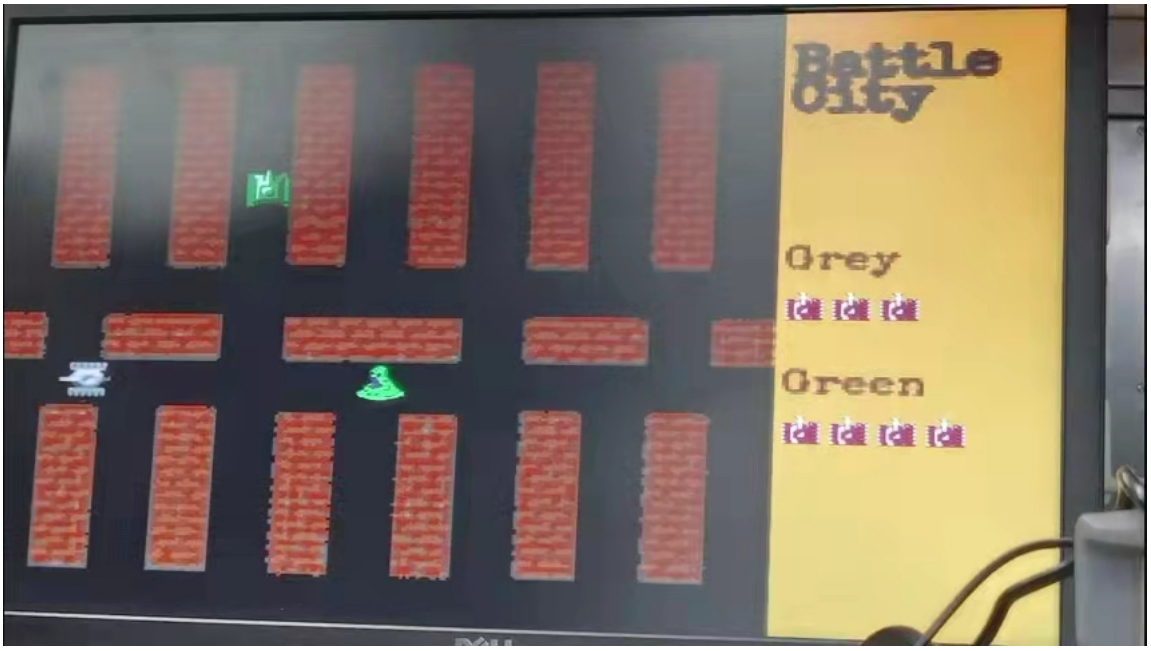

Figure 1: An overview of our game

When the game starts, a startmenu will be displayed (see Fig.2). Once ENTER is pressed, we will enter the game. Once the game is ended, either by pressing ESC or one of the players loses all his HP, the game will stop and an "game over" interface will be displayed (see Fig.3).

For each player, four direction keys are used to control its movement, and one key is used to shoot. The shooting direction must be consistent with the face direction (i.e., the moving direction), as the original game design.

The ghost will be initialized at the center of the screen, and it will chase the player whose Euclidean distance is closer to itself. The ghost can be shot as well. Once the ghost is shot, the ghost will be re-initialized at the center again.

1.2 Basic functions we realized

We mainly realized these basic functions:

- A colorful background with boundaries and walls.
- A startmenu and a end-game interface.
- 2D motion of the player.
- 2D motion of bullets.

Figure 2: The startmenu

- Correct effect of being hit.
- Correct calculation of HP.

1.3 Some special features

We mainly realized these functions to add difficulties:

- Dual players mode.
- Sound output.
- The display of HP bars. The HP bars will be displayed at the right side of the screen, represented by 5 purple tank icons.
- The addition of NPC (AI).

2 Module description

2.1 Sprites

The sprites resources are found online[1].

The original sprite resource (see Fig.4) contains much information that is meaningless to us, so we process the data to meet with our reuqirements. The tanks are processed so

[1]https://www.spriters-resource.com/nes/batcity/

Figure 3: gameover

that by calculating the offset of each image, we can correctly display the face direction of the tank. For instance, each tank in Fig.5 has size 32 * 32, so by adding an offset of 32 pixels, we can get a new face direction of this tank. The ghost is processed in the same way.

2.2 Background

The in-game interface is built upon the stage-01 of the original *Battle City*, with some modification (see Fig.6).

2.3 Draw

For drawing the background and sprites resources, we make use of the color mapper component, which is a portion of codes from Lab8 (call it CM for simplification). For support, we also build separate sv files for almost all elements to be displayed, including players, bullets, NPC and so forth (call them ESVs).

We firstly produce outputs for ESVs based on their inputs, which are mostly different parts of information about the element they are considering, coming from C code part. Based on the info, the program can determine when, how and where the image should be displayed on the screen. This kind of procedure takes care of all properties of an image, including its size in pixel unit, its position on the screen, its position on the original sprite (since we may only take a piece to display from the whole picture), and of course, its color. Note that we do not decide which color to be paint at this moment; we only record the original colors that the image displays in the form of txt file, without doing any color

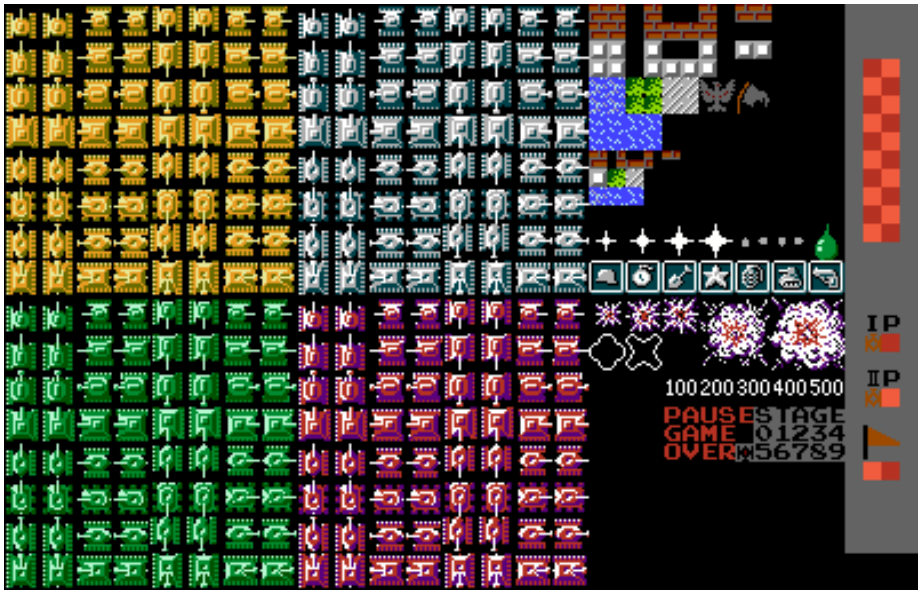

Figure 4: One of the original sprite resources

Figure 5: An example of our sprite

assignment to the screen. The outputs of ESVs are mostly data containing such drawing information of these elements.

Next, these data from ESVs will come to CM as input. CM contains two main parts: palettes and color assignments. Palette is just a set of RGB color codes, which contains 16 different colors that appears for an image. The color we can really see may be more than 16 kinds, but we need to shrink it to 16 and paint the missed color with the nearest one among the 16 chosen. Then, the color assignment variables will use the input data to match each pixel with a kind of color out of the 16 to represent the color to be drawn at the given coordinates DrawX and DrawY. Finally, these variables will be selected by the input data about whether we are displaying on element or not (only one kind of element, i.e. one kind of the ESVs will be selected to be true), and the final RGB is just the color of that. We will finally display the color of these final RGB values.

Figure 6: In-game interface

2.4 Audio

For this part, we applied and modified a module that comes from the official workspace of Intel. With that module, we are able to transform a mif audio file into readable digits in arrays and finally use that array for displaying audio.

Firstly in the provided .v file, we simply put in the parameters of our mif file. We have to firstly figure out the mif file in our working directory, and then set its width of address and data to reasonable values, based on the data that can be seen from the mif file. Finally we have to set number of words to be greater than the value indicated by the mif file and to be the integer multiple of width of words to make them suitable.

Then in the sound module file, we have three main instances: one for USB_Clock_PLL, one for I2C_Protocol, and the last one for the sound data module of the .v mentioned above. The I2C serves as a function block for contacting with the audio module on FPGA, while USB_Clock_PLL can produce a few different clock signals. For the last module, we will make use of a counter to count till all the data information of the mif file complied is read into the output, which will be displayed onto the FPGA board. In our case, this value is 93020, which can be directly read from the mif file. Beside of reading, we also implement writing in the same module, which also applies a counter for every single sound piece and then be displayed out.

2.5 Game State & Logic

There are three main game states for our game, and they are “start menu” state, “in game” state, and “game over” state. We have different backgrounds to be shown on the screen

for these states. The transition between these states is implemented in the C code part as a function. It mainly tells the program what to do in different current states if one key on the keyboard is pressed.

Our game logic is controlling the connection and interaction between four main elements: players, bullets and walls and the computer-controlled NPC. For players (there are two of them), they can shoot bullets on the direction they are moving for attacking each other or attacking the NPC, and once a player is wounded by bullets from its opposite, it will be set back to its generation point; they can move freely on the road but cannot move into the walls. For bullets, they can cause damage on HP of players (reduce by 1) when meeting them and disappear after that; they are also not able to pass the walls and will disappear when meeting them. For NPC, it can move in straight lines to chase the nearer player with it, and once it successfully catch up with a player, the player will be wounded and set back, just the same as being wounded by bullets; it is enabled to move regardless of all walls; it can be wounded by bullets from players, but it will simply be sent back to its generation point on the game map instead of experiencing an HP decrease. The game will end immediately after either player 1 or player 2 lost its last HP.

2.6 NPC

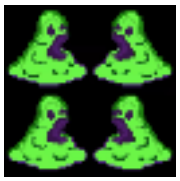

Figure 7: The ghost

The NPC (computer player) is built as follows. First, the ghost sprite (see Fig.7) is processed in the same way as the tank to ensure the display of the correct face direction. Then, the ghost will be initialized at the center of the screen (moved up a little bit to avoid being stuck).The ghost will calculate the Euclidean distance between itself and each player, and it will try to minimize the smaller distance, i.e., chasing the player that is closer to it.

The minimization is realized by exhaustion. That is, the ghost at each time step, will calculate the distance between itself and the player if it moves up, down, left and right separately. It will pick the direction which gives it the smallest result.

In order to increase difficulty, the ghost's HP will be recharge at every time step. Thus, the ghost will not disappear until the game stops.

3 Design procedure

3.1 Software

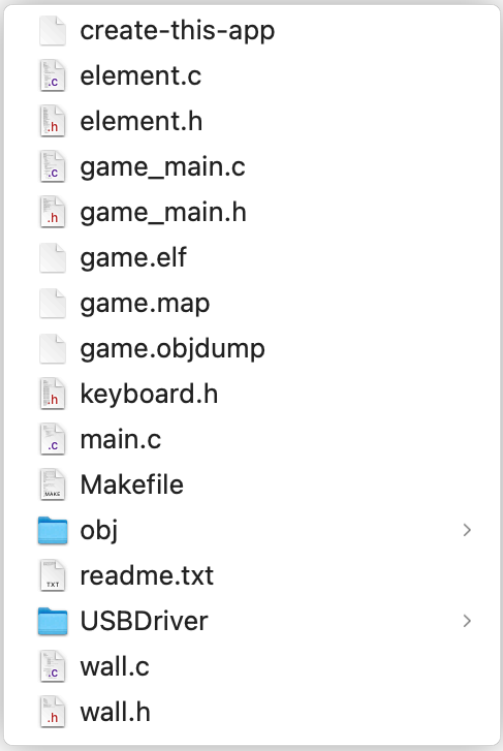

Figure 8: The structure of the C files

The software part is mainly done in C. The structure of the c files can be seen at Fig.8.

element The related files define the behaviors of the elements that will appear in the game, such as players (including NPC), bullets, etc. All properties of each element are defined in the .h file, including player start position, player speed, bullet speed, etc. The initialization of the players and the bullets are also defined within these files, as well as the animation effects.

game_main The related files define the logic of the game. Namely, when the users enter the game, many functions need to be operated real time in order to guarantee the smooth realization of the game. Specifically, the software needs to check that which state the game

is currently in; detect the pressing of keys by the users; check that whether either of the player has lost all his HP, etc.

main main.c is the top-level c file. It contains a big while loop which continuously check the state of the game and lead to corresponding operations based on the current state.

wall The related files define the boundaries of the game field and the behaviors of the players and the ghost when they are on the boundaries. We have mainly two types of boundaries in our game, the outer boundary of the screen and the walls. The logic we realize is that once any of the four corners of the player or the ghost enters the forbidden area, it will be moved out immediately to its original position.

3.2 Hardware

Beside of the ESVs and CM that we mentioned in previous sections, there are also other sv files that we apply.

- VGA_controller: This module serves as the VGA controller by getting the sync pulses and setting the parameters for each pixel. In this module, different pixels on the screen will be set to different colors to indicate the position of all the images.
- hpi_io_intf: This module works as the connection part of data between NIOS II and EZ-OTG chip. In this module, based on controlling signals, different output signals will be assigned to different values, and these outputs will control the behavior of EZ-OTG chip.
- avalon_game_interface: This module creates a register file of 64 registers, which will accept data from the input WRITE_DATA. It serves as the interface between software and hardware. Its output contains all the data to be shown to sv part and contains 2048 bit in our case, which comes from the C code and stored as its output EXPORT_DATA.
- gamefile_reader: This part is essential for building game related connections between C code and sv modules. Its input is the 2048-bit output of avalon_game_interface. In this module, the large input array will be divided into several small arrays, each with bit number of 32 and contains different sort of info of players, bullets, NPC and so forth.

4 Graphs

4.1 Block diagram

The block diagram of our game is shown in Fig.9.

4.2 RTL Viewer

The RTL of the whole toplevel module is shown on the next page.

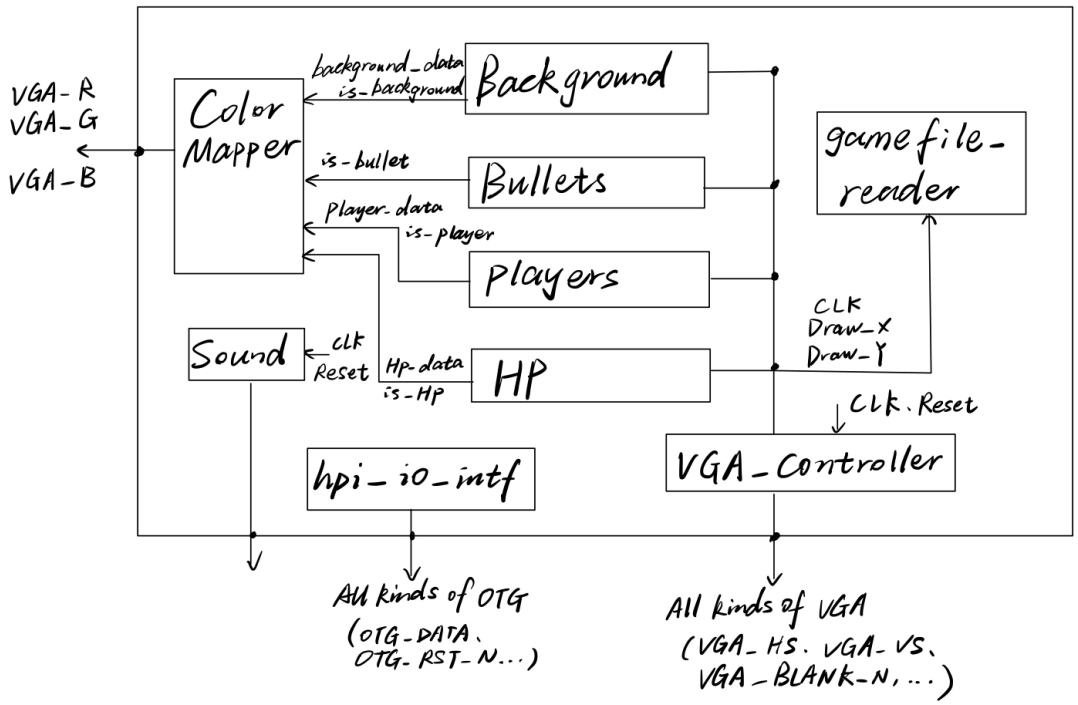

Figure 9: Block Diagram of the Main Logic

4.3 Modules

The RTL view of several modules that are new compared with previous labs are shown in Fig.10, Fig.11, Fig.12, Fig.13, Fig.14, Fig.15 and Fig.16.

4.4 State Machine

The diagram of state machine is shown in Fig.17.

5 Resource Table

The design statistic table is shown in Fig.18.

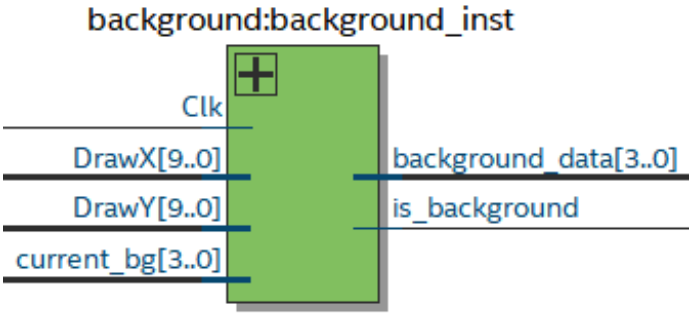

Figure 10: RTL of background

6 Conclusion

In this project, we manage to build a modified version of *Battle City* based on the original game design. We successfully realize the basic operations of the game, including the motion of the players, the correct effect and animation of shooting, etc. The game logic we designed functions smoothly as well.

In order to increase difficulty, we add some special features to our design. First, we enable dual player mode, which allows two players simultaneously operating their characters in the game. We also enable sound output effect and provide the players with background music. Additionally, we add an NPC to the game that will automatically "kill" human players in order to make the game more difficult to win.

A large portion of the game is designed using C code, including the game logic, the effect of the walls, the behavior of the players and the NPC, etc. We also use SystemVerilog to realize the functions of our hardware, including VGA output, USB I/O, etc.

The game we design literally has no bugs as we had already done the corner test and the monkey test before demo. During demo, our game also functioned smoothly.

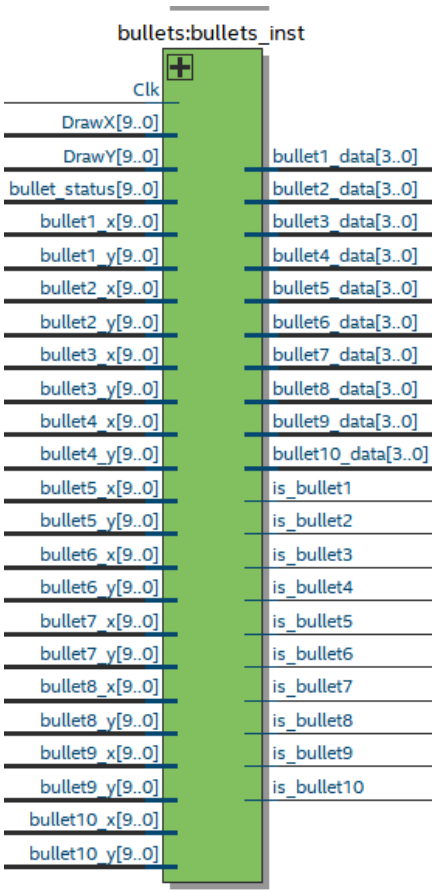

Figure 11: RTL of bullets

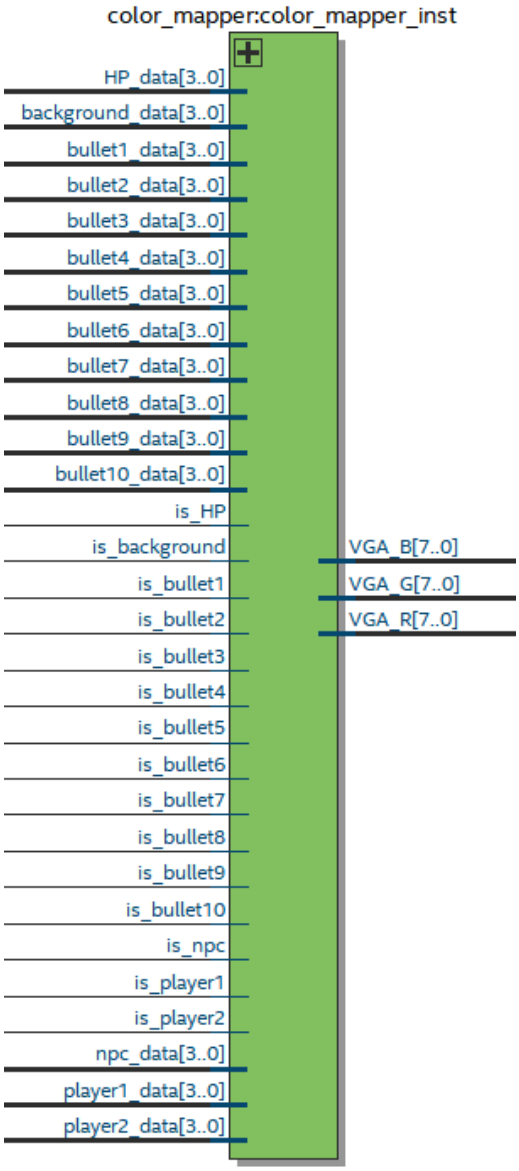

Figure 12: RTL of color mapper

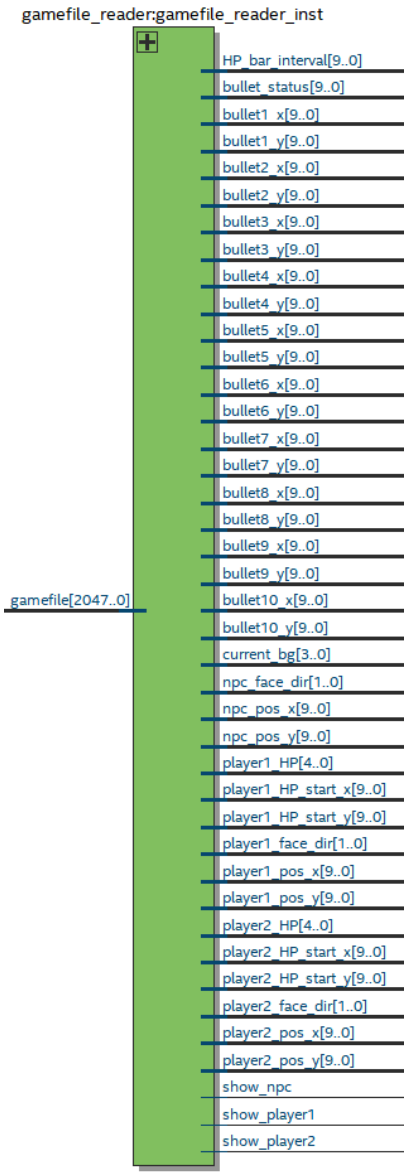

Figure 13: RTL of gamefile reader

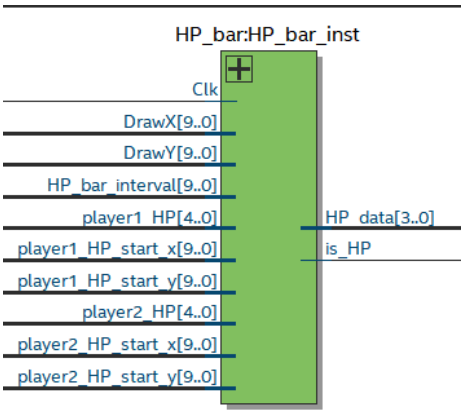

Figure 14: RTL of HP

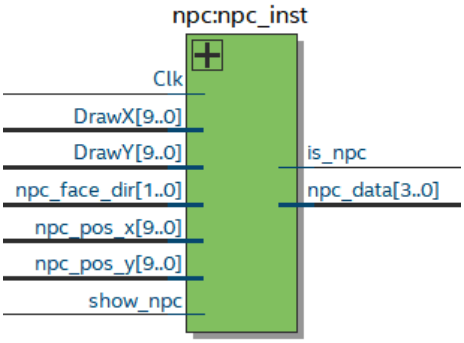

Figure 15: RTL of NPC

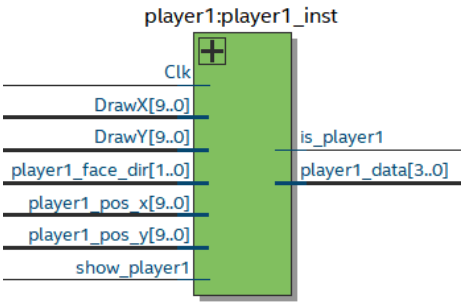

Figure 16: RTL of Player 1 (Player 2 the same)

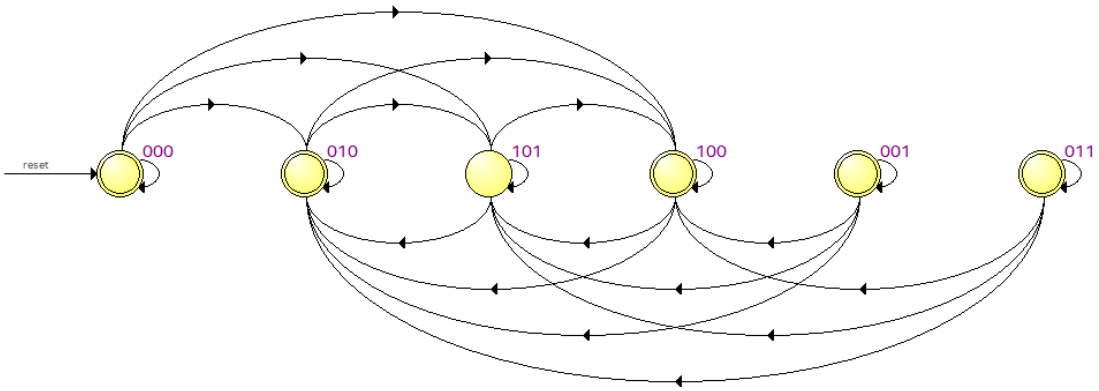

Figure 17: State Machine Diagram

	Final Design
LUT	6993
DSP	0
BRAM	2,824,320
Flip-Flop	4387
Frequency	107.74 MHz
Static Power	108.83 mW
Dynamic Power	0.77 mW
Total Power	190.96 mW

Figure 18: Design Statistics Table