Portland State University

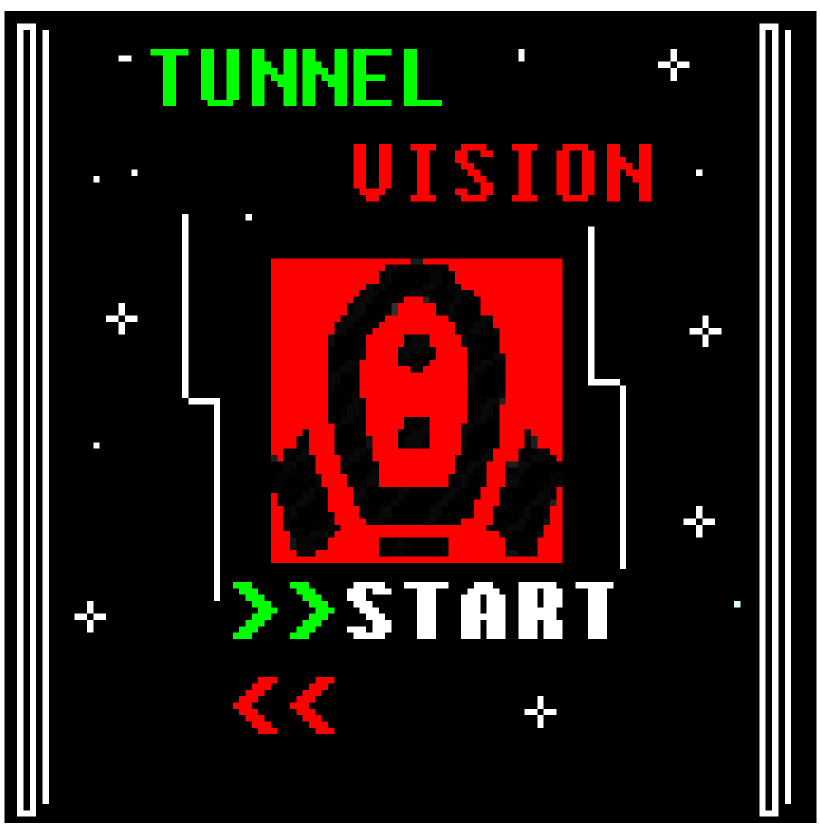
SoC Design With FPGAs

ECE540

Tunnel Vision

Erik Rhodes Bhavana Dhulipala Rohan Deshpande Nikhil Patil

March 20, 2014



Contents

1 Introduction

Tunnel Vision is a racing game that can be played on the **Xilinx Nexys3 FPGA** board and be displayed on a VGA monitor.

1.1 Gameplay

The player tries to avoid hitting the walls as it travels down the tunnel by moving the vehicle left and right. The space in between the walls steadily decreases until the player hits a wall or obstacle. The score is based on the amount of time the vehicle remains "alive", and is displayed on the 7-segment display.

1.2 Controls

The player can move his vehicle by using the left and right pushbuttons on the Nexys3. When the game is over, hitting the middle button will reset the course. The top button starts the game and the bottom pushbutton pauses it. Different icons and speeds can be selected by toggling the switches on the board.

1.3 Features

Tunnel Vision features both starting and ending screens. The courses are generated randomly through a pseudo-random number generator. Additionally, the LEDs are lit with certain patterns depending on the action the player is taking. If the player selects the harder difficulty, the score is incremented at a faster rate and with a multiplier, awarding them a higher score for the same distance travelled.

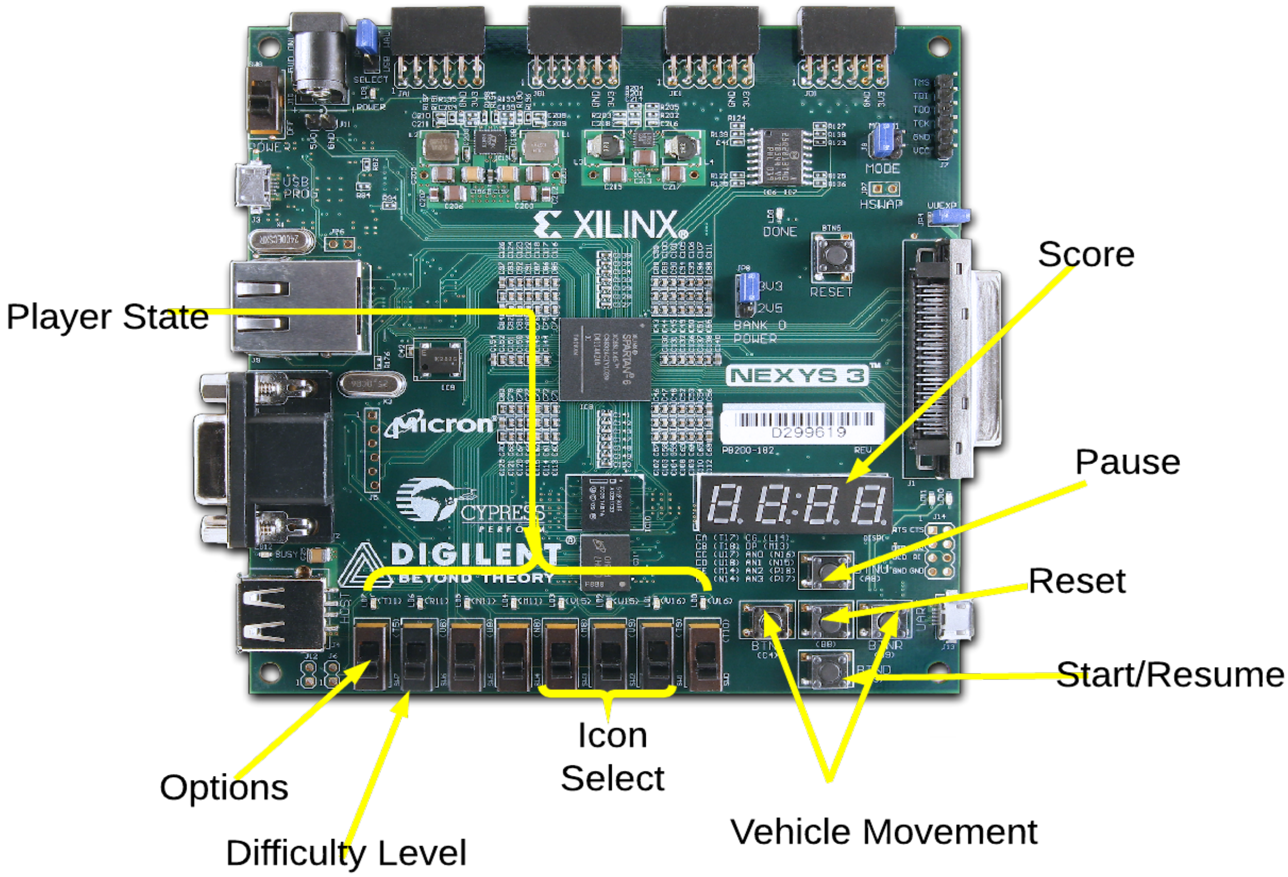


Figure 1: Player Controls

2 Implementation

The information controlled in the **PicoBlaze** core is sent to the `Video_controller` module via the `game_interface` module. This information is constructed in an 8-bit register called `game_info`. The allocation of bits is seen in Figure 3. Collisions, obstacles, and icon changes were controlled in hardware. The video controller received the inputs of game_info register and the `LFSR` module, shifted the wall appropriately, and used the collision logic to stop the game and display the **Game Over** screen.

2.1 Software

In order to ensure the correct flow of game play, the push button inputs were checked in the order seen in Figure 4. Each button check performs approximately the same function. The pushbutton input is checked, all other bits not pertaining to the specific button are masked off, and the button value is checked. If it has not been pressed, the next button check is called. If it has been pressed, the state is updated accordingly and the LEDs output the desired pattern.

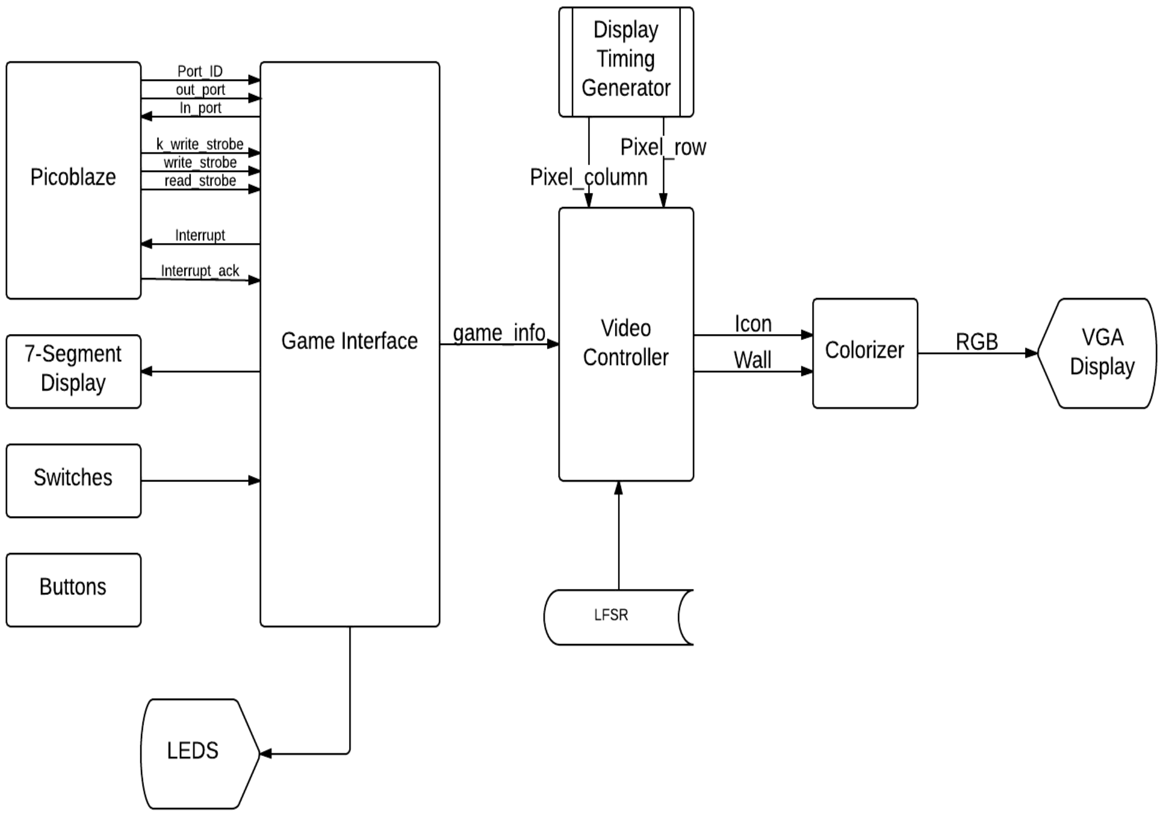


Figure 2: Game Play Block Diagram

Listing 1: Function checking if the top button has been pushed

```
FETCH   s2,   SP_BTN          ;load saved debounced button
AND     s2,   MSK_BTN_UP      ;mask with up button
COMPARE s2,   MSK_BTN_UP
JUMP    NZ,   chk_no_btn      ;go to next phase if no up button
LOAD    s3,   PAUSED_STATE    ;else pause and send out that register
STORE   s3,   STATE
FETCH   s3,   SP_GAME_INFO
LOAD    s4,   GAME_PAUSED           ;update game_info paused
AND     s3,   s4
STORE   s3,   SP_GAME_INFO
LOAD    s5,   FF              ;output LED pattern
OUTPUT  s5,   PA_LEDS
RETURN
```

If the game is active and another button than the top one has been pressed, the `active` function is called. This updates the state, score, and `game_info` output. In addition, when the left or right button has been pressed, the location of the vehicle is appropriately.

Listing 2: Update the state game info

```
CALL    update_score
LOAD    s3,   ACTIVE_STATE
```

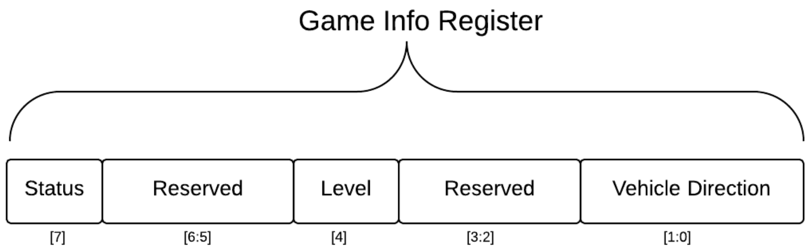


Figure 3: Allocation of bits in game_info register

```
STORE  s3 ,    STATE
FETCH  s3 ,    SP_GAME_INFO
LOAD   s4 ,    GAME_ACTIVE
OR     s3 ,    s4
STORE  s3 ,    SP_GAME_INFO
RETURN
```

In order to show the score over 4 digits, the `update_score` function increments the least significant digit until 10 is reached. At this point, the next significant digit counter is called and the previous value cleared. This algorithm continues throughout all four digits. A separate function, `display_score`, was created to actually load the stored score values to be displayed. This structure was chosen to allow for the score to still update even if the 7-segment display is used for another output. Future modifications to the game may make use of this function.

The difficulty level of **Tunnel Vision** can be changed by toggling switch 4 (Figure 1). When the user has switched levels, the speed of the game increases. This information is also sent in the game_info register (Figure 3). The speed control is doubled, but the score increases at a 5:4 proportion. This scaling is accomplished by modifying the counter in the `game_interface` module that controls the speed of the interrupts. The speed is modified in the hardware portion.

3 Video Controller Implementation

The video controller module managed the icon switching and generation, the width and location of the walls, and the collision detection.

3.1 Walls

The tunnel walls consist two black lines separated by a predefined width. The walls directions are generated randomly using a linear feedback shift register. The video controller takes 2 bits from the LFSR and moves the walls left, right, or keeps them at the same position. Since there are 4 possible numbers, the last possibility defaulted to keeping the walls at the same position.

The tunnel width, which is a function of time, decreases continuously as the user is playing the game. It is controlled by a flag that is set whenever the counter reaches a specific number. The wall speed is also monitored by a counter, which decides when to alter the current pixel and row.

Edge detection was performed in order to ensure the walls did not move out of bounds.

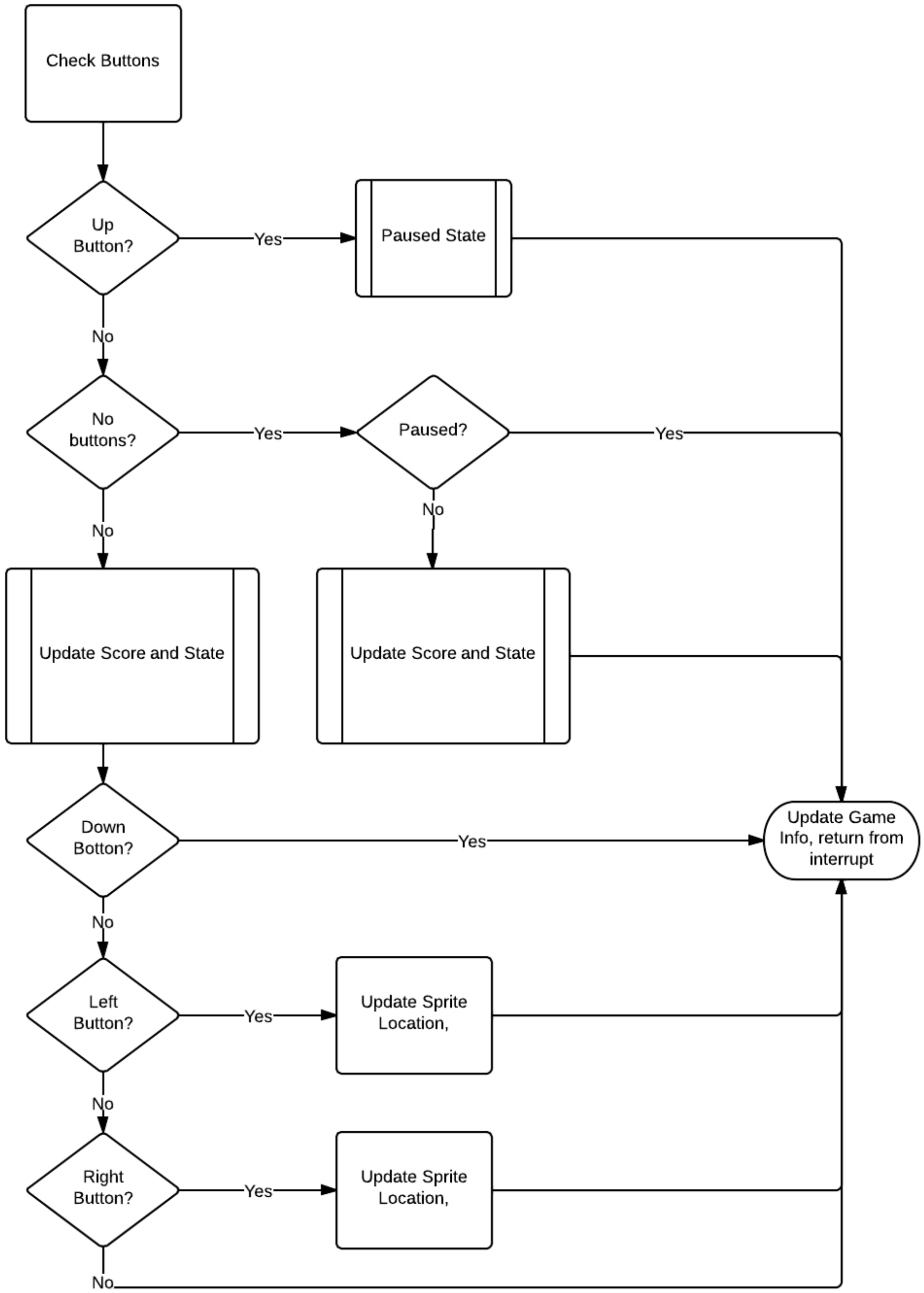


Figure 4: Flowchart of Picoblaze Logic

3.2 Collision Detection

We are detecting collisions by comparing the x and y coordinates of the wall and the x coordinate of the icon (right?) This is because the icon never moves up or down. If they are equal then that is a collision and a register called collision detect is made high. At every clock edge collision detect is checked, when one it concludes the game. At this time the icon is not displayed and a "Game Over" screen is displayed.

Wall	Icon	Color	Purpose
00	00	White	Background
01	00	Black	Walls
10	00	Red	Obstacles
11	00	Gray	Background
X	01	Maroon	Unused
X	10	Cyan	Start/End Screen
X	11	Magenta	Start/End Screen

Table 1: Color Schemes

3.3 Icons

There are currently four icons which can be selected by different switches on the board. The icons were created by directly coding the images into bits instead of reading the image from a `coe` file.

3.4 Colorizer

The `colorizer` module displays the icon and tunnel walls pixel by pixel. Table 1 summarizes the color scheme used for wall and icon.

4 Conclusion

Creating **Tunnel Vision** was an enjoyable experience (for the most part). While we did face some roadblocks, we were pleased with the overall result. There are many different paths we can take in the future to add new features to this game.

4.1 Challenges

- **ROM for map that dynamically changes**. We attempted to add memory in order to change the collision detection and dynamically switch wall coordinates, but this was not able to be implemented.
- **Integration between hard-coded images and script generated images**. We had trouble getting the different icons and splash screens created with Perl script to work with our current icon implementation. Ultimately, these graphics are only used in the report and final presentation.

	Erik Rhodes	Bhavana Dhulipala	Rohan Deshpande	Nikhil Patil
Game Logic	✓			
Video Controller	✓	✓	✓	
Icons and Walls		✓		
Graphics		✓	✓	✓
Random Number Generation			✓	
Documentation and Source Control	✓			

Table 2: Division of Tasks

4.2 Future Work

- **ROM Map:** The current implementation of wall generation does not use any memory. If this functionality was converted to be stored in a ROM, different wall widths could be displayed in one frame, and collision detection could be done based off of the coordinates themselves.
- **Improved Graphics:** Creating more icons and backgrounds would improve the game's aesthetic appeal.
- **Powerups and Bonuses:** Certain objects could give the player an extra life or a slower game speed.
- **Multiplayer Mode:** Two different tunnels could be generated, allowing multiple players to compete against each other.