The Final Report for FPGA Cat Invaders

Adam Jablonski (asj2172)
Yilei Meng (ym2974)
Fernandos Magee Jr. (fm2756)
Yuxiang Xia (yx2821)
Zhili Yan(zy2593)

Spring 2024

Table of Contents:
1 Introduction 2
2 Block Diagram 3
3 Resource Budget 3
4 Algorithms 9
5 The Hardware/Software Interface 11
6 Audio Interface 17
7 Video Display 20
8 Memory 24
9 Game Controller 27
10 Conclusion 29
11 References 31
Appendix A: Code 32
Appendix B: Platform Design 141
Appendix C: The Screen Layout 143
1 Introduction

This final report will describe the hardware and software details for implementing FPGA Cat Invaders, a game based on the classic Space Invader arcade game [1] with appearance and gaming control variations.

The FPGA Cat Invader game storyline goes as follows, the cats have invaded and our hero the poodle is defending Earth. The poodle launches a femur bone projectile and barks at the incoming cat enemies, and while this happens, the cats are launching mice back at the poodle. In the event the femur bone collides with an enemy the enemy lets out a meow and disappears. If the poodle collides with the mouse, the poodle loses a life. The cats are lined up at the top of the screen in a matrix of 9x5, and their movement pattern shifts right until at the wall, shifts down one slot, shifts left until at the wall, shifts down, shifts right. The level is completed when all the cats on screen have been eliminated and the game is over if the poodle loses three lives. The game's difficulty is controlled by the speed of cat movement and the rate of mice being launched at the poodle, as the levels progress the cats are faster and launch mice more frequently.

The game is played through a combination of hardware, software, and peripherals. The player uses an iNNEXT controller (Figure 10) to control the movement of the poodle and launch the femur bones. The game is displayed through a VGA monitor and the audio is played through mono speakers. The game algorithm runs in software, monitoring inputs from the iNNEXT controller and communicating with the FPGA hardware registers through the Avalon Bus [7]. The FPGA hardware is responsible for outputting the video to the VGA monitor and the audio to the mono speakers.

This final report is divided into sections that cover the details of the FPGA Cat Invaders system architecture. See the Table of Contents on page 1 for further information.
2 Block Diagram

The architecture of the FPGA Cat Invaders game is organized into three main categories, peripherals, software, and hardware. Figure 1 is the high-level block diagram of the overall hardware/peripheral/software system. The iNNEXT controller will be communicating both input and output commands/responses using USB to the software system. A software control loop will be running that will make all algorithm decisions for the FPGA Cat Invader game. This software will communicate through the kernel, over the Avalon bus, to both the VGA hardware algorithm and Audio hardware algorithm. The hardware process through internal algorithms and output to the VGA monitor peripheral and speaker peripheral.

Figure 1: Block Diagram describing the software and hardware at a high level.

3 Resource Budget

The DE1-SoC has 256 KB of BRAM organized as 64K x 32 bits on the DE1_SoC FPGA [3]. See Table 1 and 2 below for the sprites and audio being included in this project and the memory required for each. The sprites and audio will be stored in the FPGA read-only memory (ROM) in mif format [8]. In our ROM, image pixels are stored as 8-bit binary numbers. The highest three bits are the R data, the middle three bits are the G data, and the lowest two bits are the B data. When used as 24-bit RGB input for VGA, the R/G/B data are respectively shifted left by 5,
5, and 6 bits. Using this method, we are able to significantly reduce the amount of memory occupied by the image. All sprites and audio will fit ROM/RAM based on these calculations.

Table 1: Table organizing memory required per sprite and total amount of memory required.

<table>
<thead>
<tr>
<th>Name</th>
<th>Sprite</th>
<th>Size (bits)</th>
<th>Color (bits)</th>
<th># of Frames</th>
<th>Total Size (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Poodle</td>
<td><img src="image1.png" alt="Image" /></td>
<td>32 x 32</td>
<td>8</td>
<td>10</td>
<td>81920</td>
</tr>
<tr>
<td>Cat 1</td>
<td><img src="image2.png" alt="Image" /></td>
<td>32 x 32</td>
<td>8</td>
<td>2</td>
<td>16384</td>
</tr>
<tr>
<td>Cat 2</td>
<td><img src="image3.png" alt="Image" /></td>
<td>32 x 32</td>
<td>8</td>
<td>2</td>
<td>16384</td>
</tr>
<tr>
<td>Cat 3</td>
<td><img src="image4.png" alt="Image" /></td>
<td>32 x 32</td>
<td>8</td>
<td>2</td>
<td>16384</td>
</tr>
<tr>
<td>Cat 4</td>
<td><img src="image5.png" alt="Image" /></td>
<td>32 x 32</td>
<td>8</td>
<td>2</td>
<td>16384</td>
</tr>
<tr>
<td>Object</td>
<td>Image</td>
<td>Width</td>
<td>Height</td>
<td>Layer</td>
<td>Value</td>
</tr>
<tr>
<td>-----------------------</td>
<td>-------</td>
<td>-------</td>
<td>--------</td>
<td>-------</td>
<td>-------</td>
</tr>
<tr>
<td>Mystery</td>
<td>![Mystery Image]</td>
<td>48 x 48</td>
<td>8</td>
<td>2</td>
<td>36864</td>
</tr>
<tr>
<td>Projectile (femur bone)</td>
<td>![Projectile Femur Image]</td>
<td>8 x 24</td>
<td>8</td>
<td>1</td>
<td>1536</td>
</tr>
<tr>
<td>Projectile (mouse)</td>
<td>![Projectile Mouse Image]</td>
<td>8 x 16</td>
<td>8</td>
<td>1</td>
<td>1024</td>
</tr>
<tr>
<td>Barriers - solid</td>
<td>![Solid Barriers Image]</td>
<td>16 x 16</td>
<td>8</td>
<td>1</td>
<td>2048</td>
</tr>
<tr>
<td>Barrier - damaged 1</td>
<td>![Damaged Barrier Image]</td>
<td>16 x 16</td>
<td>8</td>
<td>1</td>
<td>2048</td>
</tr>
<tr>
<td>Image</td>
<td>Description</td>
<td>Width</td>
<td>Height</td>
<td>Expansion</td>
<td>Value</td>
</tr>
<tr>
<td>-------</td>
<td>------------------------------</td>
<td>-------</td>
<td>--------</td>
<td>-----------</td>
<td>--------</td>
</tr>
<tr>
<td><img src="image1" alt="Barrier - damaged 2" /></td>
<td>Barrier - damaged 2</td>
<td>16 x 16</td>
<td></td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td><img src="image2" alt="Barrier - damaged 3" /></td>
<td>Barrier - damaged 3</td>
<td>16 x 16</td>
<td></td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td><img src="image3" alt="Lives" /></td>
<td>Lives</td>
<td>32 x 32</td>
<td></td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td><img src="image4" alt="Explosion" /></td>
<td>Explosion</td>
<td>32 x 32</td>
<td></td>
<td>8</td>
<td>2</td>
</tr>
<tr>
<td><img src="image5" alt="Level (text)" /></td>
<td>Level (text)</td>
<td>96 x 32</td>
<td></td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td><img src="image6" alt="Score (text)" /></td>
<td>Score (text)</td>
<td>96 x 32</td>
<td></td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td><img src="image7" alt="Start (text)" /></td>
<td>Start (text)</td>
<td>144 x 48</td>
<td></td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td><img src="image8" alt="Game Over (text)" /></td>
<td>Game Over (text)</td>
<td>196 x 28</td>
<td></td>
<td>8</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td></td>
</tr>
<tr>
<td>0 (text)</td>
<td><img src="image1.png" alt="Image" /></td>
<td>32 x 32</td>
<td>8</td>
<td>1</td>
<td>8192</td>
</tr>
<tr>
<td>1 (text)</td>
<td><img src="image2.png" alt="Image" /></td>
<td>32 x 32</td>
<td>8</td>
<td>1</td>
<td>8192</td>
</tr>
<tr>
<td>2 (text)</td>
<td><img src="image3.png" alt="Image" /></td>
<td>32 x 32</td>
<td>8</td>
<td>1</td>
<td>8192</td>
</tr>
<tr>
<td>3 (text)</td>
<td><img src="image4.png" alt="Image" /></td>
<td>32 x 32</td>
<td>8</td>
<td>1</td>
<td>8192</td>
</tr>
<tr>
<td>4 (text)</td>
<td><img src="image5.png" alt="Image" /></td>
<td>32 x 32</td>
<td>8</td>
<td>1</td>
<td>8192</td>
</tr>
<tr>
<td>5 (text)</td>
<td><img src="image6.png" alt="Image" /></td>
<td>32 x 32</td>
<td>8</td>
<td>1</td>
<td>8192</td>
</tr>
<tr>
<td>Name</td>
<td>Size (bits)</td>
<td>Total Size (bits)</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
<td>------------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bark</td>
<td>4097 x 16</td>
<td>65552</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Meow</td>
<td>6178 x 16</td>
<td>98848</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Background music</td>
<td>35644 x 16</td>
<td>832224</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Total Size (bits)</strong> (KiloBytes)</td>
<td><strong>734704 (89.7 KB)</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Table 3: Table organizing memory required per RAM and the total amount of memory required.

<table>
<thead>
<tr>
<th>Name</th>
<th>Size (bits)</th>
<th>Total Size (bits)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barrier</td>
<td>24 x 8</td>
<td>192</td>
</tr>
<tr>
<td>Cat</td>
<td>55 x 8</td>
<td>440</td>
</tr>
<tr>
<td><strong>Total Size (bits)</strong></td>
<td><strong>79 B</strong></td>
<td></td>
</tr>
</tbody>
</table>

4 Algorithms

Function Descriptions:

init_cat_matrix() - Initializes the cat matrix with random cats.
controller_init() - Finds a connected controller and initializes it in software.
init_driver() - Opens the file descriptor for the Cat Invaders interface.
reset_barrier_matrix() - Resets each individual barrier to its solid, undamaged state.
reset_objects() - Resets all positions and visibilities to their default state. In addition, resets the timer used for the mystery and logic used for projectiles.

play_background() - Begins the playing of the background music.

check_all_eliminated() - Checks if all of the cats have been eliminated by the player. If they have, the player advances a level, retaining their lives.
bone_collision() - Checks for collisions between the bone and barriers, cats, and the mystery using Axis Aligned Bounding Boxes (AABB). If a collision is detected, then the bone is despawned and interacts with the collided object. Colliding with or mystery would eliminate them, and collisions with barriers updates their state.
mouse_collision() - Checks for collisions between the mouse and the barriers and dog using Axis Aligned Bounding Boxes (AABB). If a collision is detected with the player, then the player loses a life. If the mouse collides with the barrier, then the barrier updates its state. The mouse despawns on impact.
moves() - Moves the cat array. The cat initially moves right until it hits the edge of the screen. When the cats hit the edge of the screen, they move down and change direction.
animate_cats() - Changes the cat frames on a fixed interval to simulate animation.
moves_player() - Moves the player in response to the joypad input. The player would move left or right corresponding to the button held on the joypad. If the player reaches the edge, then the plate will not move past it.
animate_player() - Changes the player’s animation frames according to their state. If the player is walking, then the player cycles through the walk animations. Otherwise, if they are not moving, then the player idles. The player has animations in both directions.
fire_cat_projectile() - Spawns a mouse at a random cat left in the matrix. May spawn up to 3 mice at once until a mouse must despawn before another is fired.
move_cat Projectile() - Moves the mouse downward, maintaining its horizontal position. If the projectile moves past the dog without colliding, it is despawned.

update_dog Projectile() - Spawns a bone whenever the player presses ‘A’ and there is not currently a bone on screen. If a bone has been fired, then it is moved upwards periodically until it either collides or reaches the top of the screen, in which case it is despawned.

set_score() - Updates the score, a representation of how many cats and mysteries have been eliminated.

reached_bottom() - Determines if the cats have reached the bottom of the screen. If there is at least one cat that makes it to the bottom of the screen, then the player loses all of their lives.

check_lives() - Checks if the player lost all of their lives. If the player has, then a ‘Game Over’ is displayed. The player may play again starting from Level 1 by pressing ‘Start.’

set_status() - Sets the status of the game, whether there is a Game Over or the player is still alive.

spawn_mystery() - Spawns the mystery according to a random timer. The timer may range from 20 seconds to 30 seconds. On timeout, the mystery is spawned at an initial position.

update_mystery() - Moves the mystery if it is spawned. Moves rightward with a fixed vertical position. Despawns the mystery if it reaches the left end of the screen.

Figure 2: Pseudocode of the software algorithm for the FPGA Cat Invaders game.

```c++
#include <iostream>

using namespace std;

int main()
{
    // Initialize the game state
    Game game;

    // Start the game loop
    while (true)
    {
        // 1. Check if all of the cats have been eliminated
        check_all_eliminated();

        // 2. Check for player collisions (mouse)
        bone_collision();

        // 3. Check for cat collisions (bone)
        mouse_collision();

        // 4. Move the cats (including animations)
        move_cats();
        animate_cats();
    }

    return 0;
}
```
5 The Hardware/Software Interface

Control Registers:

The hardware and software will communicate bidirectionally in 2 byte (16bit) chunks over the Avalon Bus. Tables 3-6 below describe the registers the software and hardware can read and write to. These registers contain movement and visibility information for the sprites. Table 7 describes what each field in the audio registers represent.

Some audio notes:

- Volume: Writing a 15 sets the volume to 0; writing a 0 sets the volume to 15
- Speed: Numbers should between 0 and 6
- Play:
  - When set to 1 Background music will play continuously, 0 will turn it off
  - Bark and Meow will play once. To play Bark or Meow multiple times write a 0 to reset and then a 1.
Table 8 below contains the registers available for reading. The test_reg contains a value to ensure that reads are enabled. The frame_reg contains the current frame count for software hardware synchronization.

Table 3: Register contains - Dog position, and projectile position and visibility.

<table>
<thead>
<tr>
<th>Register #</th>
<th>Register Name</th>
<th>R/W</th>
<th>Bit description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>dog_pos_x</td>
<td>R/W</td>
<td>Reserved [5:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Dog Position X [9:0]</td>
</tr>
<tr>
<td>1</td>
<td>project_d_x</td>
<td>R/W</td>
<td>Reserved [5:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Projectile Dog X [9:0]</td>
</tr>
<tr>
<td>2</td>
<td>project_d_y</td>
<td>R/W</td>
<td>Dog Bullet visible?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reserved [4:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Projectile Dog Y [9:0]</td>
</tr>
<tr>
<td>3</td>
<td>project_c0_x</td>
<td>R/W</td>
<td>Reserved [5:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Projectile Cat0 X [9:0]</td>
</tr>
<tr>
<td>4</td>
<td>project_c0_y</td>
<td>R/W</td>
<td>Cat0 Bullet visible?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reserved [4:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Projectile Cat0 Y [9:0]</td>
</tr>
<tr>
<td>5</td>
<td>project_c1_x</td>
<td>R/W</td>
<td>Reserved [5:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Projectile Cat1 X [9:0]</td>
</tr>
<tr>
<td>6</td>
<td>project_c1_y</td>
<td>R/W</td>
<td>Cat1 Bullet visible?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reserved [4:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Projectile Cat1 Y [9:0]</td>
</tr>
<tr>
<td>7</td>
<td>project_c2_x</td>
<td>R/W</td>
<td>Reserved [5:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Projectile Cat2 X [9:0]</td>
</tr>
<tr>
<td>8</td>
<td>project_c2_y</td>
<td>R/W</td>
<td>Cat2 Bullet visible?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reserved [4:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Projectile Cat2 Y [9:0]</td>
</tr>
<tr>
<td>9</td>
<td>project_c3_x</td>
<td>R/W</td>
<td>Reserved [5:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Projectile Cat3 X [9:0]</td>
</tr>
<tr>
<td>10</td>
<td>project_c3_y</td>
<td>R/W</td>
<td>Cat3 Bullet visible?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reserved [4:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Projectile Cat3 Y [9:0]</td>
</tr>
</tbody>
</table>
### Table 4: Register category - Cat array position, and mystery position

<table>
<thead>
<tr>
<th>Register #</th>
<th>Register Name</th>
<th>R/W</th>
<th>Bit description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>[15:0]</td>
</tr>
<tr>
<td>11</td>
<td>cat_array_pos_x</td>
<td>R/W</td>
<td>Reserved [5:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cat Array Position X [9:0]</td>
</tr>
<tr>
<td>12</td>
<td>cat_array_pos_y</td>
<td>R/W</td>
<td>Reserved [5:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cat Array Position Y [9:0]</td>
</tr>
<tr>
<td>13</td>
<td>mystery_pos_x</td>
<td>R/W</td>
<td>Visibility?</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Reserved [4:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Mystery Position X [9:0]</td>
</tr>
</tbody>
</table>

### Table 5: Register category - Miscellaneous

<table>
<thead>
<tr>
<th>Register #</th>
<th>Register Name</th>
<th>R/W</th>
<th>Bit description</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td>[15:0]</td>
</tr>
<tr>
<td>14</td>
<td>status</td>
<td>R/W</td>
<td>Reserved [7:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Level [3:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Game Status [1:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Life [1:0]</td>
</tr>
<tr>
<td>15</td>
<td>score</td>
<td>R/W</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Score [15:0]</td>
</tr>
<tr>
<td>16</td>
<td>dog_ani</td>
<td>R/W</td>
<td>Reserved [11:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Dog Animation State [3:0]</td>
</tr>
<tr>
<td>17</td>
<td>cat_ani</td>
<td>R/W</td>
<td>Reserved [14:0]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>Cat Animation State</td>
</tr>
</tbody>
</table>
### Table 6: RAM category - barrier and cat matrix

<table>
<thead>
<tr>
<th>Register #</th>
<th>Register Name</th>
<th>R/W</th>
<th>Bit description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>28</td>
<td>Barrier Matrix</td>
<td>R/W</td>
<td>Address (0 to 23)</td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>Cat Matrix</td>
<td>R/W</td>
<td>Address (0 to 55)</td>
<td></td>
</tr>
</tbody>
</table>

### Table 7: Audio register map

<table>
<thead>
<tr>
<th>Register #</th>
<th>Register Name</th>
<th>R/W</th>
<th>Bit description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>42</td>
<td>Background Music</td>
<td>R/W</td>
<td>Reserved</td>
<td>Play Reserved Volume</td>
</tr>
<tr>
<td>43</td>
<td>Meow</td>
<td>R/W</td>
<td>Reserved</td>
<td>Play Reserved Volume</td>
</tr>
<tr>
<td>44</td>
<td>Bark</td>
<td>R/W</td>
<td>Reserved</td>
<td>Play Reserved Volume</td>
</tr>
<tr>
<td>45</td>
<td>Speed</td>
<td>R/W</td>
<td>Reserved</td>
<td>Speed</td>
</tr>
</tbody>
</table>

### Table 8: Test and Frame registers

<table>
<thead>
<tr>
<th>Register #</th>
<th>Register Name</th>
<th>R/W</th>
<th>Bit description</th>
</tr>
</thead>
<tbody>
<tr>
<td>48</td>
<td>Test Reg</td>
<td>R</td>
<td>Return test value of D065</td>
</tr>
<tr>
<td>49</td>
<td>Frame Reg</td>
<td>R/W</td>
<td>Frame Active Frame Count</td>
</tr>
</tbody>
</table>
Polling and Interrupt:

In order to make sure the writing operation of the registers did not interfere with the refresh of the screen we added a register to poll. This method would prevent ghosting, a trail of phantom pixels trailing the sprites. Originally we had planned to implement an interrupt, which is basically a signal generated from the hardware every time the frame updates are complete. However, through reading the documentation and testing we were unable to configure the Global Interrupt Controller (GIC) on the Arm processor to accept the hardware interrupt. Instead we turned towards using the aforementioned polling to provide this synchronization rather than an interrupt. The idea of polling was, on the hardware side, to use one register to store both the active frame bit signal and a frame counter. In the middle of the frame the active frame bit is 1, and outside it is 0. Additionally, once every frame the frame counter is incremented by 1. On the software side, there is a while loop querying this register and comparing it to the previous value. If not equal the software is safe to write to registers on the hardware. This technique ensures the writing and screen refreshing work synchronized.

Cat Matrix:

In the game, all cats are displayed and moved in the form of a matrix. It is controlled by a coordinate that indicates the position of its upper left corner, and an array of display states for the cat matrix. The coordinate is stored in control registers, while the display statuses are stored in a consecutive RAM space named Cat Matrix RAM. Both controlled through software. For each position in the matrix there is an 8 bit display state word that tells the hardware what to display.

<table>
<thead>
<tr>
<th>Cat States</th>
<th>8’d0</th>
<th>8’d1</th>
<th>8’d2</th>
<th>8’d3</th>
<th>8’d4</th>
<th>8’d5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Display Content</td>
<td>Nothing</td>
<td>Cat1</td>
<td>Cat2</td>
<td>Cat3</td>
<td>Cat4</td>
<td>Explosion</td>
</tr>
</tbody>
</table>

Table.9 shows the correspondence between the value of cat states and display content in a particular position. If the cat state is 0, it means the cat at that position has been destroyed and should display nothing. If catstatus ranges from 1-4, one of the four kinds of cats (stored in four separate ROMs) should be fetched from the image ROM and displayed. Status being 5 means the cat at that position was just hit by a dog projectile and should display the explosion image.

Barrier Matrix:

Similar to cat matrix, the barriers are also displayed as a matrix and are managed through an array of damage rate stored in a consecutive RAM space named Barrier Matrix RAM. Since the barriers do not move, no control registers are needed to control their positions. There are 4
barriers, and each barrier consists of 12 barrier blocks. For each barrier block, there is an 8 bit word that represents the damage rate from 0 to 4, where damage rate 0 means the block is undamaged and damage rate 4 means the block is fully damaged and no longer exists. The barrier block has 4 images in the same ROM with different offsets, representing 4 damage rates. The hardware will choose which image to display based on the damage rates in the Barrier Matrix RAM.

Movement Control:
In the game, movable objects are dog, cat matrix, mystery and projectiles, in which dog and mystery can only move in one direction and cat matrix and projectiles can move in all directions. Movable objects have either X or both XY coordinates stored in the control registers. During the game, software can make them move by constantly changing the coordinates of the objects.

Visibility:
Besides movement, the control registers also control the visibility of objects. For projectile and mystery, the MSB of their coordinate registers are used to control their visibility. And for life, barrier and cat matrix, visibility is controlled in their own state register/RAM. Pixel selector will omit the objects if they are turned invisible.

Numbers:
The numbers were stored in ROM and displayed for the level and the score. The software converted the decimal numbers into 4 bit binary coded decimal numbers and wrote these values to registers. The hardware used these bits to draw the correct number to the screen from the ROM.

Userspace/Kernel interface:
In order to make userspace communicate with the hw through the kernel conveniently, we created an interface between the userspace and the kernel, and translated the writing register operation into user friendly functions, like setting the position of the dog, setting the statue of the cat matrix and barrier, set the score number(do the decimal to binary) and so on. In the interface, not only do the kernel command calling, but also deal with the data that need to be reformulated or shifted to the right position, some of them also need to be combined together, and the header of the interface is shown in the code below.
6 Audio Interface

The FPGA Cat Invader games will output background music and sound effects. The DE1-SoC contains a Wolfson WM8731 audio CODEC [4] for handling 24-bit audio. Figure 3 shows the WM8731 physical connections to the rest of the FPGA as well as the Mic in, Line-in In, and Line-out jacks for microphone input and speaker output. The audio CODEC settings are adjustable supporting a range of 8ks/s to 96 ks/s sample rates for both the ADC and DAC, for this project the sample rate will be set to 8ks/s for both the ADC and DAC. This sample rate was chosen to keep the overall memory footprint of the audio small without sacrificing the overall quality of the sound.

Figure 3: Wolfson WM8731 connection to the FPGA (image source [4]).

The ALTERA University Program provides two Intellectual Property (IP) blocks of use for the control of the audio. Figure 4 is a high-level block diagram of the Audio/Video Configuration core IP block which handles the configuration and control of both Audio and Video. This core will be used for the overall configuration of the onboard audio control and datapath through the Avalon Bus. Figure 5 is the Audio core block which handles the input and output of audio through the Wolfson WM8731 audio CODEC. For this project, the IP will be set to Memory-Mapped Interface. As seen in Figure 5 the IP supports both input (Deserializer) and output, however for this project only the output path will be configured for use. Additionally, the IP supports both right and left audio output, the same sample will be fed into both channels. Lastly, the FIFO has synchronization feedback to control the flow of the data into the FIFOs, if the FIFO reports it is filled the overall audio pipeline will halt until space has been freed up.
To facilitate multiple audio clips played together simultaneously the audio will first be divided by a power of two based on the volume set for that clip (0 to 15). After reducing the amplitude the clips will be added together and placed in the audio core’s left and right fifo.

In summary, the overall flow of the audio is as follows:

1. On FPGA boot-up the Audio IP core is configured to the preset configuration.
2. The software writes to one of the audio registers with the play bit to 1.
3. The FPGA reads the sample from ROM (one at a time starting at offset 0) and copies the value into the left data of the audio core IP. If more than one audio sample is being played the samples will be added together and played through the speaker.
4. The audio core outputs the 8kHz sample and outputs 48kHz (due to the chosen phase lock loop (PLL) clock frequency). If the number of samples has been reached:
- If the loop value is 0 the FPGA will stop playing the audio.
- If the loop value is 1 the FPGA will start at the top and copy samples from offset 0.

The audio registers support both a speed and volume register. These were mainly used for debugging purposes to determine the optimal volume and speed settings for each audio clip. Speed control works by setting a number (in the register) that dictates how many times the same audio sample should be repeatedly loaded into the audio FIFOs. Volume control was implemented by using a signed right shift (shift for 0 to 15). In the end the speed register was hardcoded to a value of 2 and volume was hardcoded to 0.

Figure 6 below shows the high level implementation for the audio hardware. The audio clips are the audio mif files stored in Read Only Memory (ROM). The Audio Selector algorithm will decide based on the speed control if the ROM will be incremented to the next address or output the duplicate sample. Additionally, this logic decides which of the clips will be fed into shift and add logic before reaching the Left and Right FIFO. The Audio output logic containing the Audio IP and Codec takes the supplied samples and outputs them to the speaker.

Figure 6: Flowchart of audio implementation.
7 Video Display

For basic image display, we use logic similar to that in lab3. When hcount and vcount scan to a specific area, we read the corresponding image pixel data from the ROM and perform the aforementioned RGB bit-shifting expansion before using it as VGA input.

Among all the images, the cat matrix, explosion, poodle, bone, mouse, and mystery are movable. They need to receive starting coordinate information (for the top-left pixel) from the ctrl_reg, which is sent by the software. The remaining images such as barriers, numbers, scores, and lives are set in fixed positions on the hardware side.

We have added animation effects for cats and poodles by emitting animation state signals through the software-side control logic. The hardware side receives these signals and selects which image from the ROM to display based on the state. Through rapid state switching, the hardware can quickly switch between different images to achieve the effect of animation. The FPGA Cat Invader games will output the game to a VGA monitor. In order to accomplish this, the onboard the DE1-SoC FPGA produces the synchronization and red, green, blue (RGB) signals through the ADV7123 DAC and out a 15-pin D-SUB connector. Additionally, the circuit is capable of supporting up to 1280x1024 at 100MHz. For this project the display monitor will output 640x480 resolution. Figure 7 shows the signals connected between the FPGA and ADV7123 VGA DAC [3].

Figure 7: Connections between the FPGA and VGA(image source [3]).
Figure 8 outlines the key timing for displaying a row on a VGA screen. An active-low hsync pulse indicates the end of one row and the start of the next. After this pulse, the RGB data must be off during the back porch, then on during the display interval as pixels are activated across the row. The RGB is off again during the front porch before the next hsync pulse. Vsync timing is similar but marks the end and start of entire frames, not just rows [3].

Figure 8: VGA horizontal timing specification(image source [3]).

Display Hardware Architecture:

Figure 9 shows the overall hardware architecture of the display. All pixel information is stored in separate ROMs and will be read out one pixel every VGA cycle. During the game, ROM address generation and pixel selection works in parallel to ensure that pixels are displayed correctly right after they are read from the ROMs.

On the address generation side, the next address of each ROM is decided according to the current hcount & vcount, current ROM address, visibility and other important factors such as the states of barrier and cat matrix. On the pixel selection side, pixel data from every ROM will be sorted by priority and decide which pixel to put on the screen in this cycle.
As an example, the following code demonstrates the address generation and pixel selection logic of mystery. The address generation logic is designed like a counter that is enabled when hcount and vcount enter the display region of the mystery. As for the pixel selection logic, it can be seen that all objects are put in the cascade of if-else statements, which functions as pixel selector. This mechanism can ensure that when two objects overlap, only the object with higher priority will be displayed.

```verilog
always_ff @(posedge clk) begin
    if (~hcount[0]) begin
        if (hcount[10:1] >= mystery_pos_x && hcount[10:1] < (mystery_pos_x+48) && vcount >= mystery_pos_s_y && vcount < (mystery_pos_e_y)) begin
            if (hcount[10:1] == (mystery_pos_x+47) && vcount == (mystery_pos_e_y-1)) mystery_addr <= 0;
            else mystery_addr <= mystery_addr + 1;
        end
    end
end
```

Figure 9: Block diagram of hardware display architecture
always_comb begin
    {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};
    if (VGA_BLANK_n)
        /* start */
        if (hcount[10:1] >= start_s_x && hcount[10:1] < start_e_x && vcount >= start_s_y && vcount < start_e_y && others_data != 8'd0 &&
            game_status == 2'b00)
            {VGA_R, VGA_G, VGA_B} = {{others_data[7:5], 5'd0},
             {others_data[4:2], 5'd0}, {others_data[1:0], 6'd0}};
        ....
        /* mystery */
        else if (hcount[10:1] >= mystery_pos_x && hcount[10:1] <
            (mystery_pos_x+48) && vcount >= mystery_pos_s_y &&
            vcount < (mystery_pos_e_y) && mystery_data != 8'd0)
            {VGA_R, VGA_G, VGA_B} = {{mystery_data[7:5], 5'd0},
             {mystery_data[4:2], 5'd0}, {mystery_data[1:0], 6'd0}};
        ....
    end
end

8 Memory

In this game, we use ROMs to store all pixel data, and we use RAMs for control purposes. The following tables show the memory contents and their address range.

ROM Layout:

Table 10: Cat 1 memory layout (ROM)

<table>
<thead>
<tr>
<th></th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pose 1</td>
<td>0x0000 - 0x03FF</td>
</tr>
<tr>
<td>Pose 2</td>
<td>0x0400 - 0x07FF</td>
</tr>
</tbody>
</table>
Table 11: Cat 2 memory layout (ROM)

<table>
<thead>
<tr>
<th>Pose</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x0000 - 0x03FF</td>
</tr>
<tr>
<td>2</td>
<td>0x0400 - 0x07FF</td>
</tr>
</tbody>
</table>

Table 12: Cat 3 memory layout (ROM)

<table>
<thead>
<tr>
<th>Pose</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x0000 - 0x03FF</td>
</tr>
<tr>
<td>2</td>
<td>0x0400 - 0x07FF</td>
</tr>
</tbody>
</table>

Table 13: Cat 4 memory layout (ROM)

<table>
<thead>
<tr>
<th>Pose</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x0000 - 0x03FF</td>
</tr>
<tr>
<td>2</td>
<td>0x0400 - 0x07FF</td>
</tr>
</tbody>
</table>

Table 14: Poodle memory layout (ROM)

<table>
<thead>
<tr>
<th>Poodle Pose</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0x0000 - 0x03FF</td>
</tr>
<tr>
<td>2</td>
<td>0x0400 - 0x07FF</td>
</tr>
<tr>
<td>3</td>
<td>0x0800 - 0xBFF</td>
</tr>
<tr>
<td>4</td>
<td>0x0C00 - 0xFFF</td>
</tr>
<tr>
<td>5</td>
<td>0x1000 - 0x13FF</td>
</tr>
<tr>
<td>6</td>
<td>0x1400 - 0x17FF</td>
</tr>
<tr>
<td>7</td>
<td>0x1800 - 0x1BFF</td>
</tr>
<tr>
<td>8</td>
<td>0x1C00 - 0x1FFF</td>
</tr>
<tr>
<td>9</td>
<td>0x2000 - 0x23FF</td>
</tr>
<tr>
<td>10</td>
<td>0x2400 - 0x27FF</td>
</tr>
</tbody>
</table>
### Table 15: Barrier memory layout (ROM)

<table>
<thead>
<tr>
<th>Barrier state</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barrier 1</td>
<td>0x0000 - 0x00FF</td>
</tr>
<tr>
<td>Barrier 2</td>
<td>0x0100 - 0x01FF</td>
</tr>
<tr>
<td>Barrier 3</td>
<td>0x0200 - 0x02FF</td>
</tr>
<tr>
<td>Barrier 4</td>
<td>0x0300 - 0x03FF</td>
</tr>
</tbody>
</table>

### Table 16: Mystery ROM

<table>
<thead>
<tr>
<th>Pose</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pose 1</td>
<td>0x0000 - 0x08FF</td>
</tr>
<tr>
<td>Pose 2</td>
<td>0x0900 - 0x11FF</td>
</tr>
</tbody>
</table>

### Table 17: Bone ROM

<table>
<thead>
<tr>
<th>Pose</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pose 1</td>
<td>0x0000 - 0x00C0</td>
</tr>
</tbody>
</table>

### Table 18: Number memory layout (ROM)

<table>
<thead>
<tr>
<th>Number</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0x0000 - 0x03FF</td>
</tr>
<tr>
<td>1</td>
<td>0x0400 - 0x07FF</td>
</tr>
<tr>
<td>2</td>
<td>0x0800 - 0x0BFF</td>
</tr>
<tr>
<td>3</td>
<td>0x0C00 - 0x0FFF</td>
</tr>
<tr>
<td>4</td>
<td>0x1000 - 0x13FF</td>
</tr>
<tr>
<td>5</td>
<td>0x1400 - 0x17FF</td>
</tr>
<tr>
<td>6</td>
<td>0x1800 - 0x1BFF</td>
</tr>
<tr>
<td>7</td>
<td>0x1C00 - 0x1FFF</td>
</tr>
<tr>
<td>8</td>
<td>0x2000 - 0x23FF</td>
</tr>
<tr>
<td>9</td>
<td>0x2400 - 0x27FF</td>
</tr>
</tbody>
</table>
Table 19: Others ROM

<table>
<thead>
<tr>
<th>Other</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>explode</td>
<td>0x0000 - 0x03FF</td>
</tr>
<tr>
<td>level</td>
<td>0x0400 - 0xFFFF</td>
</tr>
<tr>
<td>Life</td>
<td>0x1000 - 0x13FF</td>
</tr>
<tr>
<td>mouse</td>
<td>0x1400 - 0x147F</td>
</tr>
<tr>
<td>score</td>
<td>0x1480 - 0x207F</td>
</tr>
<tr>
<td>start</td>
<td>0x2080 - 0x3B7F</td>
</tr>
</tbody>
</table>

Table 20: Background ROM

<table>
<thead>
<tr>
<th>Audio Clip</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Background Music</td>
<td>0x0000 - 0x8B3C</td>
</tr>
</tbody>
</table>

Table 21: Cat Meow ROM

<table>
<thead>
<tr>
<th>Audio Clip</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cat meow</td>
<td>0x0000 - 0x1822</td>
</tr>
</tbody>
</table>

Table 22: Dog Bark ROM

<table>
<thead>
<tr>
<th>Audio Clip</th>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dog Bark</td>
<td>0x0000 - 0x1001</td>
</tr>
</tbody>
</table>

RAM Layout:

Table 23: Barrier Matrix RAM

<table>
<thead>
<tr>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Barrier Matrix</td>
</tr>
<tr>
<td>0x0000 - 0x0018</td>
</tr>
</tbody>
</table>

Table 24: Cat Matrix RAM

<table>
<thead>
<tr>
<th>Address range</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cat Matrix</td>
</tr>
<tr>
<td>0x0000 - 0x0037</td>
</tr>
</tbody>
</table>
We categorize images and audios by type and store them in different ROMs. We first created the audio mifs by modifying 365-audio-tools wavToMif.py [15] python script and the sprite mifs with by modifying Nananas, ImageToMIf python script [14]. We merge mif files of the same category using a Python script, then use Qsys Platform Designer to create the corresponding IP core. Since a ROM often contains multiple mif files of images, we also generate an address map when creating the merged mif file. This facilitates reading different image data from the same ROM in the hardware display code.

9 Game Controller

The player will interact with the game using a iNNEXT joypad, which is shown in Figure 10. The controller sends and receives over the usb connection and the software uses the libusb library to communicate.

Figure 10: Image of iNNEXT Joypad.

The buttons on the joypad and their corresponding functions are shown in Table 25. In summary, there will be 4 buttons used in the whole game; the left and right arrows for moving the dog on the horizontal axis, the ‘start’ button for starting the game, and the ‘A’ button for attacking. When holding either the left or right arrows the dog will move repeatedly in the direction of the arrow until reaching the boundary. Similarly, holding the attack (‘A’) button will cause the dog to keep firing at a fixed interval.

Table 25: Buttons and corresponding functions.

<table>
<thead>
<tr>
<th>Button</th>
<th>Function</th>
</tr>
</thead>
<tbody>
<tr>
<td>Left arrow</td>
<td>Move Left</td>
</tr>
<tr>
<td>Right arrow</td>
<td>Move right</td>
</tr>
<tr>
<td>Start</td>
<td>Start the game</td>
</tr>
<tr>
<td>A</td>
<td>Attack</td>
</tr>
</tbody>
</table>
Just as the keyboard USB interface does, the joypad will be configured through libusb, which follows USB protocol. To be specific, the protocol message is 8 bytes for each event; some of the parameters are displayed in Table 26.

**Table 26: Communication parameters of the device over libusb.**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>idvendor (VID)</td>
<td>0x0079</td>
</tr>
<tr>
<td>idProduct (PID)</td>
<td>0x0011</td>
</tr>
<tr>
<td>Packet size</td>
<td>1*8 bytes</td>
</tr>
</tbody>
</table>

The full detail of the communication protocol is demonstrated in the Table 27 below, including all the single button events and combo events which may happen in the game. When each button is pressed the value at the corresponding byte changes from its default value to the new value.
Table 27: Communication protocol packet definition of button presses over libusb.

<table>
<thead>
<tr>
<th></th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
</tr>
</thead>
<tbody>
<tr>
<td>default</td>
<td>01</td>
<td>7F</td>
<td>7F</td>
<td>7F</td>
<td>0F</td>
<td>00</td>
<td>002</td>
<td>202</td>
</tr>
<tr>
<td>A</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>2F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>B</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>4F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>8F</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Up arrow</td>
<td></td>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Down arrow</td>
<td></td>
<td></td>
<td>00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Left arrow</td>
<td></td>
<td></td>
<td></td>
<td>FF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Right arrow</td>
<td></td>
<td></td>
<td></td>
<td>FF</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>L2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>01</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>R2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>02</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Start</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Select</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>10</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Left arrow + A</td>
<td>00</td>
<td></td>
<td></td>
<td>2F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Right arrow +A</td>
<td>FF</td>
<td></td>
<td></td>
<td>2F</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
10 Conclusion

The FPGA Cat Invaders game is a fully functional demo complete with all features described in this report. The cat matrix moved in the correct pattern and dropped mice, the poodle fired bones and eliminated the cats (+1 point) using the controller, a mystery sprite appeared at random intervals (+10 points), and all the music/sound clips played. In the end, the hardware and software worked together to make this game possible.

10.1 Challenges and Lesson learned

A lesson we learned is to always have a good debugging tool in hand. In our case, we did not make use of simulation and waveform display tools such as Verilator and GTKwave when debugging and validating our Verilog code. Instead we debug by loading the code to the FPGA everytime and see what shows up, which turned out to be very inefficient and slowed down our progress in hardware development.

Another lesson we learned is to budget all available resources carefully before implementing. We ran out of memories several times in the course of development due to the large audio files. Each time we have to reallocate the memory, which takes a lot of time.

Working on a team interfacing both hardware and software, it very quickly became apparent that cross referencing hardware implementation was extremely important when developing the software. Some hardware bugs may be less obvious only until a significant portion of software has been implemented, and at times it is easier to work with and try to correct the errors in software rather than trying to solve the issue in hardware.

Communication and coordination was also something that we began to get better at as time progressed. Scheduling meetings and discussing what has been done and what needed to be done became easier and more productive as the project progressed and we began to settle more comfortably as a group.

10.2 Contributions

Adam Jablonski: Responsible for audio section from generating the mifs to playing out of the speakers, barrier matrix with RAM implementation, hardware cleanup, software help, and project coordination.

Yuxiang Xia: Responsible for memory management, image ROM generation, display logic for movable images, image animation logic, and overall adjustment and optimization of hardware display.
Yilei Meng: Responsible for software kernel stuff, corresponding to the register table and enable the write data into the register; Programmed the interface for the software user space to use for each commands todo with the hardware; Enable the controller input through the usb protocol; Generated the display of top level signs including score, level and life in HW.

Zhili Yan: Responsible for display logic for cat matrix, specification and hardware implementation of control registers. Added movement and visibility control to the hardware display.

Fernandos Magee: Responsible for creating sprites and animations, generating MIFs for sprites with RGB 332 color mapping, and designing and implementation of software algorithms used in the final product’s game logic.

10.3 Acknowledgement

We would like to thank Marc Jablonski, a New York City based sound designer for providing all the audio clips used in this project [9][10][11].
11 References


Appendix A: Code

12.1 cat_invaders.sv

/*
 * Avalon memory-mapped peripheral that generates VGA
 * Stephen A. Edwards
 * Columbia University
 */

module cat_invaders(input logic clk,
                    input logic reset,
                    input logic left_channel_ready,
                    input logic right_channel_ready,
                    input logic [15:0] writedata,
                    input logic write,
                    input chipselect,
                    input logic [7:0] address,
                    output logic [15:0] readdata,
                    output logic [7:0] VGA_R, VGA_G, VGA_B,
                    output logic VGA_CLK, VGA_HS, VGA_VS,
                    VGA_BLANK_n,
                    output logic VGA_SYNC_n,
                    output logic left_channel_valid,
                    output logic right_channel_valid,
                    output logic [15:0] left_channel_data,
                    );


```verilog
output logic [15:0] right_channel_data,
output logic eof_irq

);

parameter CAT_SIZE = 32;
parameter CAT_MATRIX_SIZE_X = 352;
parameter CAT_MATRIX_SIZE_Y = 160;

/****************************
Register Group #0
Dog position; and projectile position and visibility
/****************************/

logic [3:0] cat_bullet; //cat bullet in flight?
logic dog_bullet; //dog bullet in flight?
logic [9:0] dog_pos_x; //dog position x
logic [9:0] dog_project_x; //dog bullet position x
logic [9:0] dog_project_y; //dog bullet position y
logic [9:0] cat_project_x[3:0]; //cat 0-3 bullet position x
logic [9:0] cat_project_y[3:0]; //cat 0-3 bullet position y
logic [9:0] cat_array_pos_x; //cat upper west corner position x
logic [9:0] cat_array_pos_y; //cat upper west corner position y
logic [9:0] mystery_pos_x; //mystery position
logic mystery_vis; //mystery visible

/****************************
Register Group #1
Miscellaneous
/****************************/

logic [3:0] level; //game difficulty level 0-7
```
logic [1:0] game_status;    // game status (start, in process, game over)
logic [1:0] life;            // lives remaining 0-3
logic [15:0] score;         // score
logic [3:0] dog_ani_state;
logic    cat_ani_state;

ctrl_regs ctr_regs_0(.*);

logic [10:0] hcount;
logic [9:0] vcount;
logic    irq;
logic [14:0] frame_cnt;
logic    frame_active;

logic [7:0]    background_r, background_g, background_b;
logic play, play_cat, play_dog;
logic [3:0] vol, vol_cat, vol_dog;
logic [3:0] delay;

vga_counters counters(.clk50(clk), .*);

audio_player audio(.clk50(clk), .*);

/**********************************************
display start
**********************************************/
logic [23:0] display_rgb;

logic [10:0] display_addr;

logic test;

logic [10:0] cat2_addr;
logic [10:0] cat3_addr;
logic [10:0] cat4_addr;
logic [10:0] explode_addr;
logic [15:0] poodle_addr;
logic [15:0] mystery_addr;
logic [15:0] barrier_addr;
logic [15:0] game_over_addr;
logic [15:0] num_addr;
logic [15:0] others_addr;

//logic [15:0] others_explosion_addr;

logic [15:0] cat_mat_addr;
logic [15:0] barrier_mat_addr;
logic [15:0] bone_addr;

logic [7:0] display_data;
logic [7:0] cat2_data;
logic [7:0] cat3_data;
logic [7:0] cat4_data;
logic [7:0] explode_data;
logic [7:0] poodle_data;
logic [7:0] mystery_data;
logic [7:0] barrier_data;
logic [7:0] game_over_data;
logic [7:0] num_data;
logic [7:0] others_data;
logic [7:0] cat_mat_data;
logic [7:0] barrier_mat_data;
logic [7:0] bone_data;
logic [7:0] output_data;

logic [15:0] cat_mat_s_x;
logic [15:0] cat_mat_s_y;
logic [15:0] cat_mat_e_x;
logic [15:0] cat_mat_e_y;

------------------------------------------/mouse
logic [15:0] mouse1_s_x;
logic [15:0] mouse1_s_y;
logic [15:0] mouse1_e_x;
logic [15:0] mouse1_e_y;

logic [15:0] mouse2_s_x;
logic [15:0] mouse2_s_y;
logic [15:0] mouse2_e_x;
logic [15:0] mouse2_e_y;

logic [15:0] mouse3_s_x;
logic [15:0] mouse3_s_y;
logic [15:0] mouse3_e_x;
logic [15:0] mouse3_e_y;

/////////////////////////////////////
```verilog
//bone

logic [15:0] bone_s_x;
logic [15:0] bone_s_y;

logic [15:0] bone_e_x;
logic [15:0] bone_e_y;

logic [4:1] ccc;

logic b_write;
logic b_cs;
logic [17:0] b_write_addr;
logic [15:0] b_writedata;

logic c_write;
logic c_cs;
logic [17:0] c_write_addr;
logic [15:0] c_writedata;

soc_system_ROM_cat_1 cat_display (.address(display_addr), .clk(clk), .clken(1),
reset(reset), .reset_req(0), .readdata(display_data));

soc_system_ROM_cat_2 cat2_display (.address(cat2_addr), .clk(clk), .clken(1),
reset(reset), .reset_req(0), .readdata(cat2_data));

soc_system_ROM_cat_3 cat3_display (.address(cat3_addr), .clk(clk), .clken(1),
reset(reset), .reset_req(0), .readdata(cat3_data));

soc_system_ROM_cat_4 cat4_display (.address(cat4_addr), .clk(clk), .clken(1),
reset(reset), .reset_req(0), .readdata(cat4_data));

soc_system_ROM_explode explode_display (.address(explode_addr), .clk(clk), .clken(1),
reset(reset), .reset_req(0), .readdata(explode_data));

soc_system_cat_matrix cat_mat_display (.address(cat_mat_addr), .address2(c_write_addr),
chipselect2(c_cs), .clk(clk), .clken(1), .clken2(1), .reset(reset), .reset_req(0),
write2(c_write), .writedata2(c_writedata), .readdata(cat_mat_data));
```
soc_system_ROM_Poodles poodle_display(.address(poodle_addr), .clk(clk), .clken(1), .reset(reset), .reset_req(0), .readdata(poodle_data));
soc_system_ROM_mystery mystery_display(.address(mystery_addr), .clk(clk), .clken(1), .reset(reset), .reset_req(0), .readdata(mystery_data));
soc_system_ROM_barrier barrier_display(.address(barrier_addr), .clk(clk), .clken(1), .reset(reset), .reset_req(0), .readdata(barrier_data));
soc_system_ROM_game_over game_over_display(.address(game_over_addr), .clk(clk), .clken(1), .reset(reset), .reset_req(0), .readdata(game_over_data));
soc_system_ROM_num num_display(.address(num_addr), .clk(clk), .clken(1), .reset(reset), .reset_req(0), .readdata(num_data));
soc_system_ROM_others others_display(.address(others_addr), .clk(clk), .clken(1), .reset(reset), .reset_req(0), .readdata(others_data));
soc_system_ROM_bone bone_display(.address(bone_addr), .clk(clk), .clken(1), .reset(reset), .reset_req(0), .readdata(bone_data));

/*********************************************/
display_end
/*********************************************/
logic [4:0] cat_cnt_x;
logic [4:0] cat_cnt_y;
logic [10:0] cat_offset_x;
logic [10:0] cat_offset_y;
logic [10:0] cat_addr;
logic [15:0] explosion_addr;
logic in_box;
assign cat_cnt_x = (({1'b0,hcount[10:1]}-cat_mat_s_x[10:0]+1) & 1l'hlf);
assign cat_cnt_y = (({1'b0,vcount}-cat_mat_s_y[10:0]) & 1l'hlf);
assign cat_offset_x = (({1'b0,hcount[10:1]}-cat_mat_s_x[10:0]) & 1l'hlf);
assign cat_addr = {cat_ani_state, cat_offset_y[4:0], cat_offset_x[4:0]};

assign in_box = (hcount[10:1] >= (cat_mat_s_x-1) && (hcount[10:1] < cat_mat_e_x) && (vcount > cat_mat_s_y) && (vcount < cat_mat_e_y));
assign display_addr = {16{in_box}} & cat_addr;
assign cat2_addr = {16{in_box}} & cat_addr;
assign cat3_addr = {16{in_box}} & cat_addr;
assign cat4_addr = {16{in_box}} & cat_addr;
assign explode_addr = {16{in_box}} & cat_addr;

/******************************mice***************************/
/* mouse */
logic [15:0] mouse1_addr, mouse2_addr, mouse3_addr;
logic [10:0] mouse1_offset_x;
logic [10:0] mouse1_offset_y;
logic [10:0] mouse2_offset_x;
logic [10:0] mouse2_offset_y;
logic [10:0] mouse3_offset_x;
logic [10:0] mouse3_offset_y;
logic in_box_mouse1;
logic in_box_mouse2;
logic in_box_mouse3;

assign mouse1_offset_x = (({1'b0,hcount[10:1]}-mouse1_s_x[10:0]+2) & 1l'hlf);
assign mouse1_offset_y = (({1'b0,vcount}-mouse1_s_y[10:0]) & 1l'hlf);
assign mouse1_addr = {9'd0, mouse1_offset_y[1:0], mouse1_offset_x[2:0]};

assign mouse2_offset_x = (({1'b0,hcount[10:1]}-mouse2_s_x[10:0]+2) & 1l'hlf);
assign mouse2_offset_y = (({1'b0,vcount}-mouse2_s_y[10:0]) & 1l'hlf);
assign mouse2_addr = {9'd0, mouse2_offset_y[3:0], mouse2_offset_x[2:0]};

assign mouse3_offset_x = {{1'b0, hcount[10:1]}-mouse3_s_x[10:0]+2} & 11'hlf;
assign mouse3_offset_y = {{1'b0, vcount}-mouse3_s_y[10:0]} & 11'hlf;
assign mouse3_addr = {9'd0, mouse3_offset_y[3:0], mouse3_offset_x[2:0]};

assign in_box_mouse1 = (hcount[10:1]>=mouse1_s_x-2) && (hcount[10:1]<mouse1_e_x) && (vcount>=mouse1_s_y) && (vcount<mouse1_e_y);
assign in_box_mouse2 = (hcount[10:1]>=mouse2_s_x-2) && (hcount[10:1]<mouse2_e_x) && (vcount>=mouse2_s_y) && (vcount<mouse2_e_y);
assign in_box_mouse3 = (hcount[10:1]>=mouse3_s_x-2) && (hcount[10:1]<mouse3_e_x) && (vcount>=mouse3_s_y) && (vcount<mouse3_e_y);

assign bone_s_x = dog_project_x;
assign bone_s_y = dog_project_y;
assign bone_e_x = bone_s_x + 16'd8;
assign bone_e_y = bone_s_y + 16'd24;

assign mouse1_s_x = cat_project_x[1];
assign mouse2_s_x = cat_project_x[2];
assign mouse3_s_x = cat_project_x[3];
assign mouse1_s_y = cat_project_y[1];
assign mouse2_s_y = cat_project_y[2];
assign mouse3_s_y = cat_project_y[3];

assign mouse1_e_x = mouse1_s_x + 4'd8;
assign mouse2_e_x = mouse2_s_x + 4'd8;
assign mouse3_e_x = mouse3_s_x + 4'd8;
assign mouse1_e_y = mouse1_s_y + 5'd16;
assign mouse2_e_y = mouse2_s_y + 5'd16;
assign mouse3_e_y = mouse3_s_y + 5'd16;

assign cat_mat_s_x = cat_array_pos_x;
assign cat_mat_s_y = cat_array_pos_y;
assign cat_mat_e_x = cat_mat_s_x + CAT_MATRIX_SIZE_X;
assign cat_mat_e_y = cat_mat_s_y + CAT_MATRIX_SIZE_Y;

/*************************************************************/

logic [15:0] level_addr, level_num_addr, score_addr, score_n0_addr, score_n1_addr, score_n2_addr, life1_addr, life2_addr, life3_addr, start_addr;

parameter level_s_x = 10; parameter level_e_x = level_s_x + 96;
parameter level_num_s_x = 110; parameter level_num_e_x = level_num_s_x + 32;
parameter score_s_x = 400; parameter score_e_x = score_s_x + 96;
parameter score_n0_s_x = 500; parameter score_n0_e_x = score_n0_s_x + 32;
parameter score_n1_s_x = 540; parameter score_n1_e_x = score_n1_s_x + 32;
parameter score_n2_s_x = 580; parameter score_n2_e_x = score_n2_s_x + 32;
parameter life1_s_x = 200; parameter life1_e_x = life1_s_x + 32;
parameter life2_s_x = 240; parameter life2_e_x = life2_s_x + 32;
parameter life3_s_x = 280; parameter life3_e_x = life3_s_x + 32;
parameter start_s_x = 248; parameter start_e_x = 392;
parameter start_s_y = 210; parameter start_e_y = 258;
parameter dog_pos_s_y = 430; parameter dog_pos_e_y = 462;
parameter mystery_pos_s_y = 50; parameter mystery_pos_e_y = 98;
parameter game_over_s_x = 222; parameter game_over_e_x = 418;
parameter game_over_s_y = 210; parameter game_over_e_y = 238;

always_ff @(posedge clk) begin
  if (reset) cat_mat_addr <= 0;
  else begin
    if (cat_offset_x == 31 && in_box && ~hcount[0]) begin
      if (cat_cnt_x == 10) begin
        if (cat_offset_y == 31) cat_mat_addr <= (cat_cnt_y == 4)? 0 : (cat_mat_addr + 1);
    }}
else cat_mat_addr <= cat_mat_addr - 10;
end

else cat_mat_addr <= cat_mat_addr + 1;
end
else cat_mat_addr <= cat_mat_addr;
end
end

always_comb begin
  if ($(hcount[10:1])>=cat_mat_s_x) && $(hcount[10:1]<cat_mat_e_x) && $(vcount>=(cat_mat_s_y)) && $(vcount<(cat_mat_e_y))
    begin
      case(cat_mat_data)
        0: output_data = 8'd0;
        1: output_data = display_data;
        2: output_data = cat2_data;
        3: output_data = cat3_data;
        4: output_data = cat4_data;
        5: output_data = explode_data;
        default: output_data = 8'hff;
      endcase
    end
  else output_data = 8'd0;
end

logic [15:0] cat_mat_count;
logic [7:0] i_barrier_cnt;
logic [4:0] o_barrier_cnt;
parameter delta_x = 70;
parameter delta_y = 10;

always_ff @(posedge clk)begin
  if (reset) begin
    cat_mat_count <= 0;
    i_barrier_cnt <= 0;
    o_barrier_cnt <= 0;
  end
  else begin
    if (i障) begin
      o_barrier_cnt <= o_barrier_cnt + 1;
      if (o_barrier_cnt == delta_x) begin
        i_barrier_cnt <= i_barrier_cnt - 1;
      end
    end
  end
end
cat_mat_count <= 0;
    test <= 0;
    barrier_addr <= 0;
    i_barrier_cnt <= 0;
    o_barrier_cnt <= 0;
end
else
    if(!hcount[0]) begin

        if (hcount[10:1]==dog_pos_x && hcount[10:1]<{dog_pos_x +32} &&
            vcount==dog_pos_s_y && vcount<(dog_pos_e_y)) begin
            test <=1;

            if (hcount[10:1] == (dog_pos_x+31) && vcount == (dog_pos_e_y-1))
               //poodle_addr <= 16'h0000;

                begin
                    case(dog_ani_state)
                        4'd0:  poodle_addr <= 16'h0000 ;
                        4'd1:  poodle_addr <= 16'h0400 ;
                        4'd2:  poodle_addr <= 16'h0800 ;
                        4'd3:  poodle_addr <= 16'h0C00 ;
                        4'd4:  poodle_addr <= 16'h1000 ;
                        4'd5:  poodle_addr <= 16'h1400 ;
                        4'd6:  poodle_addr <= 16'h1800 ;
                        4'd7:  poodle_addr <= 16'h1C00 ;
                        4'd8:  poodle_addr <= 16'h2000 ;
                        4'd9:  poodle_addr <= 16'h2400 ;
                        default: poodle_addr <= 16'h0000 ;
                    endcase
                end

        else  poodle_addr <= poodle_addr + 1;
    end
    test <= 1;
    if (hcount[10:1] == (mystery_pos_x + 47) && vcount == (mystery_pos_e_y - 1))
        mystery_addr <= 0;
    else mystery_addr <= mystery_addr + 1;
end

if (hcount[10:1] >= game_over_s_x && hcount[10:1] < game_over_e_x && vcount == game_over_e_y && hcount[0]) begin
    test <= 1;
    if (hcount[10:1] == (game_over_e_x - 1) && vcount == (game_over_e_y - 1))
        game_over_addr <= 0;
    else game_over_addr <= game_over_addr + 1;
end

if (hcount[10:1] == bone_s_x && hcount[10:1] < bone_e_x && vcount == bone_e_y && hcount[0]) begin
    test <= 1;
    if (hcount[10:1] == bone_e_x - 1 && vcount == bone_e_y - 1)
        bone_addr <= 0;
    else bone_addr <= bone_addr + 1;
end

if ((hcount[10:1] >= 54 + delta_x && hcount[10:1] < 102 + delta_x && vcount == 336 + delta_y) ||
    (hcount[10:1] >= 354 + delta_x && hcount[10:1] < 402 + delta_x && vcount == 336 + delta_y)) begin
    if (i_barrier_cnt == 191 && vcount == 351 + delta_y)
        begin
            o_barrier_cnt <= 0;
            i_barrier_cnt <= 0;
            barrier_mat_addr <= 12;
        end
end else if (i_barrier_cnt == 191 && vcount != 351 + delta_y) begin

o_barrier_cnt <= o_barrier_cnt + 1;
i_barrier_cnt <= 0;
barrier_mat_addr <= (vcount < 352\*delta_y) ? 0 : 12;
end else if(i_barrier_cnt[3:0] == 15) begin
barrier_mat_addr <= barrier_mat_addr + 1;
i_barrier_cnt <= i_barrier_cnt + 1;
end else begin
i_barrier_cnt <= i_barrier_cnt + 1;
end
end

```case(barrier_mat_data)
1: barrier_addr = (8'h0, o_barrier_cnt[3:0], i_barrier_cnt[3:0]);
2: barrier_addr = (8'h1, o_barrier_cnt[3:0], i_barrier_cnt[3:0]);
3: barrier_addr = (8'h2, o_barrier_cnt[3:0], i_barrier_cnt[3:0]);
4: barrier_addr = (8'h3, o_barrier_cnt[3:0], i_barrier_cnt[3:0]);
default: barrier_addr = 16'h3F0;
endcase
```

if(hcount[10:1] == 52 && vcount == 336) begin
barrier_mat_addr <= 0;
oBarrier_cnt <= 0;
i_barrier_cnt <= 0;
end
/* level*/
if (hcount[10:1]>=level_s_x && hcount[10:1]<level_e_x) && vcount>=10 && vcount<42 && hcount[0]) begin
test <=1;
if (hcount[10:1] == (level_s_x) && vcount == (10)) level_addr <= 16'h0400;
else begin
    level_addr <= level_addr + 1;
    others_addr <= level_addr;
end
end

/* level number */
if (hcount[10:1]>=level_num_s_x && hcount[10:1]<level_num_e_x && vcount>=10
&& vcount<(42) && ~hcount[0]) begin
    test <=1;
    if (hcount[10:1] == (110) && vcount == (10)) begin
        case(level)
            0: level_num_addr <= 16'h0000;
            1: level_num_addr <= 16'h0400;
            2: level_num_addr <= 16'h0800;
            3: level_num_addr <= 16'h0C00;
            4: level_num_addr <= 16'h1000;
            5: level_num_addr <= 16'h1400;
            6: level_num_addr <= 16'h1800;
            7: level_num_addr <= 16'h1C00;
            8: level_num_addr <= 16'h2000;
            9: level_num_addr <= 16'h2400;
            default: level_num_addr <= 16'h0000;
        endcase
    end
else begin
    level_num_addr <= level_num_addr + 1;
    num_addr <= level_num_addr;
end
end

/* mice */
if (hcount[10:1]>= mouse1_s_x-2 && hcount[10:1]<mouse1_e_x &&
vcount>=mouse1_s_y && vcount<mouse1_e_y && ~hcount[0] ) begin
```haskell
\[
\begin{align*}
test & <= 1; \\
others_addr & <= mouse1_addr + 16'h1400; \\
end \end{align*}
```

```
else if (hcount[10:1] >= mouse2_s_x-2 && hcount[10:1] <= mouse2_e_x && vcount>=mouse2_s_y && vcount<=mouse2_e_y && -hcount[0] ) begin
\[
\begin{align*}
test & <= 1; \\
others_addr & <= mouse2_addr + 16'h1400; \\
end \end{align*}
```

```
else if (hcount[10:1] >= mouse3_s_x-2 && hcount[10:1] <= mouse3_e_x && vcount>=mouse3_s_y && vcount<=mouse3_e_y && -hcount[0] ) begin
\[
\begin{align*}
test & <= 1; \\
others_addr & <= mouse3_addr + 16'h1400; \\
end \end{align*}
```

```
/* lives */
```

```
if (hcount[10:1] >= life1_s_x && hcount[10:1] <= life1_e_x && vcount>=10 && vcount<=42 && -hcount[0] && life>0 && life <4) begin
\[
\begin{align*}
test & <= 1; \\
if (hcount[10:1] == (life1_s_x) && vcount == (10)) life1_addr <= 16'h1000; \\
else begin \\
life1_addr <= life1_addr + 1; \\
others_addr <= life1_addr; \\
end \end{align*}
```

```
end
```

```
if (hcount[10:1] >= life2_s_x && hcount[10:1] <= life2_e_x && vcount>=10 && vcount<=42 && -hcount[0] && life>1 && life <4) begin
\[
\begin{align*}
test & <= 1; \\
if (hcount[10:1] == (life2_s_x) && vcount == (10)) life2_addr <= 16'h1000; \\
else begin \\
life2_addr <= life2_addr + 1; \\
others_addr <= life2_addr; \\
end \end{align*}
```

```
end
```
```
if (hcount[10:1] >= life3_s_x && hcount[10:1] <= life3_e_x && vcount >= 10 && vcount < 42) && ~hcount[0] && life == 3) begin
    test <= 1;
    if (hcount[10:1] == (life3_s_x) && vcount == 10) life3_addr <= 16'h1000;
    else begin
        life3_addr <= life3_addr + 1;
        others_addr <= life3_addr;
    end
end

/* score */
if (hcount[10:1] >= score_s_x && hcount[10:1] <= score_e_x && vcount >= 10 && vcount < 42) && ~hcount[0]) begin
    test <= 1;
    if (hcount[10:1] == (score_s_x) && vcount == 10) score_addr <= 16'h1480;
    else begin
        score_addr <= score_addr + 1;
        others_addr <= score_addr;
    end
end

/* score num 0*/
if (hcount[10:1] >= score_n0_s_x && hcount[10:1] <= score_n0_e_x && vcount >= 10 && vcount < 42) && ~hcount[0]) begin
    test <= 1;
    if (hcount[10:1] == (score_n0_s_x) && vcount == 10) begin
        case(score[3:0])
            0: score_n0_addr <= 16'h0000;
            1: score_n0_addr <= 16'h0400;
            2: score_n0_addr <= 16'h0800;
            3: score_n0_addr <= 16'h0C00;
            4: score_n0_addr <= 16'h1000;
        endcase
        others_addr <= score_n0_addr;
    end
5: score_n0_addr <= 16'h1400;
6: score_n0_addr <= 16'h1800;
7: score_n0_addr <= 16'h1C00;
8: score_n0_addr <= 16'h2000;
9: score_n0_addr <= 16'h2400;
default: score_n0_addr <= 16'h0000;
endcase
end
else begin
    score_n0_addr <= score_n0_addr + 1;
    num_addr <= score_n0_addr;
end
end

/* score num 1*/
if (hcount[10:1] >= score_n1_s_x && hcount[10:1] < (score_n1_e_x) && vcount == 10 && vcount < 42 && ~hcount[0]) begin
    test <= 1;
if (hcount[10:1] == (score_n1_s_x) && vcount == (10)) begin
    case(score[7:4])
        0: score_n1_addr <= 16'h0000;
        1: score_n1_addr <= 16'h0400;
        2: score_n1_addr <= 16'h0800;
        3: score_n1_addr <= 16'h0C00;
        4: score_n1_addr <= 16'h1000;
        5: score_n1_addr <= 16'h1400;
        6: score_n1_addr <= 16'h1800;
        7: score_n1_addr <= 16'h1C00;
        8: score_n1_addr <= 16'h2000;
        9: score_n1_addr <= 16'h2400;
default: score_n1_addr <= 16'h0000;
endcase
end
end
else begin
score_n1_addr <= score_n1_addr + 1;

num_addr <= score_n1_addr;
end
end

/* score num 2*/

if (hcount[10:1] >= score_n2_s_x && hcount[10:1] < score_n2_e_x && vcount >= 10 && vcount < 42 && ~hcount[0]) begin

  test <= 1;
  if (hcount[10:1] == (score_n2_s_x) && vcount == (10)) begin
  case(score[11:8])

    0: score_n2_addr <= 16'h0000;
    1: score_n2_addr <= 16'h0400;
    2: score_n2_addr <= 16'h0800;
    3: score_n2_addr <= 16'h0C00;
    4: score_n2_addr <= 16'h1000;
    5: score_n2_addr <= 16'h1400;
    6: score_n2_addr <= 16'h1800;
    7: score_n2_addr <= 16'h1C00;
    8: score_n2_addr <= 16'h2000;
    9: score_n2_addr <= 16'h2400;
    default: score_n2_addr <= 16'h0000;
  endcase
end
else begin

  score_n2_addr <= score_n2_addr + 1;
  num_addr <= score_n2_addr;
end
end

/* Start */

if (hcount[10:1] >= start_s_x && hcount[10:1] < start_e_x && vcount >= start_s_y && vcount < start_e_y && ~hcount[0] && (game_status == 2'b00)) begin
test <= 1;

if (hcount[10:1] == (start_s_x) && vcount == (start_s_y)) start_addr <= 16'h2080;
else begin
    start_addr <= start_addr + 1;
    others_addr <= start_addr;
end
end
end

/**************************************************************************
 * VGA Display combo logic
**************************************************************************/

always_comb begin
    {VGA_R, VGA_G, VGA_B} = {8'h0, 8'h0, 8'h0};

    if (VGA_BLANK_n)

        /* Start */

        if (hcount[10:1]>=start_s_x && hcount[10:1]< start_e_x && vcount>= start_s_y && vcount< start_e_y && others_data != 8'd0 && game_status == 2'b00)
            {VGA_R, VGA_G, VGA_B} = {{others_data[7:5], 5'd0}, {others_data[4:2], 5'd0}, {others_data[1:0], 6'd0}};

        /* Game Over */

        else if (hcount[10:1]>=game_over_s_x && hcount[10:1]< (game_over_e_x) && vcount>=game_over_s_y && vcount<(game_over_e_y) && game_over_data != 8'd0 &&
            game_status == 2'b10)
            {VGA_R, VGA_G, VGA_B} = {{game_over_data[7:5], 5'd0},
                               {game_over_data[4:2], 5'd0}, {game_over_data[1:0], 6'd0}};
/* Bone */

else if (hcount[10:1] >= bone_s_x && hcount[10:1] < bone_e_x && vcount >= bone_s_y && vcount < bone_e_y && bone_data != 8'd0 && dog_bullet)

    {VGA_R, VGA_G, VGA_B} = {{bone_data[7:5], 5'd0}, {bone_data[4:2], 5'd0},
    {bone_data[1:0], 6'd0}};

/* Mouse 1 */

else if (((hcount[10:1] >= mouse1_s_x) && (hcount[10:1] < mouse1_e_x) &&
(vcount >= mouse1_s_y) && (vcount < mouse1_e_y) && others_data != 8'd0) && cat_bullet[1])

    {VGA_R, VGA_G, VGA_B} = {{others_data[7:5], 5'd0}, {others_data[4:2], 5'd0},
    {others_data[1:0], 6'd0}};

/* Mouse 2 */

else if (((hcount[10:1] >= mouse2_s_x) && (hcount[10:1] < mouse2_e_x) &&
(vcount >= mouse2_s_y) && (vcount < mouse2_e_y) && others_data != 8'd0) && cat_bullet[2])

    {VGA_R, VGA_G, VGA_B} = {{others_data[7:5], 5'd0}, {others_data[4:2], 5'd0},
    {others_data[1:0], 6'd0}};

/* Mouse 3 */

else if (((hcount[10:1] >= mouse3_s_x) && (hcount[10:1] < mouse3_e_x) &&
(vcount >= mouse3_s_y) && (vcount < mouse3_e_y) && others_data != 8'd0) && cat_bullet[3])

    {VGA_R, VGA_G, VGA_B} = {{others_data[7:5], 5'd0}, {others_data[4:2], 5'd0},
    {others_data[1:0], 6'd0}};

/* Cat Matrix */

else if ((hcount[10:1] >= cat_mat_s_x) && (hcount[10:1] < cat_mat_e_x) &&
(vcount >= cat_mat_s_y) && (vcount < cat_mat_e_y) && output_data != 8'd0)

    {VGA_R, VGA_G, VGA_B} = {{output_data[7:5], 5'd0}, {output_data[4:2], 5'd0},
    {output_data[1:0], 6'd0}};

/* Mystery */

else if (hcount[10:1] >= mystery_pos_x && hcount[10:1] < (mystery_pos_x+48) &&
vcount >= mystery_pos_s_y && vcount < mystery_pos_e_y && mystery_data != 8'd0 &&
mystery_vis)

    {VGA_R, VGA_G, VGA_B} = {{mystery_data[7:5], 5'd0}, {mystery_data[4:2], 5'd0},
    {mystery_data[1:0], 6'd0}};

/* Poodle */
else if (hcount[10:1] >= dog_pos_x && hcount[10:1] < (dog_pos_x + 32) &&
vcount >= dog_pos_s_y && vcount < (dog_pos_s_y + 32) && poodle_data != 8'd0)
{
  {VGA R, VGA G, VGA B} = {{poodle_data[7:5], 5'd0}, {poodle_data[4:2], 5'd0}, {poodle_data[1:0], 6'd0}};
}

/* Explosion */
  // else if (hcount[10:1] >= 532 && hcount[10:1] < (564 && vcount) >= 400 &&
vcount < (432) && others_data != 8'd0)
  //  {VGA R, VGA G, VGA B} = {{others_data[7:5], 5'd0}, {others_data[4:2], 5'd0}, {others_data[1:0], 6'd0}};

  /* Barriers */
else if (hcount[10:1] >= 54 + delta_x && hcount[10:1] < (102 + delta_x &&
vcount) >= 336 + delta_y && vcount < (368 + delta_y) ||
          hcount[10:1] >= 154 + delta_x &&
hcount[10:1] < (202 + delta_x && vcount) >= 336 + delta_y && vcount < (368 + delta_y) ||
          hcount[10:1] >= 254 + delta_x &&
hcount[10:1] < (302 + delta_x && vcount) >= 336 + delta_y &&
vcount < (368 + delta_y) ||
          hcount[10:1] >= 354 + delta_x &&
hcount[10:1] < (402 + delta_x && vcount) >= 336 + delta_y && vcount < (368 + delta_y) &&
          barrier_data != 8'd0)
{
  {VGA R, VGA G, VGA B} = {{barrier_data[7:5], 5'd0}, {barrier_data[4:2], 5'd0}, {barrier_data[1:0], 6'd0}};
}

  /* level */
else if (hcount[10:1] >= level_s_x && hcount[10:1] < (level_e_x &&
vcount) >= 10 && vcount < (42) &&
          others_data != 8'd0)
{
  {VGA R, VGA G, VGA B} = {{others_data[7:5], 5'd0}, {others_data[4:2], 5'd0}, {others_data[1:0], 6'd0}};
}

  /* level number */
else if (hcount[10:1] >= level_num_s_x && hcount[10:1] < (level_num_e_x &&
vcount) >= 10 && vcount < (42) &&
          num_data != 8'd0)
{
  {VGA R, VGA G, VGA B} = {{num_data[7:5], 5'd0}, {num_data[4:2], 5'd0}, {num_data[1:0], 6'd0}};
}

  /* score */
else if (hcount[10:1] >= score_s_x && hcount[10:1] < (score_e_x &&
vcount) >= 10 && vcount < (42) &&
          others_data != 8'd0)
{
  {VGA R, VGA G, VGA B} = {{others_data[7:5], 5'd0}, {others_data[4:2], 5'd0}, {others_data[1:0], 6'd0}};
else if (((hcount[10:1] >= score_n0_s_x && hcount[10:1] < score_n0_e_x) ||
(hcount[10:1] >= score_n1_s_x && hcount[10:1] < score_n1_e_x) ||
(hcount[10:1] >= score_n2_s_x && hcount[10:1] < score_n2_e_x)) &&
vcount >= 10 && vcount < 42 && num_data != 8'd0)

{VGA_R, VGA_G, VGA_B} = {
(num_data[7:5], 5'd0),
(num_data[4:2], 5'd0),
(num_data[1:0], 6'd0)};

/* lives */
else if (((hcount[10:1] >= life1_s_x && hcount[10:1] < life1_e_x) && life > 0 && life < 4) ||
(hcount[10:1] >= life3_s_x && hcount[10:1] < life3_e_x) && life == 3) &&
vcount >= 10 && vcount < 42 && others_data != 8'd0)

{VGA_R, VGA_G, VGA_B} = {
(others_data[7:5], 5'd0),
(others_data[4:2], 5'd0),
(others_data[1:0], 6'd0)};

else

{VGA_R, VGA_G, VGA_B} = {background_r, background_g, background_b};
end

/**************************************************************************************************
Avalon Bus start
**************************************************************************************************/

assign eof_irq = irq;
always_ff @(posedge clk) begin

if((hcount[10:0] == 0) && (vcount[9:0] == 463)) begin
irq <= 1'b1;
frame_cnt <= frame_cnt + 1;
end else if ((hcount[10:0] >= 0) && (hcount[10:0] <= 1280) && (vcount[9:0] >= 0) &&
(vcount[9:0] < 463)) begin
frame_active <= 1;
end else begin
    frame_active <= 0;
end

if (reset) begin
    background_r <= 8'h0;
    background_g <= 8'h0;
    background_b <= 8'h0;

    irq <= 1'b0;

    play <= 0;
    play_cat <= 0;
    play_dog <= 0;
    delay <= 3;

end else if (chipselect && write) begin
    case (address)

    7'h29: begin
        c_write <= 1;
        c_cs <= 1;
        c_write_addr <= writedata[15:4];
        c_writedata <= writedata[3:0];
        end

    7'h2A: begin
        play <= writedata[5];
        vol[3:0] <= writedata[3:0];
        end

    7'h2B: begin
        play_cat <= writedata[5];
vol_cat[3:0] <= writedata[3:0];

end

7'h2C : begin
    play_dog <= writedata[5];
    vol_dog[3:0] <= writedata[3:0];
end

7'h2D : delay <= writedata;

7'h2E: begin
    b_write <= 1;
    b_cs <= 1;
    b_write_addr <= writedata[15:4];
    b_writedata <= writedata[3:0];
end

7'h2F : irq <= 0;
endcase
end else if(chipselect && !write && address == 7'h30) begin
readdata <= 16'hD065;
end else if(chipselect && !write && address == 7'h31) begin
readdata <= {frame_active, frame_cnt[14:0]};
end else begin
    b_write <= 0;
    b_cs <= 0;
    c_write <= 0;
    c_cs <= 0;
end

end

endmodule
module vga_counters(  
input logic clk50, reset,  
output logic [10:0] hcount, // hcount[10:1] is pixel column  
output logic [9:0] vcount,  // vcount[9:0] is pixel row  
output logic VGA_CLK, VGA_HS, VGA_VS, VGA_BLANK_n, VGA_SYNC_n);  

/*  
* 640 x 480 VGA timing for a 50 MHz clock: one pixel every other cycle  
*  
* HCOUNT 1599 0 1279 1599 0  
*  
* |SYNC| BP |<-- HACTIVE -->|FP|SYNC| BP |<-- HACTIVE  
*  
* |___| VGA_HS |___|  
*/  

// Parameters for hcount  
parameter HACTIVE = 11'd 1280,  
HFRONT_PORCH = 11'd 32,  
HSYNC = 11'd 192,  
HBACK_PORCH = 11'd 96,  
HTOTAL = HACTIVE + HFRONT_PORCH + HSYNC + HBACK_PORCH;  // 1600
// Parameters for vcount

`parameter` VACTIVE = 10'd 480,
VFRONT_PORCH = 10'd 10,
VSYNC = 10'd 2,
VBACK_PORCH = 10'd 33,
VTOTAL = VACTIVE + VFRONT_PORCH + VSYNC +
VBACK_PORCH; // 525

logic endOfLine;

always_ff @(posedge clk50 or posedge reset)
  begin
    if (reset) hcount <= 0;
    else if (endOfLine) hcount <= 0;
    else hcount <= hcount + 11'd 1;
  end

assign endOfLine = hcount == HTOTAL - 1;

logic endOfField;

always_ff @(posedge clk50 or posedge reset)
  begin
    if (reset) vcount <= 0;
    else if (endOfLine)
      begin
        if (endOfField) vcount <= 0;
        else vcount <= vcount + 10'd 1;
      end
    else
      begin
        if (endOfField) vcount <= 0;
        else vcount <= vcount + 10'd 1;
      end
  end

assign endOfField = vcount == VTOTAL - 1;

// Horizontal sync: from 0x520 to 0x5DF (0x57F)
// 101 0010 0000 to 101 1101 1111
assign VGA_HS = !(hcount[10:8] == 3’b101) &
                !(hcount[7:5] == 3’b111));
assign VGA_VS = !(vcount[9:1] == (VACTIVE + VFRONT_PORCH) / 2);

assign VGA_SYNC_n = 1'b0; // For putting sync on the green signal; unused

// Horizontal active: 0 to 1279       Vertical active: 0 to 479
// 101 0000 0000 1280       01 1110 0000 480
// 110 0011 1111 1599       10 0000 1100 524
                    !(vcount[9] | (vcount[8:5] == 4'b1111));

/* VGA_CLK is 25 MHz
   *
   * __ __ __ __
   * _|_ _ _ _|
   *
   * __ __ __ __
   * _hcount0_ _ _ _|
   */
assign VGA_CLK = hcount[0]; // 25 MHz clock: rising edge sensitive

endmodule

12.2 ctrl_reg.sv

module ctrl_regs(
  input logic clk,
  input logic reset,
  input logic [15:0] writedata,
  input logic write,
  input chipselect,
  input logic [7:0] address,
)

/*****************************/
Register Group #0

Dog position, and projectile position and visibility

*******************************************************************************

output logic [3:0] cat_bullet,   //cat bullet in flight?
output logic     dog_bullet,    //dog bullet in flight?
output logic [9:0] dog_pos_x,   //dog position x
output logic [9:0] dog_project_x, //dog bullet position x
output logic [9:0] dog_project_y, //dog bullet position y
output logic [9:0] cat_project_x[3:0], //cat 0~3 bullet position x
output logic [9:0] cat_project_y[3:0], //cat 0~3 bullet position y
output logic [9:0] cat_array_pos_x, //cat upper west corner position x
output logic [9:0] cat_array_pos_y, //cat upper west corner position y
output logic [9:0] mystery_pos_x, //mystery position
output logic     mystery_vis,

*******************************************************************************

Register Group #1

Miscellaneous

*******************************************************************************

output logic [3:0] level,       //game difficulty level 0~7
output logic [1:0] game_status, //game status (start, in process, game over)
output logic [1:0] life,        //lives remaining 0~3
output logic [15:0] score,      //score
output logic [3:0] dog_ani_state,
output logic     cat_ani_state

);

logic [15:0] control_regs[17:0];     //all control registers

//update values from avalon bus
genvar i, j;
generate
for(i=0; i<=17; i=i+1)begin:loop
  always_ff @(posedge clk)begin
    if(reset) control_regs[i] <= 16'd0;
    else control_regs[i] <= (write & chipselect & (address == i))? writedata : control_regs[i];
  end
end
endgenerate

//output decode & data alignment
assign cat_bullet = {control_regs[10][15], control_regs[8][15], control_regs[6][15], control_regs[4][15]};
assign dog_bullet = control_regs[2][15];
assign dog_pos_x = control_regs[0][9:0];
assign dog_project_x = control_regs[1][9:0];
assign dog_project_y = control_regs[2][9:0];
assign cat_project_x[0] = control_regs[3][9:0];
assign cat_project_y[0] = control_regs[4][9:0];
assign cat_project_x[1] = control_regs[5][9:0];
assign cat_project_y[1] = control_regs[6][9:0];
assign cat_project_x[2] = control_regs[7][9:0];
assign cat_project_y[2] = control_regs[8][9:0];
assign cat_project_x[3] = control_regs[9][9:0];
assign cat_project_y[3] = control_regs[10][9:0];
assign cat_array_pos_x = control_regs[11][9:0];
assign cat_array_pos_y = control_regs[12][9:0];
assign mystery_pos_x = control_regs[13][9:0];
assign mystery_vis = control_regs[13][15];
assign level = control_regs[14][7:4];
assign game_status = control_regs[14][3:2];
assign life = control_regs[14][1:0];
assign score = control_regs[15];
assign dog_ani_state = control_regs[16][3:0];
assign cat_ani_state = control_regs[17][0];
endmodule

12.3 audio.sv

module audio_player(
    input logic clk50, reset,
    input logic left_channel_ready,
    input logic right_channel_ready,
    input logic play,
    input logic play_cat,
    input logic play_dog,
    input logic [3:0] vol,
    input logic [3:0] vol_cat,
    input logic [3:0] vol_dog,
    input logic [3:0] delay,
    output logic left_channel_valid,
    output logic right_channel_valid,
    output logic [15:0] left_channel_data,
    output logic [15:0] right_channel_data
);

parameter MAX_BG = 17'd 34680, MAX_CAT = 15'd 6170, MAX_DOG = 15'd 4090;
```verilog
logic [3:0] counter;

//logic [3:0] slowdown;

logic[17:0] addr;
logic[17:0] cat_addr;
logic[17:0] dog_addr;

logic[15:0] data;
logic[15:0] cat_data;
logic[15:0] dog_data;

soc_system_background_music music (.address(addr), .clk(clk50), .clken(1),
.reset(reset), .reset_req(0), .readdata(data));

soc_system_ROM cat meow cat (.address(cat_addr), .clk(clk50), .clken(1),
.reset(reset), .reset_req(0), .readdata(cat_data));

soc_system_ROM dog bark dog (.address(dog_addr), .clk(clk50), .clken(1),
.reset(reset), .reset_req(0), .readdata(dog_data));

always_ff @(posedge clk50 or posedge reset) begin
    if (reset) begin
        counter <= 0;
        // slowdown <= 6;
        left_channel_data <= 15'b0;
        right_channel_data <= 15'b0;
        left_channel_valid <= 0;
        right_channel_valid <= 0;
        addr <= 16'b0;
        cat_addr <= 16'b0;
        dog_addr <= 16'b0;
    end else begin

        /*if(left_channel_ready == 1 && counter < 6250) begin
```
counter = counter + 1;
left_channel_valid <= 0;
right_channel_valid <= 0;
end else */

if(left_channel_ready == 1 && right_channel_ready == 1) begin //&& counter >= 6250) begin
  //counter <= 0;
  left_channel_valid <= 1;
  right_channel_valid <= 1;

  if(counter == delay)
    counter <= 0;
  else
    counter <= counter + 1;

  case ({play, play_cat, play_dog})
    3'b001 : begin
      left_channel_data <= dog_data;
      right_channel_data <= dog_data;

      if(counter == delay) dog_addr <= (dog_addr > MAX_DOG) ? MAX_DOG : dog_addr + 1;
    end

    3'b010 : begin
      left_channel_data <= cat_data;
      right_channel_data <= cat_data;

      if(counter == delay) cat_addr <= (cat_addr > MAX_CAT) ? MAX_CAT : cat_addr + 1;
    end

    3'b011 : begin

endcase

end
left_channel_data <= (cat_data >>> vol_cat) + (dog_data >>> vol_dog);
right_channel_data <= (cat_data >>> vol_cat) + (dog_data >>> vol_dog);

if(counter == delay) begin
  cat_addr <= (cat_addr > MAX_CAT) ? MAX_CAT : cat_addr + 1;
  dog_addr <= (dog_addr > MAX_DOG) ? MAX_DOG : dog_addr + 1;
end

3'b100 : begin
  left_channel_data <= data;
  right_channel_data <= data;

  if(counter == delay) addr <= addr + 1;
end

3'b101 : begin
  left_channel_data <= (data >>> vol) + (dog_data >>> vol_dog);
  right_channel_data <= (data >>> vol) + (dog_data >>> vol_dog);

  if(counter == delay) begin
    addr <= addr + 1;
    dog_addr <= (dog_addr > MAX_DOG) ? MAX_DOG : dog_addr + 1;
  end
end

3'b110 : begin
  left_channel_data <= (data >>> vol) + (cat_data >>> vol_cat);
  right_channel_data <= (data >>> vol) + (cat_data >>> vol_cat);
if (counter == delay) begin
    addr <= addr + 1;
    cat_addr <= (cat_addr > MAX_CAT) ? MAX_CAT : cat_addr + 1;
end
end

3'b111 : begin
    left_channel_data <= (data >> vol) + (cat_data >> vol_cat) + (dog_data >> vol_cat);
    right_channel_data <= (data >> vol) + (cat_data >> vol_cat) + (dog_data >> vol_cat);

    if (counter == delay) begin
        addr <= addr + 1;
        cat_addr <= (cat_addr > MAX_CAT) ? MAX_CAT : cat_addr + 1;
    end
end

default : begin
    addr <= 0;
    cat_addr <= 0;
    dog_addr <= 0;
end
endcase

end else begin
    left_channel_valid <= 0;
    right_channel_valid <= 0;
end

if (addr > MAX_BG || play == 0)
    addr <= 0;
if (play_cat == 0)
    cat_addr <= 0;

if (play_dog == 0)
    dog_addr <= 0;

end
end

endmodule

12.4 hello.c

#include <stdio.h>
#include <unistd.h>
#include <stdlib.h>
#include <stdbool.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <time.h>

#include "interface.h"
#include "cat_invaders.h"
#include "../controller/usbjoypad.h"

#define MIN_X 0
#define MAX_X 640 - 32
#define BONE_START_Y 416
#define BONE_END_Y 24
#define PROJ_SPEED 2
#define MYSTERY_SPEED 1
#define BONE_SPEED 3
#define MYSTERY_Y_POS 50
#define POODLE_Y_POS 430
#define MOUSE_END_Y 500
#define DOG_SPEED 2
#define CAT_SPEED 1
#define BOTTOM_CUTOFF 378
#define CAT_LEFT_WALL 1
#define CAT_RIGHT_WALL 288 // Only stores the value for the left

#define max(a, b) ((a) > (b) ? (a) : (b))
#define min(a, b) ((a) < (b) ? (a) : (b))

typedef struct {

    unsigned short x1Min;
    unsigned short x1Max;
    unsigned short y1Min;
    unsigned short y1Max;
}
unsigned short x2Min;
unsigned short x2Max;
unsigned short y2Min;
unsigned short y2Max;
)

CollisionBoxes;

unsigned short dogPos;
unsigned short mysteryPos;
Position catMatPos;
unsigned char explode_cnt[55];
unsigned char numCats;
unsigned short score = 0;
unsigned char level = 1;
unsigned char lives = 3;
unsigned short gameStatus = 0;
Position bonePos;
Position micePos[4];

void init_cat_matrix(unsigned short (*catMatrix)[55]);
void reset_cat_matrix(unsigned short (*catMatrix)[55]);
void *explode_cat(void *catNum);

/*  ------------------------ General Functions ------------------------ */
// Might need a tracker for cat movement

void *animation_tracker(void *frame) {
    unsigned short *frameNum = (unsigned short *) frame;
    *frameNum = *frameNum == 5 ? 0 : *frameNum + 1;
}

void reset_objects(unsigned short (*catMatrix)[55], int *mysterySpawnTime, bool *mysteryInPlay) {
    dogPos = 300;
    catMatPos.x = 40;
    catMatPos.y = 90;
    numCats = 55;
    *mysterySpawnTime = -1;
    *mysteryInPlay = false;
    set_ufo_pos(mysteryPos, 0);

    reset_cat_matrix(catMatrix);
    set_sprite_pos(catMatPos.x, catMatPos.y);
    set_dog_pos(dogPos);
    set_projectile_dog_position(dogPos+16, BONE_START_Y, 0);

    for (int i = 1; i < 4; i++) {
        set_projectile_sprite(i, 0, 0, 0);
    }
for(int i = 0; i < 55; i++) {
    explode_cnt[i] = 0;
}

unsigned short clamp(unsigned short val, unsigned short minV, unsigned short maxV) {
    return (unsigned short) max(min(val, maxV), minV);
}

bool barrier_hit(unsigned short (*barrierMatrix)[24], Position projPos, unsigned short projW, unsigned short projH) {
    Position barrierStart;
    barrierStart.y = 346;

    for (int j = 0; j < 2; j++) {
        for (int k = 0; k < 12; k++) {
            if (k >= 9) {
                barrierStart.x = 424;
            }
        }
    }
}
else if (k >= 6) {
    barrierStart.x = 324;
}
else if (k >= 3) {
    barrierStart.x = 224;
}
else {
    barrierStart.x = 124;
}

int barrierNum = k + 12 * j;

Position blockPos = (barrierStart.x + (16 * (k%3)),
barrierStart.y + 16 * j);

CollisionBoxes collision = (projPos.x, projPos.x + projW,
projPos.y, projPos.y + projH,
blockPos.x, blockPos.x + 16,
blockPos.y, blockPos.y + 16);

if (collision.x1Min < collision.x2Max &&
    collision.x1Max > collision.x2Min &&
    collision.y1Min < collision.y2Max &&
    collision.y1Max > collision.y2Min &&
    (*barrierMatrix)[barrierNum] != 0) {

(*barrierMatrix)[barrierNum] = ( (*barrierMatrix)[barrierNum] == 4 ) ? 0 :

(*barrierMatrix)[barrierNum] + 1;

set_barrier(barrierNum*2, (*barrierMatrix)[barrierNum]);

return true;

void reset_cat_matrix(unsigned short (*catMatrix)[55]) {
    for (int i = 0; i < 55; i++) {
        (*catMatrix)[i] = 0;
        set_sprite_matrix(i*2, 0);
    }
}

void init_cat_matrix(unsigned short (*catMatrix)[55]) {
    time_t t;
srand((unsigned) time(&t));

for (int i = 0; i < 55; i++) {
    int status = (rand() % 4) + 1;

    (*catMatrix)[i] = status;
    set_sprite_matrix(i*2, status);
}

void reset_barrier_matrix(unsigned short (*barrierMatrix)[24]) {
    for (int i = 0; i < 24; i++) {
        (*barrierMatrix)[i] = 1;
        set_barrier(i*2, 1);
    }
}

void check_lives(unsigned short (*catMatrix)[55], unsigned short (*barrierMatrix)[24], int *mysterySpawnTime, bool *mysteryInPlay) {
    if( lives == 0 ) {
        reset_cat_matrix(catMatrix);
        set_status(level, 2, lives);

        while(controller_get_state().startPressed == 0);
        while(controller_get_state().startPressed == 1);
lives = 3;
gameStatus = 0;
score = 0;

set_score(score);
level = 1;
set_status(level, gameStatus, lives);
reset_objects(catMatrix, mysterySpawnTime, mysteryInPlay);
reset_barrier_matrix(barrierMatrix);

while(controller_get_state().startPressed == 0);
while(controller_get_state().startPressed == 1);

gameStatus = 1;
set_status(level, gameStatus, lives);
init_cat_matrix(catMatrix);

void animate_player(unsigned short frame, bool dogDir, bool isWalking) {
// Frame cycles through 0,1,2:

// Assuming that left is 0-4, right is 5-9
// Walking animation

if (isWalking) {
    // Assuming that there is a 2 offset because of the two idle animation frames
    unsigned short walkFrame = 2 + frame % 3;
    set_dog_ani(dogDir ? walkFrame : walkFrame + 5);
}

// Idle animation
else {
    // NOTE: Not suse if i can use the dogDirection with the standard that i've placed but-
    unsigned short idleFrame = frame % 2;
    set_dog_ani(dogDir ? idleFrame : idleFrame + 5);
}

void move_player(ControllerState state) {
    // If the player is trying to move left, move if they are not at the boundary.
    if (state.leftArrowPressed && dogPos > MIN_X) {
        if (dogPos < DOG_SPEED) {
            dogPos = 0;
        }
    }
}
}  

else {  

dogPos = clamp(dogPos - DOG_SPEED,  
(unsigned short) MIN_X, (unsigned short) MAX_X);  
}  

set_dog_pos(dogPos);  
}  

else if (state.rightArrowPressed && dogPos < MAX_X) {  

dogPos = clamp(dogPos + DOG_SPEED,  
(unsigned short) MIN_X, (unsigned short) MAX_X);  

set_dog_pos(dogPos);  
}  

}

void update_player_state(bool *dogDir, bool *isWalking, ControllerState state) {  

if (state.leftArrowPressed) {  

*dogDir = true;  
*isWalking = true;  

return;  
}  

else if (state.rightArrowPressed) {  

*dogDir = false;  
*isWalking = true;  

return;  
}
void update_bone_projectile(ControllerState state, bool *dogInFlight) {
    // If there is no bullet currently flying and the player presses A
    if (state.buttonAPressed && !(dogInFlight)) {
        bonePos.x = dogPos + 16;
        bonePos.y = BONE_START_Y;
        set_projectile_dog_position(bonePos.x, bonePos.y, 1);
        *dogInFlight = true;
        play_bark(0);
    }
    else if (*dogInFlight) {
        if (bonePos.y < BONE_END_Y) {
            *dogInFlight = false;
            set_projectile_dog_position(dogPos, BONE_START_Y, 0);
        }
        else {
            bonePos.y -= BONE_SPEED;
            set_projectile_dog_position(bonePos.x, bonePos.y, 1);
        }
    }
}

*isWalking = false;
void bone_collision(bool *dogInFlight, bool *mysteryInPlay, int *mysterySpawnTime,
unsigned short (*catMatrix)[55], unsigned short (*barrierMatrix)[24]) {
  if (!(*dogInFlight)) {
    return;
  }

  unsigned short catNum = 0;

  // Barrier hit
  if ( barrier_hit(barrierMatrix, bonePos, 8, 24) ) {
    set_projectile_dog_position(dogPos + 16, BONE_START_Y, 0);
    *dogInFlight = false;
    return;
  }

  // Mystery hit
  else if (*mysteryInPlay) {

    CollisionBoxes mysteryCollision = {bonePos.x, bonePos.x + 8,
                                         bonePos.y, bonePos.y + 24,
                                         mysteryPos, mysteryPos + 48,
                                         mysteryPos + 40, mysteryPos + 40 + 24};
```c
if ( mysteryCollision.x1Min < mysteryCollision.x2Max &&
    mysteryCollision.x1Max > mysteryCollision.x2Min &&
    mysteryCollision.y1Min < mysteryCollision.y2Max &&
    mysteryCollision.y1Max > mysteryCollision.y2Min ) {

    *mysteryInPlay = false;
    *mysterySpawnTime = -1;
    score += 10;

    set_ufo_pos(mysteryPos, 0);
    set_projectile_dog_position(dogPos + 16, BONE_START_Y, 0);
    *dogInFlight = false;

}

// Checking cat collisions
for (int i = 0; i < 5; i++) {
    for (int j = 0; j < 11; j++) {
        catNum = 11 * i + j;
        Position catPos = {catMatPos.x + 32 * j, catMatPos.y + 32 * i};
```
CollisionBoxes collision = (bonePos.x, bonePos.x + 8,
     bonePos.y, bonePos.y + 24,
     catPos.x, catPos.x + 32,
     catPos.y, catPos.y + 32);

// Collision detected between cat

if ( collision.x1Min < collision.x2Max &&
     collision.x1Max > collision.x2Min &&
     collision.y1Min < collision.y2Max &&
     collision.y1Max > collision.y2Min &&
     (*catMatrix)[(int) catNum] != 0 ) {

    (*catMatrix)[(int) catNum] = 0;
    explode_cat(&catNum);

    play_meow(0);
    set_projectile_dog_position(dogPos + 16, BONE_START_Y, 0);
    *dogInFlight = false;

    return;
}
}
/* ------------------------------------------ Player Functions */
/* ------------------------------------------ Cat Functions */

void *explode_cat(void *catNum) {
    unsigned char *target = (unsigned char *) catNum;
    set_sprite_matrix(*target * 2, 5);
    explode_cnt[*target] = 1;

    numCats--;

    (score < 1000) ? score++ : 0;
}

void move_cats(bool *catDir) {
    // Move right

    if (*catDir && catMatPos.x < CAT_RIGHT_WALL) {
        catMatPos.x = clamp(catMatPos.x + CAT_SPEED,
                             (unsigned short) CAT_LEFT_WALL, (unsigned short) CAT_RIGHT_WALL);

        set_sprite_pos(catMatPos.x, catMatPos.y);
    }
// Move left
else if (!(*catDir) && catMatPos.x > CAT_LEFT_WALL) {
    if (catMatPos.x < CAT_SPEED) {
        catMatPos.x = 0;
    }
    else {
        catMatPos.x = clamp(catMatPos.x - CAT_SPEED,
                            (unsigned short) CAT_LEFT_WALL,
                            (unsigned short) CAT_RIGHT_WALL);
    }
    set_sprite_pos(catMatPos.x, catMatPos.y);
}

// Move down
else if ( ( *catDir && catMatPos.x == CAT_RIGHT_WALL ) ||
          ( !(*catDir) && catMatPos.x == CAT_LEFT_WALL ) ) {
    catMatPos.y += 16;
    set_sprite_pos(catMatPos.x, catMatPos.y);
    *catDir = !(*catDir);
}
void check_all_eliminated(unsigned short (*catMatrix)[55], bool *dogInFlight, bool (*catInFlight)[4], int *mysterySpawnTime, bool *mysteryInPlay) {
    if(numCats == 0) {
        (level<10) ? level++ : 0;
        numCats = 55;
        reset_objects(catMatrix, mysterySpawnTime, mysteryInPlay);
        sleep(2);
        init_cat_matrix(catMatrix);

        *dogInFlight = false;
        for (int i = 0; i < 4; i++) {
            (*catInFlight)[i] = false;
        }

        bonePos.x = dogPos + 16;
        bonePos.y = BONE_START_Y;
        set_projectile_dog_position(bonePos.x, bonePos.y, 0);

        for (int i = 1; i < 4; i++) {
            micePos[i].x = 0;
            micePos[i].y = 0;
            set_projectile_sprite(i, micePos[i].x, micePos[i].y, 0);
        }
    }
}
void animate_cats(unsigned short frame) {
    set_cat_ani(frame % 2);
}

unsigned short pick_random_cat(unsigned short catMatrix[55], unsigned short numCats) {
    unsigned short tmp[numCats];
    int counter = 0;

    for (int i = 0; i < 55; i++) {
        if (catMatrix[i] != 0) {
            tmp[counter++] = i;
        }
    }

    time_t t;
    srand((unsigned) time(&t));

    unsigned short randCat = tmp[rand() % numCats];
    return randCat;
}

int fire_cat Projectile(bool (*catInFlight)[4], unsigned short catMatrix[55], unsigned short numCats) {
// Just picks the first available projectile slot

for (int i = 1; i < 4; i++) {
    if (!(*catInFlight)[i]) {
        (*catInFlight)[i] = true;
    }
}

// Spawns the selected projectile

unsigned short randCat = pick_random_cat(catMatrix, numCats);

Position mousePos = (catMatPos.x + 32 * (randCat % 11) + 16,
                     catMatPos.y + (32 * ((int) (randCat / 11)) ) + 10);;

micePos[i].x = mousePos.x;
micePos[i].y = mousePos.y;

set_projectile_sprite(i, micePos[i].x, micePos[i].y, 1);

return i;
}

void move_cat Projectile(bool (*catInFlight)[4], int *lastShot) {
    for (int i = 1; i < 4; i++) {
        // Shoots the rest if they are in flight
if (*catInFlight)[i] && i != *lastShot) {

    if (micePos[i].y > MOUSE_END_Y) {
        (*catInFlight)[i] = false;
        set_projectile_sprite(i, micePos[i].x, micePos[i].y, 0);
    }
    else {
        micePos[i].y = micePos[i].y + PROJ_SPEED;
        set_projectile_sprite(i, micePos[i].x, micePos[i].y, 1);
    }
}

*lastShot = 5;

void mouse_collision(bool (*catInFlight)[4], unsigned short (*barrierMatrix)[24]) {

    for (int i = 1; i < 4; i++) {
        bool *currentCatInFlight = &(*catInFlight)[i];

        if (!(currentCatInFlight)) {
            continue;
        }
    }
}
// Checking dog collisions

Position completeDogPos = {dogPos, POODLE_Y_POS};

CollisionBoxes collision = {micePos[i].x, micePos[i].x + 8,
   micePos[i].y, micePos[i].y + 16,
   completeDogPos.x, completeDogPos.x + 32,
   completeDogPos.y, completeDogPos.y + 32};

// Barrier hit

if ( barrier_hit(barrierMatrix, micePos[i], 8, 16) ) {
   set_projectile_sprite(i, micePos[i].x, micePos[i].y, 0);
   (*catInFlight)[i] = false;
}

// Collision detected between dog

else if ( collision.x1Min < collision.x2Max &&
   collision.x1Max > collision.x2Min &&
   collision.y1Min < collision.y2Max &&
   collision.y1Max > collision.y2Min) {

   lives--;

   play_bark(0);
   set_projectile_sprite(i, micePos[i].x, micePos[i].y, 0);
int pick_mystery_spawn() {
    time_t t;
    srand((unsigned) time(&t));
    int offset = rand() % 6;

    if (rand() % 2 == 0) {
        offset *= -1;
    }

    return (25 + offset) * 60;
}

void spawn_mystery(bool *mysteryInPlay) {
    printf("Mystery spawned\n");
    *mysteryInPlay = true;
    mysteryPos = 640;
    set_ufo_pos(mysteryPos, 1);
}
void update_mystery(bool *mysteryInPlay, int *mysterySpawnTime) {
    if (*mysteryInPlay) {
        if (mysteryPos != 0) {
            if (mysteryPos < MYSTERY_SPEED) {
                mysteryPos = 0;
            }
            else {
                mysteryPos -= MYSTERY_SPEED;
            }
        }
        printf("Mystery Moving: %u\n", mysteryPos);
        set_ufo_pos(mysteryPos, 1);
    }
    else {
        *mysteryInPlay = false;
        *mysterySpawnTime = -1;
        printf("Mystery despawned\n");
        set_ufo_pos(mysteryPos, 0);
    }
}

void reached_bottom(unsigned short catMatrix[55]) {
    int bottomMostRow;
}
bool broke = false;

for (int i = 4; i >= 0; i--) {
    bottomMostRow = i;

    for (int j = 0; j < 11; j++) {
        int catNum = j + 11 * i;

        if (catMatrix[catNum] != 0) {
            broke = true;
            break;
        }
    }
}

if (broke) {
    break;
}

unsigned short bottomY = catMatPos.y + 32 * (bottomMostRow + 1);

if (bottomY > BOTTOM_CUTOFF) {
    lives = 0;
    printf("Cats reached the bottom, you lose.\n");
}
void start_game() {
    unsigned short catMatrix[55];
    unsigned short barrierMatrix[24];
    unsigned short tmp = 0;
    int lastShot = 5;

    unsigned short frame = 0;

    unsigned long long counter = 0;

    // true = left, false = right
    bool dogDir = true;
    bool catDir = true;

    bool isWalking = false;

    bool dogInFlight = false;
    bool catInFlight[4] = {false, false, false, false};
    bool mysteryInPlay = false;
int mysterySpawnTime = -1;

ccontroller_init();
init_driver();

reset_barrier_matrix(&barrierMatrix);
reset_objects(&catMatrix, &mysterySpawnTime, &mysteryInPlay);
set_speed(2);
play_background(0);

set_status(level, gameStatus, lives);

while(controller_get_state().startPressed == 0);
while(controller_get_state().startPressed == 1);
init_cat_matrix(&catMatrix);
gameStatus = 1;

// Update this thing

while (true) {
    // Wait
    while ( (tmp&0x7FFF) == (get_frame_reg()&0x7FFF) );
    tmp = get_frame_reg();

    if(counter % 30 == 0) {
animation_tracker(&frame);

}

counter++;

ControllerState state = controller_get_state();

update_player_state(&dogDir, &isWalking, state);

// If the spawn time hasn't been found yet, then pick one
if (mysterySpawnTime < 0) {
    mysterySpawnTime = pick_mystery_spawn();
    printf("New mystery timer: %i\n", mysterySpawnTime);
}

// 1. Check if all of the cats have been eliminated
check_all_eliminated(&catMatrix, &dogInFlight, &catInFlight, &mysterySpawnTime, &mysteryInPlay);

// 2. Check for player collisions (mouse)
bone_collision(&dogInFlight, &mysteryInPlay, &mysterySpawnTime, &catMatrix, &barrierMatrix);

// 3. Check for cat collisions (bone)
mouse_collision(&catInFlight, &barrierMatrix);

// 4. Move the cats (including animations)
// Calls move more often when there are less cats.

if( (numCats != 0) && ( counter % (5 - ( (int) ( ( 55 - numCats ) / 11 ))) == 0)) {

    move_cats(&catDir);
}

if(counter % 60 == 0) {

    // Q: i was 56, was there a reason for that? wouldn't that overflow?

    for(int i = 0; i < 55; i++) {

        if(explode_cnt[i] == 1) {

            set_sprite_matrix(i * 2, 0);
            explode_cnt[i] = 0;

        }

    }

}

animate_cats(frame);

// 5. Move the player (including animations)

move_player(state);

animate_player(frame, dogDir, isWalking); // Change the frame to the updated thing

// 6. Fire (if applicable) and move cat projectiles

if( counter % (330 - (30 * level)) == 0)
lastShot = fire_cat_projectile(&catInFlight, catMatrix, numCats);

    move_cat_projectile(&catInFlight, &lastShot);

    // 7. Fire (if applicable) and move player projectile
    update_bone_projectile(state, &dogInFlight);

    // 8. Update score
    set_score(score);

    // 9. Check if cats reached the bottom
    reached_bottom(catMatrix);

    // 10. Update status
    check_lives(&catMatrix, &barrierMatrix, &mysterySpawnTime, &mysteryInPlay);

    set_status(level, gameStatus, lives);

    // 11. Update the mystery
    if (counter % mysterySpawnTime == 0)
        spawn_mystery(&mysteryInPlay);

    update_mystery(&mysteryInPlay, &mysterySpawnTime);
int main() {

    start_game();

    return 0;
}

12.5 interface.c

;/*
 * Userspace program that communicates with the cat_invaders device driver through ioctls
 *
 * Stephen A. Edwards
 * Columbia University
 */

#include <stdio.h>
#include "cat_invaders.h"
#include "interface.h"
#include <sys/ioctl.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#include <string.h>
include <unistd.h>

int cat_invaders_fd;

struct cat_invaders_arg_t vla;

void init_driver()
{
    static const char filename[] = "/dev/cat_invaders";
    if ( (cat_invaders_fd = open(filename, O_RDWR)) == -1) {
        fprintf(stderr, "could not open %s\n", filename);
    }
    printf("driver init");
}

void set_dog_pos(unsigned short value)
{
    vla.dog_p_x = value;

    if (ioctl(cat_invaders_fd, WRITE_DOG_POSITION , &vla)) {
        perror("ioctl(WRITE_DOG_POSITION) failed");
    }
    //printf("Write DOG X_LSB: 0x%0X X_MSB: 0x%0X\n", reg_lsb, reg_msb);
void set_projectile_dog(unsigned short x, unsigned short y, unsigned short visibility){
    vla.projectile_dog.p_x = x;
    vla.projectile_dog.p_y = (y |=(visibility << 15));

    if (ioctl(cat_invaders_fd, WRITE_PROJECTILE_DOG, &vla)) {
        perror("ioctl(WRITE_PROJECTILE_DOG) failed");
    }
}

//printf("Write X_LSB: 0x%X X_MSB: 0x%X Y:0x%X\n", reg_lsb, reg_msb, y);

void set_projectile_sprite(unsigned char number, unsigned short x, unsigned short y, unsigned short visibility){

    switch (number){
        case 0: vla.projectile_cat_group.c0.x = x;
            vla.projectile_cat_group.c0.y = (y |=(visibility << 15));
            if (ioctl(cat_invaders_fd, WRITE_PROJECTILE_CAT_0, &vla)) {
                perror("ioctl(WRITE_PROJECTILE_SPRITES_0) failed");
                break;
            }

        case 1: vla.projectile_cat_group.c1.x = x;
            break;
    }
vla.projectile_cat_group.c1.y = (y |=(visibility << 15));

if (ioctl(cat_invaders_fd, WRITE_PROJECTILE_CAT_1, &vla)) {
    perror("ioctl(WRITE_PROJECTILE_SPRITES_1) failed");
    break;
}

    case 2: vla.projectile_cat_group.c2.x = x;
    vla.projectile_cat_group.c2.y = (y |=(visibility << 15));
    if (ioctl(cat_invaders_fd, WRITE_PROJECTILE_CAT_2, &vla)) {
        perror("ioctl(WRITE_PROJECTILE_SPRITES_2) failed");
        break;
}
    case 3: vla.projectile_cat_group.c3.x = x;
    vla.projectile_cat_group.c3.y = (y |=(visibility << 15));
    if (ioctl(cat_invaders_fd, WRITE_PROJECTILE_CAT_3, &vla)) {
        perror("ioctl(WRITE_PROJECTILE_SPRITES_3) failed");
        break;
    }

    default: break;
}

//printf("Write S%X: X:0x%X  Y:0x%X\n", number, *x, *y);
void set_sprite_pos(unsigned short x, unsigned short y) {
    vla.cat_matrix_pos.x = x;
    vla.cat_matrix_pos.y = y;

    if (ioctl(cat_invaders_fd, WRITE_CAT_MATRIX_POSITION, &vla)) {
        perror("ioctl(WRITE_SPRITE_POSITION) failed");

        //printf("Write SPRITE POS: X: 0x%X  Y:0x%X\n", *x, *y);
    }
}

void set_ufo_pos(unsigned short x, unsigned short visibility) {
    vla.mystery_pos_x = x;

    if (ioctl(cat_invaders_fd, WRITE_MYSTERY_POSITION, &vla)) {
        perror("ioctl(WRITE_UFO_POSITION) failed");

        //printf("Write UFO POS: X_LSB: 0x%X X_MSB: 0x%X\n", reg_lsb, reg_msb);
    }
}

void set_level(unsigned short value) {
    unsigned short status = vla.status;

    status &= 0xF0FF;
    status |= (value << 4);

    vla.status = status;
}
if (ioctl(cat_invaders_fd, WRITE_STATUS, &vla))
    perror("ioctl(WRITE_level) failed");

}

void set_lives(unsigned short value)
{
    unsigned short status = vla.status;
    status &= 0xFFFFC;
    status |= value;
    vla.status = status;
    if (ioctl(cat_invaders_fd, WRITE_STATUS, &vla))
    perror("ioctl(WRITE_lives) failed");
}

void set_gameStatus(unsigned short value)
{
    unsigned short status = vla.status;
    status &= 0xFFFF3;
    status |= value<<2;
    vla.status = status;
    if (ioctl(cat_invaders_fd, WRITE_STATUS, &vla))
    perror("ioctl(WRITE_gameStatus) failed");
}

void set_score(unsigned short score)
{
    unsigned short units = score % 10;
    unsigned short tens = (score / 10) % 10;
    unsigned short hundreds = (score / 100) % 10;
vla.score = hundreds | (tens << 4) | (units << 8);

    if (ioctl(cat_invaders_fd, WRITE_SCORE , &vla))
{perror("ioctl(WRITE_SCORE ) failed");}

    // printf("Write score: %X\n", *score);
}

void set_dog_ani(unsigned short state){

    vla.dog_ani = state;

    if (ioctl(cat_invaders_fd, WRITE_DOG_ANIMATION , &vla))
{perror("ioctl(WRITE_ANIMATION ) failed");}

    // printf("Write animation target: %X, direction: %X, state:%X\n", 
    target, *direction, *state)
}

void set_cat_ani(unsigned short state){

    vla.cat_ani = state;

    if (ioctl(cat_invaders_fd, WRITE_CAT_ANIMATION , &vla))
{perror("ioctl(WRITE_ANIMATION ) failed");}

    // printf("Write animation target: %X, direction: %X, state:%X\n", 
    target, *direction, *state)
}

void set_barrier(unsigned short number, unsigned short status){

    unsigned short barrier = (number << 3) | status;

    vla.barrier_mat = barrier;

    if (ioctl(cat_invaders_fd, WRITE_BARRIER , &vla))
{perror("ioctl(WRITE_BARRIER ) failed");}
//printf("Write Barrier: %X, state:%X\n", *number, *status);

void set_sprite_matrix(unsigned char target, unsigned short status) {
    vla.cat_mat = (target << 3) | status;

    if (ioctl(cat_invaders_fd, WRITE_CAT_MATRIX, &vla)) {
        perror("ioctl(WRITE_CAT_MATRIX) failed");
    }
}

void play_background(unsigned char volume) {
    //vla.audio.background = (0x20 | volume);
    vla.audio.background = 0xf0;

    if (ioctl(cat_invaders_fd, WRITE_BG, &vla)) {
        perror("ioctl(WRITE_BG) failed");
    }

    //printf("Play Background: %X\n", *state);
}

void play_bark(unsigned char volume) {
    vla.audio.bark = 0x00;

    if (ioctl(cat_invaders_fd, WRITE_BARK, &vla)) {
        perror("ioctl(WRITE_BARK) failed");
    }

    vla.audio.bark = (0x20 | volume);
if (ioctl(cat_invaders_fd, WRITE_BARK, &vla))
{perror("ioctl(WRITE_BARK ) failed");}

    //printf("Play Bark: %X\n", *state);
}

void play_meow(unsigned char volume){
    vla.audio.meow = 0x00;
    if (ioctl(cat_invaders_fd, WRITE_MEOW, &vla))
{perror("ioctl(WRITE_MEOW ) failed");}
    vla.audio.meow = (0x20|volume);
    if (ioctl(cat_invaders_fd, WRITE_MEOW, &vla))
{perror("ioctl(WRITE_MEOW ) failed");}
    //printf("Play Meow: %X\n", *state);
}

void set_speed(unsigned char speed){
    vla.audio.speed = speed;
    if (ioctl(cat_invaders_fd, WRITE_SPEED, &vla))
{perror("ioctl(WRITE_MEOW ) failed");}
    //printf("Play Meow: %X\n", *state);
}

void clear_irq(unsigned short irq){
    vla.irq = irq;
    if (ioctl(cat_invaders_fd, CLEAR_IRQ, &vla))
{perror("ioctl(WRITE_MEOW ) failed");}
    //printf("Play Meow: %X\n", *state);
12.6 interface.h

```c
#ifndef CAT_INVADERS_INTERFACE_H
#define CAT_INVADERS_INTERFACE_H

// #include <sys/types.h>
#include "cat_invaders.h"

typedef struct {
    unsigned short x;
    unsigned short y;
} Position;

cat_invaders_arg_t vla;

void init_driver(void);

/* set the dog's x position, y is fixed */
void set_dog_pos(unsigned short x);

/* dog bullet position x and y, idk what is the visibility used for in hw, but should be added when calling the function */
void set_projectile_dog_position(unsigned short x, unsigned short y, unsigned short visibility);

/* set the 4 projectile positions for cats. num: 0-3; position x, y */
```

void set_projectile_sprite(unsigned char number, unsigned short x, unsigned short y, unsigned short visibility);

/*set the cat matrix position (Left Top position)*/
void set_sprite_pos(unsigned short x, unsigned short y);

/*set ufo's x position, y is fixed*/
void set_ufo_pos(unsigned short x, unsigned short vis);
void set_status(unsigned short level, unsigned short gameStatus, unsigned short life);
/*set level range: 0-8*/
void set_level(unsigned short value);
/*set lives range: 0-3*/
void set_lives(unsigned short value);
/*set game status: 0: 1: 2: */
void set_gameStatus(unsigned short value);
/*set score range: 0-999*/
void set_score(unsigned short value);
void set_dog_ani(unsigned short state);
void set_cat_ani(unsigned short state);

/*To set the four barrier’s state, number 0-23 (0-5, 6-11, 12-17, 18-23 for each barrier respectively)
and status 0-4 (from full to destroyed) */
void set_barrier(unsigned short number, unsigned short status);
/*set a cat's statue, cat should be 0-54, and status:0-4 */
void set_sprite_matrix(unsigned char target, unsigned short status);

/* volume from 0 - 15*/
void play_background(unsigned char volume);
void play_bark(unsigned char volume);
void play_meow(unsigned char volume);

/* speed ranging from 0 - 6*/
void set_speed(unsigned char speed);
void set_explosion(unsigned short x, unsigned short y, unsigned short visibility);
void clear_irq(unsigned short irq);

unsigned short get_test_reg(void);
unsigned short get_frame_reg(void);

// unsigned char get_bullet(void);
// unsigned short get_dog_pos(void);
// Position get_projectile_dog(void);
// Position get_projectile_sprites(unsigned char target);
// Position get_sprite_pos(void);
// void get_sprite_matrix(void);
// unsigned short get_ufo_pos(void);
// unsigned char get_miscl(void);
// unsigned short get_score(void);
// unsigned char get_barrier(unsigned char target);
// unsigned char get_animation_dog(void);
// unsigned char get_animation_cat(void);
// unsigned char get_background(void);
// unsigned char get_bark(void);
// unsigned char get_meow(void);

#endif

12.7 cat_invaders.c

/* * Device driver for the VGA video and audio generator *
 *
 * A Platform device implemented using the misc subsystem *
 *
 * Stephen A. Edwards
 * Columbia University *
 *
 * References:
 * Linux source: Documentation/driver-model/platform.txt
 * drivers/misc/arm-charlcd.c
 * http://www.linuxforu.com/tag/linux-device-drivers/
 * http://free-electrons.com/docs/
 *
 * "make" to build
 * insmod cat_invaders.ko *
 *
 * Check code style with
/* checkpatch.pl --file --no-tree cat_invaders.c */

#include <linux/module.h>
#include <linux/init.h>
#include <linux/errno.h>
#include <linux/version.h>
#include <linux/kernel.h>
#include <linux/platform_device.h>
#include <linux/miscdevice.h>
#include <linux/slab.h>
#include <linux/io.h>
#include <linux/of.h>
#include <linux/of_address.h>
#include <linux/fs.h>
#include <linux/uaccess.h>
#include "cat_invaders.h"
#include <linux/interrupt.h>
#include <linux/ioport.h>

#define DRIVER_NAME "cat_invaders"

/* Device registers */
#define DOG_POS(x) (x)
#define PROJECT_D_X(x) ((x)+2)
#define PROJECT_D_Y(x) ((x)+4)
#define PROJECT_C0_X(x) ((x)+6)
#define PROJECT_C0_Y(x) ((x)+8)
#define PROJECT_C1_X(x) ((x)+10)
#define PROJECT_C1_Y(x) ((x)+12)
#define PROJECT_C2_X(x) ((x)+14)
#define PROJECT_C2_Y(x) ((x)+16)
#define PROJECT_C3_X(x) ((x)+18)
#define PROJECT_C3_Y(x) ((x)+20)
#define CAT_ARRAY_POS_X(x) ((x)+22)
#define CAT_ARRAY_POS_Y(x) ((x)+24)
#define MYSTERY_POS_X(x) ((x)+26)
#define STATUS(x) ((x)+28)
#define SCORE(x) ((x)+30)
#define DOG_ANI(x) ((x)+32)
#define CAT_ANI(x) ((x)+34)
#define BARRIER_MAT(x) ((x)+(46*2))
#define CAT_MAT(x) ((x)+58)
#define AUDIO_BG(x) ((x)+84)
#define AUDIO_BARK(x) ((x)+86)
#define AUDIO_MEOW(x) ((x)+88)
#define AUDIO_SPEED(x) ((x)+90)
#define EOF_IRQ(x) ((x)+94)
#define EXPLOSION_X(x) ((x) + 18*2)
#define EXPLOSION_Y(x) ((x) + 19*2)
#define UINPUT_INT_NUM 72

/*
 * Information about our device
 */

struct cat_invaders_dev {
    struct resource res; /* Resource: our registers */
    void __iomem *virtbase; /* Where registers can be accessed in memory */

    unsigned short dog_p_x;
    projectile_dog_t projectile_dog;
    projectile_cat_group_t projectile_cat_group;
    cat_matrix_position_t cat_matrix_pos;
    unsigned short mystery_pos_x;
    unsigned short status;
    unsigned short score;
    unsigned short dog_ani;
    unsigned short cat_ani;
    unsigned short barrier_mat;
    unsigned short cat_mat;
    audio_t audio;
    unsigned short irq;
    explosion_t explosion;
} dev;
/* IRQ STUFF*/

static DECLARE_WAIT_QUEUE_HEAD(interrupt_wq);
static int interrupt_flag = 0;
static DEFINE_SPINLOCK(interrupt_flag_lock);

/*functions*/

/*dog position*/
static void write_dog_pos(unsigned short *value)
{
    iowrite16(*value, DOG_POS(dev.virtbase));
    dev.dog_p_x = *value;
}

/*projectile dog*/
static void write_projectile_dog(projectile_dog_t *pos)
{
    iowrite16(pos->p_x, PROJECT_D_X(dev.virtbase));
    iowrite16(pos->p_y, PROJECT_D_Y(dev.virtbase));
    dev.projectile_dog = *pos;
}

/*projectile cats*/
static void write_projectile_cats(projectile_cat_group_t *pro_cats,
unsigned char num)
```c
switch (num) {
    case 0: iowrite16(pro_cats->c0.x, PROJECT_C0_X(dev.virtbase));
        iowrite16(pro_cats->c0.y, PROJECT_C0_Y(dev.virtbase));
        dev.projectile_cat_group.c0 = pro_cats->c0; break;
    case 1: iowrite16(pro_cats->c1.x, PROJECT_C1_X(dev.virtbase));
        iowrite16(pro_cats->c1.y, PROJECT_C1_Y(dev.virtbase));
        dev.projectile_cat_group.c1 = pro_cats->c1; break;
    case 2: iowrite16(pro_cats->c2.x, PROJECT_C2_X(dev.virtbase));
        iowrite16(pro_cats->c2.y, PROJECT_C2_Y(dev.virtbase));
        dev.projectile_cat_group.c2 = pro_cats->c2; break;
    case 3: iowrite16(pro_cats->c3.x, PROJECT_C3_X(dev.virtbase));
        iowrite16(pro_cats->c3.y, PROJECT_C3_Y(dev.virtbase));
        dev.projectile_cat_group.c3 = pro_cats->c3; break;
    default: break;
}

/*cat matrix position*/
static void write_cat_pos(cat_matrix_position_t *pos)
{
    iowrite16(pos->x, CAT_ARRAY_POS_X(dev.virtbase));
    iowrite16(pos->y, CAT_ARRAY_POS_Y(dev.virtbase));
    dev.cat_matrix_pos = *pos;
}
/*ufo position*/

static void write_ufo_pos(unsigned short *value)
{
    iowrite16(*value, MYSTERY_POS_X(dev.virtbase));
    dev.mystery_pos_x = *value;
}

/* misc 1*/

static void write_status(unsigned short *status)
{
    iowrite16(*status, STATUS(dev.virtbase));
    dev.status = *status;
}

/* score*/

static void write_score(unsigned short *score)
{
    iowrite16(*score, SCORE(dev.virtbase));
    dev.score = *score;
}

/* dog animation*/

static void write_dog_animation(unsigned short *dog_ani)
{ 
    iowrite16(*dog_ani, DOG_ANI(dev.virtbase));

    dev.dog_ani = *dog_ani;
}

/* cat animation*/

static void write_cat_animation(unsigned short *cat_ani)
{
    iowrite16(*cat_ani, CAT_ANI(dev.virtbase));

    dev.cat_ani = *cat_ani;
}

/* barrier*/

static void write_barrier(unsigned short *barrier)
{
    iowrite16(*barrier, BARRIER_MAT(dev.virtbase));

    dev.barrier_mat = *barrier;
}

/* cat matrix*/

static void write_cat_matrix(unsigned short *value)
{
    iowrite16(*value, CAT_MAT(dev.virtbase));

    dev.cat_mat = *value;
}
/ * audio*/

static void write_bg(unsigned short *value)
{
    iowrite16(*value, AUDIO_BG(dev.virtbase));
    dev.audio.background = *value;
}

static void write_bark(unsigned short *value)
{
    iowrite16(*value, AUDIO_BARK(dev.virtbase));
    dev.audio.bark = *value;
}

static void write_meow(unsigned short *value)
{
    iowrite16(*value, AUDIO_MEOW(dev.virtbase));
    dev.audio.meow = *value;
}

static void write_speed(unsigned short *value)
{
    iowrite16(*value, AUDIO_SPEED(dev.virtbase));
    dev.audio.speed = *value;
}

static void clear_irq(unsigned short *value)
{
    iowrite16(*value, EOF_IRQ(dev.virtbase));
}
dev.irq = *value;
}

static void write_explosion(explosion_t *explosion) {

    iowrite16(explosion->x, EXPLOSION_X(dev.virtbase));
    iowrite16(explosion->y, EXPLOSION_Y(dev.virtbase));
    dev.explosion = *explosion;
}

.filePath="/dev/explosion",

/*
 * Handle ioctl() calls from userspace:
 * Read or write the segments on single digits.
 * Note extensive error checking of arguments
 */

static long cat_invaders_ioctl(struct file *f, unsigned int cmd, unsigned long arg) {

    cat_invaders_arg_t vla;

    switch (cmd) {

    case WRITE_DOG_POSITION:

        if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))
            return -EACCES;

        write_dog_pos(&vla.dog_p_x);

}
break;

case WRITE_PROJECTILE_DOG:

    if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))
        return -EACCES;
    write_projectile_dog(&vla.projectile_dog);
    break;

case WRITE_PROJECTILE_CAT_0:

    if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))
        return -EACCES;
    write_projectile_cats(&vla.projectile_cat_group, 0);
    break;

case WRITE_PROJECTILE_CAT_1:

    if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))
        return -EACCES;
    write_projectile_cats(&vla.projectile_cat_group, 1);
    break;
case WRITE_PROJECTILE_CAT_2:

    if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, 
    sizeof(cat_invaders_arg_t)))

        return -EACCES;

    write_projectile_cats(&vla.projectile_cat_group, 2);

    break;

    

    

    case WRITE_PROJECTILE_CAT_3:

        if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, 
        sizeof(cat_invaders_arg_t)))

            return -EACCES;

        write_projectile_cats(&vla.projectile_cat_group, 3);

        break;

        
        
        case WRITE_CAT_MATRIX_POSITION:

            if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, 
            sizeof(cat_invaders_arg_t)))

                return -EACCES;

            write_cat_pos(&vla.cat_matrix_pos);

            break;

            
            
            case WRITE_MYSTERY_POSITION:
if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))
    return -EACCES;

write_ufo_pos(&vla.mystery_pos_x);
break;

  case WRITE_STATUS:

    if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))
        return -EACCES;

    write_status(&vla.status);
    break;

  case WRITE_SCORE:

    if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))
        return -EACCES;

    write_score(&vla.score);
    break;

  case WRITE_DOG_ANIMATION:
if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))

    return -EACCES;

    write_dog_animation(&vla.dog_ani);

    break;

case WRITE_CAT_ANIMATION:

    if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))

        return -EACCES;

        write_cat_animation(&vla.cat_ani);

        break;

case WRITE_BARRIER:

    if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))

        return -EACCES;

        write_barrier(&vla.barrier_mat);

        break;

case WRITE_CAT_MATRIX:

    if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))
return -EACCES;
write_cat_matrix(&vla.cat_mat);
break;

case WRITE_BG:

    if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))
        return -EACCES;
write_bg(&vla.audio.background);
break;

case WRITE_BARK:

    if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))
        return -EACCES;
write_bark(&vla.audio.bark);
break;

case WRITE_MEOW:

    if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))
        return -EACCES;
write_meow(&vla.audio.meow);
break;

case WRITE_SPEED:

    if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))
        return -EACCES;

    write_speed(&vla.audio.speed);

    break;

case CLEAR_IRQ:

    if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))
        return -EACCES;

    clear_irq(&vla.irq);

    break;

case WRITE_EXPLOSION:

    if (copy_from_user(&vla, (cat_invaders_arg_t *) arg, sizeof(cat_invaders_arg_t)))
        return -EACCES;

    write_explosion(&vla.explosion);

    break;

default:

    return -EINVAL;

}
static const struct file_operations cat_invaders_fops = {
    .owner = THIS_MODULE,
    .unlocked_ioctl = cat_invaders_ioctl,
};

/* Information about our device for the "misc" framework -- like a char dev */
static struct miscdevice cat_invaders_misc_device = {
    .minor = MISC_DYNAMIC_MINOR,
    .name = DRIVER_NAME,
    .fops = &cat_invaders_fops,
};

static int __init cat_invaders_probe(struct platform_device *pdev)
{
    //cat_invaders_color_t beige = { 0xf9, 0xe4, 0xb7 };
    //audio_t audio_begin = { 0x00, 0x00, 0x00 };
    int ret;
/* Register ourselves as a misc device: creates /dev/cat_invaders */

ret = misc_register(&cat_invaders_misc_device);

/* Get the address of our registers from the device tree */

ret = of_address_to_resource(pdev->dev.of_node, 0, &dev.res);
if (ret) {
    ret = -ENOENT;
    goto out_deregister;
}

/* Make sure we can use these registers */

if (request_mem_region(dev.res.start, resource_size(&dev.res),
        DRIVER_NAME) == NULL) {
    ret = -EBUSY;
    goto out_deregister;
}

/* Arrange access to our registers */

dev.virtbase = of_iomap(pdev->dev.of_node, 0);
if (dev.virtbase == NULL) {
    ret = -ENOMEM;
    goto out_release_mem_region;
}
return 0;

cat_invaders_remove:
    release_mem_region(dev.res.start, resource_size(&dev.res));

cat_invaders_remove:
    misc_deregister(&cat_invaders_misc_device);
    return ret;
}

/* Clean-up code: release resources */
static int cat_invaders_remove(struct platform_device *pdev)
{
    iounmap(dev.virtbase);
    release_mem_region(dev.res.start, resource_size(&dev.res));
    misc_deregister(&cat_invaders_misc_device);
    return 0;
}

/* Which "compatible" string(s) to search for in the Device Tree */
#ifdef CONFIG_OF
static const struct of_device_id cat_invaders_of_match[] = {
    { .compatible = "csee4840,cat_invaders-1.0" },
    {},
};
#endif
static struct platform_driver cat_invaders_driver = {
    .driver = {
        .name = DRIVER_NAME,
        .owner = THIS_MODULE,
        .of_match_table = of_match_ptr(cat_invaders_of_match),
    },
    .remove = __exit_p(cat_invaders_remove),
};

static int __init cat_invaders_init(void)
{
    pr_info(DRIVER_NAME": init\n");
    return platform_driver_probe(&cat_invaders_driver,
cat_invaders_probe);
}

static void __exit cat_invaders_exit(void)
{
    platform_driver_unregister(&cat_invaders_driver);
}
pr_info(DRIVER_NAME " : exit\n");

module_init(cat_invaders_init);
module_exit(cat_invaders_exit);

MODULE_LICENSE("GPL");
MODULE_AUTHOR("Stephen A. Edwards, Columbia University");
MODULE_DESCRIPTION("cat_invaders driver");

12.8 cat_invaders.h

#ifndef _CAT_INVADERS_H
#define _CAT_INVADERS_H

#include <linux/ioctl.h>

/*projectile_dog*/
typedef struct{
   unsigned short p_x;
   unsigned short p_y;
}projectile_dog_t;

/*projectile_cat_sprite*/
typedef struct {

typedef struct {
    projectile_cat_t c0;
    projectile_cat_t c1;
    projectile_cat_t c2;
    projectile_cat_t c3;
} projectile_cat_group_t;

typedef struct {
    unsigned short x;
    unsigned short y;
} explosion_t;

/*cat matrix position*/
typedef struct {
    unsigned short x;
    unsigned short y;
} cat_matrix_position_t;

/*score*/
typedef struct{
    unsigned short score_lsb;
} score_t;
unsigned short score_msb;
}

/*animation*/
typedef struct {
    unsigned short dog_ani;
    unsigned short cat_ani;
} animation_t;

/*audio*/
typedef struct {
    unsigned short background;
    unsigned short bark;
    unsigned short meow;
    unsigned short speed;
} audio_t;

/*Argument*/
typedef struct {
    unsigned short dog_p_x;
    projectile_dog_t projectile_dog;
    projectile_cat_group_t projectile_cat_group;
    explosion_t explosion;
    cat_matrix_position_t cat_matrix_pos;
    unsigned short mystery_pos_x;
    unsigned short status;
unsigned short score;
unsigned short dog_ani;
unsigned short cat_ani;
unsigned short barrier_mat;
unsigned short cat_mat;

audio_t audio;
unsigned short irq;
unsigned short test_reg;
unsigned short frame_reg;

} cat_invaders_arg_t;

#define CAT_INVADERS_MAGIC 'q'

/* ioctls and their arguments */

// #define WRITE_MISC_0 _IOW(CAT_INVADERS_MAGIC, 1, cat_invaders_arg_t *)
#define WRITE_DOG_POSITION _IOW(CAT_INVADERS_MAGIC, 1, cat_invaders_arg_t *)
#define WRITE_PROJECTILE_DOG _IOW(CAT_INVADERS_MAGIC, 2, cat_invaders_arg_t *)
#define WRITE_PROJECTILE_CAT_0 _IOW(CAT_INVADERS_MAGIC, 3, cat_invaders_arg_t *)
#define WRITE_PROJECTILE_CAT_1 _IOW(CAT_INVADERS_MAGIC, 4, cat_invaders_arg_t *)
#define WRITE_PROJECTILE_CAT_2 _IOW(CAT_INVADERS_MAGIC, 5, cat_invaders_arg_t *)
#define WRITE_PROJECTILE_CAT_3 _IOW(CAT_INVADERS_MAGIC, 6, cat_invaders_arg_t *)
#define WRITE_CAT_MATRIX_POSITION _IOW(CAT_INVADERS_MAGIC, 7, cat_invaders_arg_t *)
#define WRITE_MYSTERY_POSITION _IOW(CAT_INVADERS_MAGIC, 8, cat_invaders_arg_t *)
#define WRITE_STATUS _IOW(CAT_INVADERS_MAGIC, 9, cat_invaders_arg_t *)
#define WRITE_SCORE _IOW(CAT_INVADERS_MAGIC, 10, cat_invaders_arg_t *)
#define WRITE_BARRIER _IOW(CAT_INVADERS_MAGIC, 11, cat_invaders_arg_t *)

#define WRITE_CAT_MATRIX _IOW(CAT_INVADERS_MAGIC, 12, cat_invaders_arg_t *)

#define WRITEDOG_ANIMATION _IOW(CAT_INVADERS_MAGIC, 13, cat_invaders_arg_t *)

#define WRITE_CAT_ANIMATION _IOW(CAT_INVADERS_MAGIC, 14, cat_invaders_arg_t *)

#define WRITE_BG _IOW(CAT_INVADERS_MAGIC, 15, cat_invaders_arg_t *)

#define WRITE_BARK _IOW(CAT_INVADERS_MAGIC, 16, cat_invaders_arg_t *)

#define WRITE_MEOW _IOW(CAT_INVADERS_MAGIC, 17, cat_invaders_arg_t *)

#define WRITE_SPEED _IOW(CAT_INVADERS_MAGIC, 18, cat_invaders_arg_t *)

#define CLEAR_IRQ _IOW(CAT_INVADERS_MAGIC, 19, cat_invaders_arg_t *)

#define WRITE_EXPLOSION _IOW(CAT_INVADERS_MAGIC, 20, cat_invaders_arg_t*)

#define READ_TEST_REG _IOR(CAT_INVADERS_MAGIC, 21, cat_invaders_arg_t *)

#define READ_FRAME_REG _IOR(CAT_INVADERS_MAGIC, 22, cat_invaders_arg_t*)

// #define READ_MISC_0 _IOR(CAT_INVADERS_MAGIC, 18, cat_invaders_arg_t *)

// #define READ_DOG_POSITION _IOR(CAT_INVADERS_MAGIC, 19, cat_invaders_arg_t *)

// #define READ_PROJECTILE_DOG _IOR(CAT_INVADERS_MAGIC, 20, cat_invaders_arg_t *)

// #define READ_PROJECTILE_SPRITES _IOR(CAT_INVADERS_MAGIC, 24, cat_invaders_arg_t *)

// #define READ_SPRITE_POSITION _IOR(CAT_INVADERS_MAGIC, 25, cat_invaders_arg_t *)

// #define READ_SPRITE_MATRIX _IOR(CAT_INVADERS_MAGIC, 26, cat_invaders_arg_t *)

// #define READ_UFO_POSITION _IOR(CAT_INVADERS_MAGIC, 27, cat_invaders_arg_t *)

// #define READ_MISC_1 _IOR(CAT_INVADERS_MAGIC, 28, cat_invaders_arg_t *)

// #define READ_SCORE _IOR(CAT_INVADERS_MAGIC, 29, cat_invaders_arg_t *)

// #define READ_BARRIER _IOR(CAT_INVADERS_MAGIC, 30, cat_invaders_arg_t *)

// #define READ_ANIMATION _IOR(CAT_INVADERS_MAGIC, 31, cat_invaders_arg_t *)

// #define READ_BG _IOR(CAT_INVADERS_MAGIC, 32, cat_invaders_arg_t *)
// #define READ_BARK_IOR(CAT_INVADERS_MAGIC, 33, cat_invaders_arg_t *)
// #define READ_MEOW_IOR(CAT_INVADERS_MAGIC, 34, cat_invaders_arg_t *)

#endif

12.9 usbjoypad.c

#include "usbjoypad.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <arpa/inet.h>
#include <unistd.h>
#include <pthread.h>

/* References on libusb 1.0 and the USB HID/joypad protocol *
 *
* http://libusb.org
 *
 *
 *
* https://usb.org/sites/default/files/hut1_5.pdf
 */
/*
 * Find and return a USB joypad device or NULL if not found
 * The argument con
 *
 */

struct libusb_device_handle *openjoypad(uint8_t *endpoint_address) {

    libusb_device **devs;
    struct libusb_device_handle *joypad = NULL;
    struct libusb_device_descriptor desc;
    ssize_t num_devs, d;
    uint8_t i, k;

    /* Start the library */
    if (libusb_init(NULL) < 0) {

        fprintf(stderr, "Error: libusb_init failed\n");

        exit(1);
    }

    /* Enumerate all the attached USB devices */
    if ( (num_devs = libusb_get_device_list(NULL, &devs)) < 0 ) {

        fprintf(stderr, "Error: libusb_get_device_list failed\n");

        exit(1);
    }

    /* Look at each device, remembering the first HID device that speaks
     * the joypad protocol */

for (d = 0 ; d < num_devs ; d++) {

    libusb_device *dev = devs[d];

    if ( libusb_get_device_descriptor(dev, &desc) < 0 ) {

        fprintf(stderr, "Error: libusb_get_device_descriptor failed\n");

        exit(1);
    }

    if (desc.bDeviceClass == LIBUSB_CLASS_PER_INTERFACE) {

        struct libusb_config_descriptor *config;

        libusb_get_config_descriptor(dev, 0, &config);

        for (i = 0 ; i < config->bNumInterfaces ; i++)

            for (k = 0 ; k < config->interface[i].num_altsetting ; k++) {

                const struct libusb_interface_descriptor *inter =

                    config->interface[i].altsetting + k ;

                if ( inter->bInterfaceClass == LIBUSB_CLASS_HID &&

                    inter->bInterfaceProtocol == USB_HID_joypad_PROTOCOL) {

                    int r;

                    if ( ((r = libusb_open(dev, &joypad)) != 0) )

                        fprintf(stderr, "Error: libusb_open failed: %d\n", r);

                    exit(1);
                }

                if (libusb_kernel_driver_active(joypad,i))

                    libusb_detach_kernel_driver(joypad, i);

                libusb_set_auto_detach_kernel_driver(joypad, i);

                if ( ((r = libusb_claim_interface(joypad, i)) != 0) )

                    fprintf(stderr, "Error: libusb_claim_interface failed: %d\n", r);
exit(1);

*endpoint_address = inter->endpoint[0].bEndpointAddress;

goto found;

}

}

}

}

}

}

found:

libusb_free_device_list(devs, 1);

return joypad;

}

struct libusb_device_handle *joypad;

uint8_t endpoint_address;

volatile ControllerState state = {false, false, false, false};

pthread_mutex_t stateMutex;


void* controller_update_thread(void* arg) {

    struct usb_joypad_packet packet;

    int transferred;

    while (1) {


int result = libusb_interrupt_transfer(joypad, endpoint_address, (unsigned char*)&packet, sizeof(packet), &transferred, 0);

if (result == 0 && transferred == sizeof(packet)) {
    pthread_mutex_lock(&stateMutex);
    state.leftArrowPressed = (packet.keycode[1] == 0x00);
    state.rightArrowPressed = (packet.keycode[1] == 0xFF);
    state.buttonAPressed = (packet.keycode[3] == 0x2F);
    state.startPressed = (packet.keycode[4] == 0x20);
    pthread_mutex_unlock(&stateMutex);
} else {
    continue;
}

usleep(100000);

return NULL;

void controller_init() {

    pthread_t threadId;

    pthread_mutex_init(&stateMutex, NULL);

    if ( (joypad = openjoypad(&endpoint_address)) == NULL ) {
        fprintf(stderr, "Did not find a joypad\n");
        exit(1);
    }
}
ControllerState controller_get_state() {

    ControllerState currentState;

    pthread_mutex_lock(&stateMutex);
    currentState = state;
    pthread_mutex_unlock(&stateMutex);

    return currentState;
}

12.10 usbjoypad.h

#ifndef _USBjoypad_H
#define _USBjoypad_H

#include <libusb-1.0/libusb.h>
#include <stdbool.h>

#define USB_HID_joypad_PROTOCOL 0

struct usb_joypad_packet {
    uint8_t modifiers;
    uint8_t reserved;
};
typedef struct {
    bool leftArrowPressed;
    bool rightArrowPressed;
    bool buttonAPressed;
    bool startPressed;
} ControllerState;

void controller_init();

void controller_update();

ControllerState controller_get_state();

/* Find and open a USB joypad device. Argument should point to
   space to store an endpoint address. Returns NULL if no joypad
   device was found. */

extern struct libusb_device_handle *openjoypad(uint8_t *);
## Appendix B: Platform Design

<table>
<thead>
<tr>
<th>Use</th>
<th>Connections</th>
<th>Name</th>
<th>Description</th>
<th>Export</th>
<th>Clock</th>
<th>Base</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>clk 0</td>
<td>Clock Source</td>
<td>clk</td>
<td>exported</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk_in</td>
<td>Clock Input</td>
<td>reset</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk_reset</td>
<td>Reset Input</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk_out</td>
<td>Clock Output</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk_reset</td>
<td>Reset Output</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>hps_0</td>
<td>Arria V Cylcone V Hard Proce...</td>
<td>hps_0_h2_</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>memory</td>
<td>Memory</td>
<td>hps</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>hps_io</td>
<td>I/O Port</td>
<td>hps</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>hps_reset</td>
<td>Reset</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>hps_aw_clock</td>
<td>Clock</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>hps_aw_master</td>
<td>Master</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>t2h_aw_clock</td>
<td>Clock</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>t2h_aw_slave</td>
<td>Slave</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>h2f_aw_clock</td>
<td>Clock</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>h2f_aw_master</td>
<td>Master</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>f2h_intp</td>
<td>Interrupt</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>f2h_irq</td>
<td>IRQ</td>
<td>clock</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>cat_invaders_0</td>
<td>Cat Invaders</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reset</td>
<td>Reset</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>vga</td>
<td>VGA</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>vga_out</td>
<td>Buffer</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>audio_0</td>
<td>Audio</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>audio_out</td>
<td>Output</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>audio_in</td>
<td>Input</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>audio_pll_0</td>
<td>Clock</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>audio_pll_1</td>
<td>Clock</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>audio_pll_2</td>
<td>Clock</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>background_mem</td>
<td>Memory</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk1</td>
<td>Clock</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>s1</td>
<td>Avalon Memory Mapped Slave</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reset1</td>
<td>Reset</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ROM_cat_1</td>
<td>Clock</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>clk1</td>
<td>Clock</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>s1</td>
<td>Avalon Memory Mapped Slave</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>reset1</td>
<td>Reset</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>ROM_cat_2</td>
<td>Clock</td>
<td>clk</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Diagram:**

- The diagram illustrates various components and connections within the platform design, including clock sources, interrupt lines, and memory mapped slaves.

**Notes:**

- Clock sources and inputs are labeled with appropriate connections.
- Exported signals are indicated with a double-click to label.
- Clock distribution is shown withclk andreset signals.
- Memory mapped slaves (e.g., Avalon) are connected to specific components.
Appendix C: The Screen Layout