

The Design Document 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	15
7	VGA Output	17
8	Game Controller	18
9	Milestones	21
10	References	22

1 Introduction

This design document will describe the hardware and software details for implementing FPGA Cat Invaders, a game based on the classic Space Invader arcade game (Reference [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.

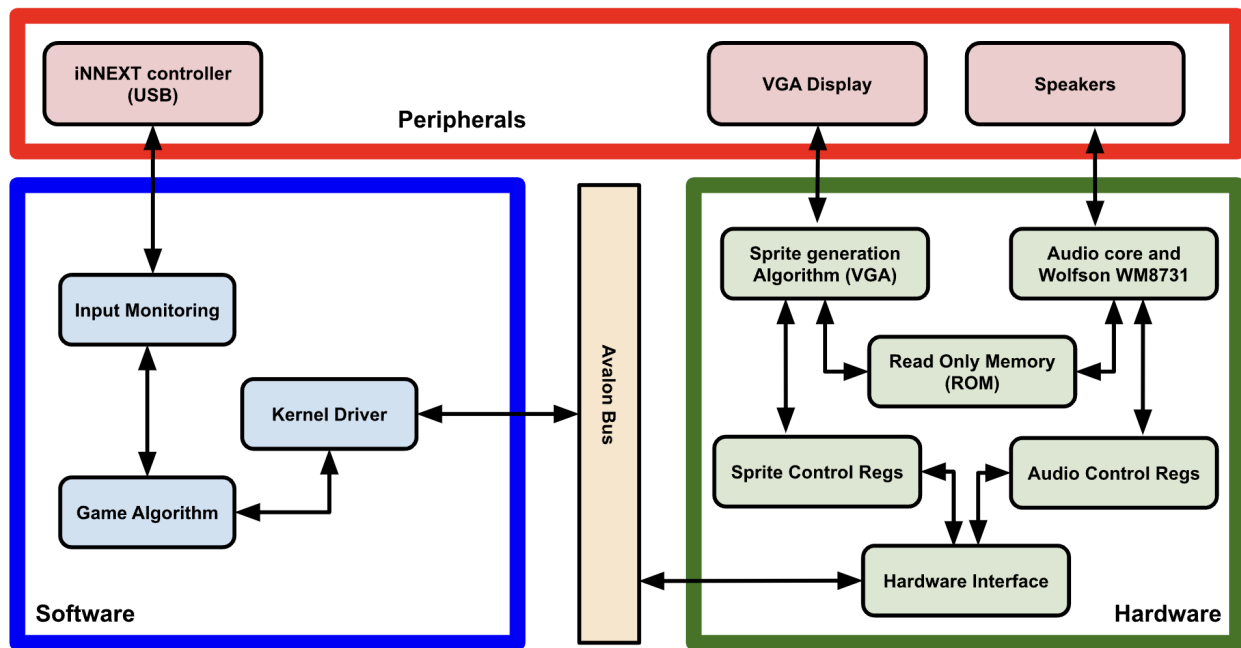
The game is played through a combination of hardware, software, and peripherals. The player uses an iNNEXT controller (Figure 9) to control the movement of the poodle and launch the femur bones. The game is displayed through a VGA monitor and the audio is played through mono speakers. The game algorithm runs in software, monitoring inputs from the iNNEXT controller and communicating with the FPGA hardware registers through the Avalon Bus (Reference [7]). The FPGA hardware is responsible for outputting the video to the VGA monitor and the audio to the mono speakers.

This design document 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.

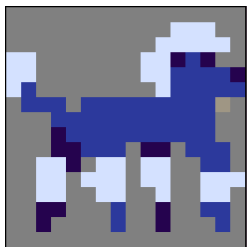
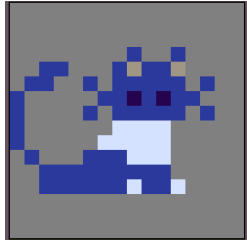

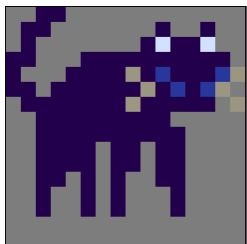

Figure 1: Block Diagram describing the software and hardware at a high level.


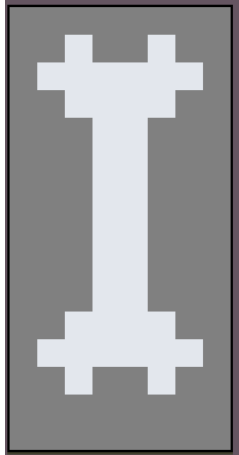
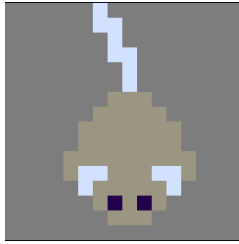
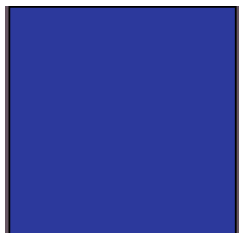
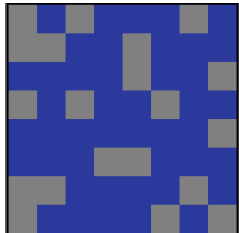


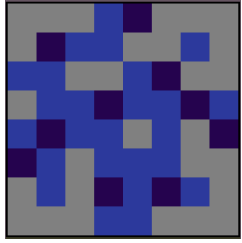
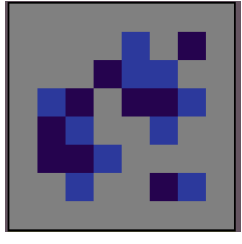
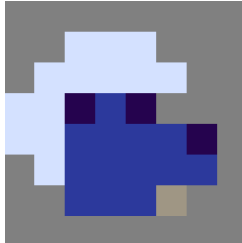
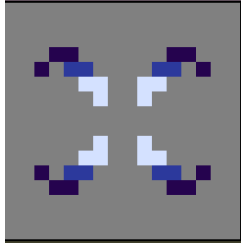



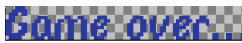
3 Resource Budget


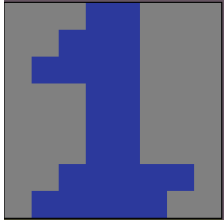

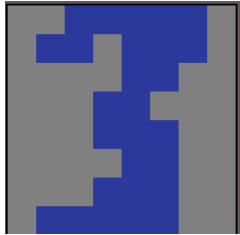

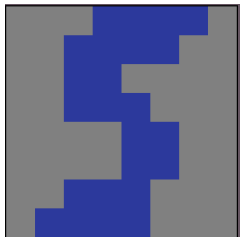
The DE1-SoC has 64MB of BRAM on the DE1_SoC FPGA [3]. See Table 1 and 2 below for the sprites and audio being included in this project and the memory required for each. The sprites and audio will be stored in the FPGA read-only memory (ROM) in mif format (Reference [8]).

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

Name	Sprite	Size (bits)	Color (bits)	Total Size (bits)
Poodle		16 x 16	24	6144
Cat 1		16 x 16	24	6144
Cat 2		16 x 16	24	6144
Cat 3		16 x 16	24	6144
Cat 4		16 x 16	24	6144

Mystery		24 x 24	24	13824
Projectile (femur bone)		8 x 16	24	3072
Projectile (mouse)		16 x 16	24	6144
Barriers - solid		8 x 8	24	1536
Barrier - damaged 1		8 x 8	24	1536

Barrier - damaged 2		8 x 8	24	1536
Barrier - damaged 3		8 x 8	24	1536
Lives		8 x 8	24	1536
Explosion		8 x 8	24	1536
Level (text)		64 x 8	24	12288
Score (text)		64 x 8	24	12288
Start (text)		64 x 8	24	12288
Game Over (text)		240 x 32	24	184320

0 (text)		8 x 8	24	1536
1 (text)		8 x 8	24	1536
2 (text)		8 x 8	24	1536
3 (text)		8 x 8	24	1536
4 (text)		8 x 8	24	1536
5 (text)		8 x 8	24	1536

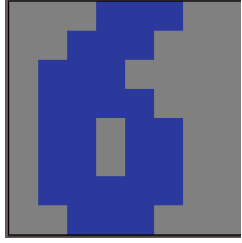
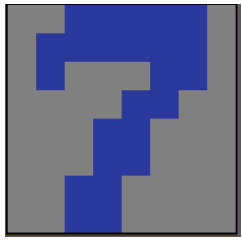
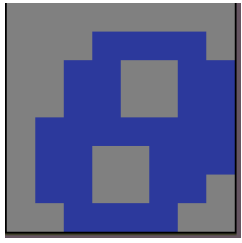
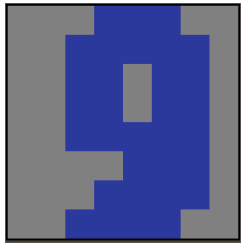
6 (text)		8 x 8	24	1536
7 (text)		8 x 8	24	1536
8 (text)		8 x 8	24	1536
9 (text)		8 x 8	24	1536
Total Size (bits) (Bytes)				299420 (36 KB)

Table 2: Table organizing memory required per audio and the total amount of memory required.

Name	Size (bits)	Total Size (bits)
Bark	8000 x 16	128000
Meow	8000 x 16	128000
Background music	2560000	2560000
Total Size (bits) (Bytes)		2585600 (0.34 MB)

4 Algorithms

Enemies Function Descriptions:

spawn() - The cats will spawn in a fixed position at the start of the game and on advancing a level.

move() - The cats move from left to right and when they hit the edge of the screen, they move down one unit and change directions. The speed of the movement is dependent on the level and cats will move faster the higher the level.

update(int level) - Updates and increases / sets the cats' movement speed and rate of fire according to the current level to set the difficulty.

fire() - One of the remaining cats will shoot a projectile (a mouse) at the dog periodically with frequency depending on the level. The cats will shoot more frequently as the player advances levels.

checkCollisions() - If the pixels of a bone collide with the pixels of a cat, then a collision has occurred and the cat will meow and despawn and be freed from memory.

check bottom () - If any of the cats reach the bottom of the screen where the poodle is, then a game-over occurs.

eliminated () - If all of the cats are eliminated, then the player advances one level.

Players Function Descriptions:

spawn() - The poodle will spawn in a fixed position at the start of the game and on advancing a level.

move() - The player can move the poodle left or right using the controller's left joystick. If the joystick is positioned right, then all of the poodle's pixels will move right, and if the joystick is positioned left, all of the poodle's pixels are moved to the left. If the joystick is neutral, the poodle will not move. The player can only move the poodle to the bounds of the screen and cannot move past either the left or right boundaries.

fire() - When the player hits the A button, a bone is spawned at the top of the poodle and will move in a straight line until it either collides with a cat or barrier or moves off-screen. The player will not be able to shoot until after a timer expires.

checkCollisions() - If a mouse's pixels and the poodle's pixels overlap, then the player will lose a life. The player will be briefly intangible (not lose lives on collision) both as a way to avoid losing multiple lives on a single collision and to give the player time to reposition.

GameOver() - When the player loses all of their lives, a game over will occur, resetting the player back to the first level.

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

```
C/C++
Game {
    Cat cats[][]; // All of the cats that are currently on screen.
    Poodle player; // The poodle that the player moves.
    int level = 1; // The current level that the player is on.
    int lives = 3; // The number of lives that the player currently has.

    cats.update(level);
    cats.spawn();
    player.spawn();

    while (true) {
        // Updating the cats
        cats.move();
        cats.fire();
        cats.checkCollisions();
        if (cats.checkBottom()) {
            lives -= 1;
        }
        // If all of the cats are eliminated, then the level advances and the scene resets.
        if (cats.allEliminated()) {
            level += 1;
            cats.update(level);
            cats.spawn();
            player.spawn();
        }

        // Updating the player.
        player.move();
        player.fire();
        if (player.checkCollisions()) {
            lives -= 1;
        }
        if (lives <= 0) {
            player.gameOver();
        }
    }
}
```

5 The Hardware/Software Interface

The hardware and software will communicate bidirectionally in 1 byte (8bit) chunks over the Avalon Bus. Tables 3-7 below describe the registers the software and hardware can read and write to. Table 8 describes what each field in the audio registers represent.

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

Register #	Register Name	R/W	Bit description							
			7	6	5	4	3	2	1	0
0	misc_0	R/W	Reserved	Cat 3 bullet	Cat 2 bullet	Cat 1 bullet	Cat 0 bullet	Dog bullet	Dog Pos X LSB	
1	dog_pos_msb	R/W	Dog Pos X MSB							
2	project_d_x_lsb	R/W	Reserved						Projectile dog X LSB	
3	project_d_x_msb	R/W	Projectile dog X MSB							
4	project_d_y	R/W	Projectile dog Y MSB							
5	project_s0_x	R/W	Projectile sprite0 X MSB							
6	project_s0_y	R/W	Projectile sprite0 Y MSB							
7	project_s1_x	R/W	Projectile sprite1 X MSB							
8	project_s1_y	R/W	Projectile sprite1 Y MSB							
9	project_s2_x	R/W	Projectile sprite2 X MSB							
10	project_s2_y	R/W	Projectile sprite2 Y MSB							
11	project_s3_x	R/W	Projectile sprite3 X MSB							
12	project_s3_y	R/W	Projectile sprite3 Y MSB							

Table 4: Register category - Cat position and visibility, and UFO position

Register #	Register Name	R/W	Bit description							
			7	6	5	4	3	2	1	0
13	sprite_pos_x	R/W	Cat Array Pos X MSB							
14	sprite_pos_y	R/W	Cat Array Pos Y MSB							
15	sprite_matrix[0]	R/W	Cat visible 0 to 7							
16	sprite_matrix[1]	R/W	Cat visible 8 to 15							
17	sprite_matrix[2]	R/W	Cat visible 16 to 23							
18	sprite_matrix[3]	R/W	Cat visible 24 to 31							
19	sprite_matrix[4]	R/W	Cat visible 32 to 39							
20	sprite_matrix[5]	R/W	Cat visible 40 to 47							
21	sprite_matrix[6]	R/W	Reser ved	Cat visible 48 to 54						
22	ufo_pos_x_msb	R/W	UFO Position X MSB							
23	ufo_pos_x_lsb	R/W	Reserved				UFO Position X LSB			

Table 5: Register category - Miscellaneous

Register #	Register Name	R/W	Bit description		
			7..4	3..2	1..0
24	misc_1	R/W	Level	Game Status	Life
25	score_lsb	R/W	Score LSB		
26	score_msb	R/W	Score MSB		

Table 6: Register category - Barriers

Register #	Register Name	R/W	Bit description			
			7..6	5..4	3..2	1..0
27	barrier[0]	R/W	bar0[3]	bar0[2]	bar0[1]	bar0[0]
28	barrier[1]	R/W	bar0[7]	bar0[6]	bar0[5]	bar0[4]
29	barrier[2]	R/W	bar0[11]	bar0[10]	bar0[9]	bar0[8]
30	barrier[3]	R/W	bar1[3]	bar1[2]	bar1[1]	bar1[0]
31	barrier[4]	R/W	bar1[7]	bar1[6]	bar1[5]	bar1[4]
32	barrier[5]	R/W	bar1[11]	bar1[10]	bar1[9]	bar1[8]
33	barrier[6]	R/W	bar2[3]	bar2[2]	bar2[1]	bar2[0]
34	barrier[7]	R/W	bar2[7]	bar2[6]	bar2[5]	bar2[4]
35	barrier[8]	R/W	bar2[11]	bar2[10]	bar2[9]	bar2[8]
36	barrier[9]	R/W	bar3[3]	bar3[2]	bar3[1]	bar3[0]
37	barrier[10]	R/W	bar3[7]	bar3[6]	bar3[5]	bar3[4]
38	barrier[11]	R/W	bar3[11]	bar3[10]	bar3[9]	bar3[8]

Table 7: Audio register map

Register #	Register Name	R/W	Bit description			
			7..3	2	1	0
39	Background Music	R/W	Reserved	Play	Loop	Status
40	Bark	R/W	Reserved	Play	Loop	Status
41	Meow	R/W	Reserved	Play	Loop	Status

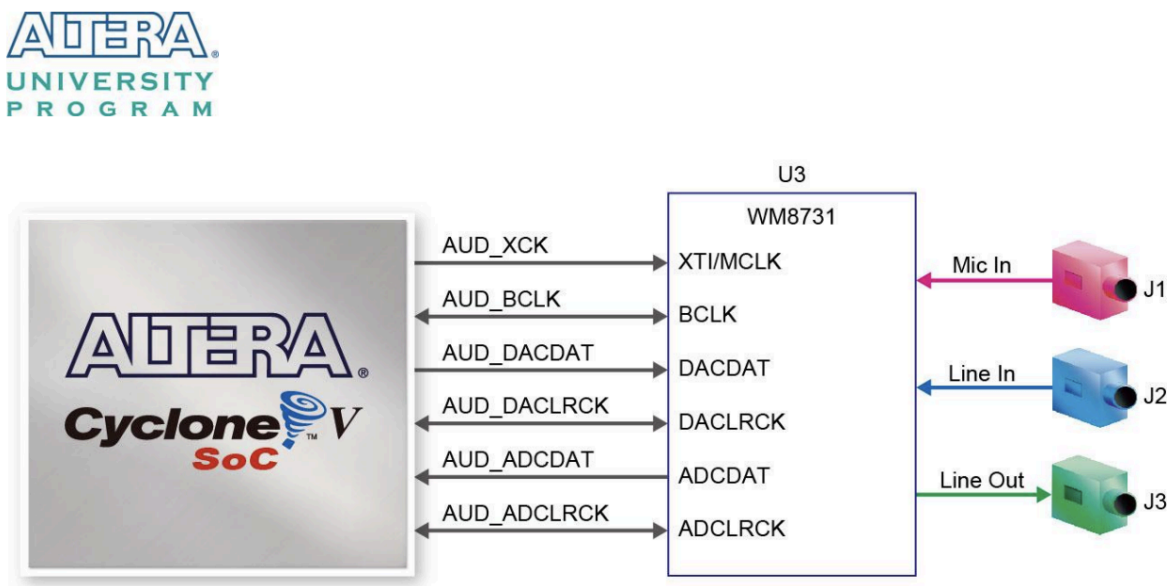
Table 8: Background Music, Bark, and Meow register definition

Bit #	Name	R/W	Description
7..3	Reserved	R/W	Unused bits
2	Play	W	Used to start/stop the audio. Writing a 1 will play the audio clip, and writing a 0 will stop the audio.
1	Loop	W	Used to tell the hardware to start back at the beginning once the end is detected. Writing a 1 will cause the audio clip to loop, and writing a 0 will cause the audio to stop once the end is reached
0	Status	R	Used to detect if the audio is currently playing. Reading a 1 means the audio clip is in the process of playing, reading a 0 means the audio clip is not playing.

6 Audio Interface

The FPGA Cat Invader games will output background music and sound effects. The DE1-SoC contains a Wolfson WM8731 audio CODEC (Reference [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 Reference [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, for this project only the left audio output (left FIFO) will be used. The IP requires synchronization between the left and right output FIFO, to avoid issues with this synchronization the right side FIFO buffers will be set to

0. 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.

Figure 4: Block diagram for Audio and Video core (image source Reference [6]).

AUDIO/VIDEO CONFIGURATION CORE FOR DE-SERIES BOARDS

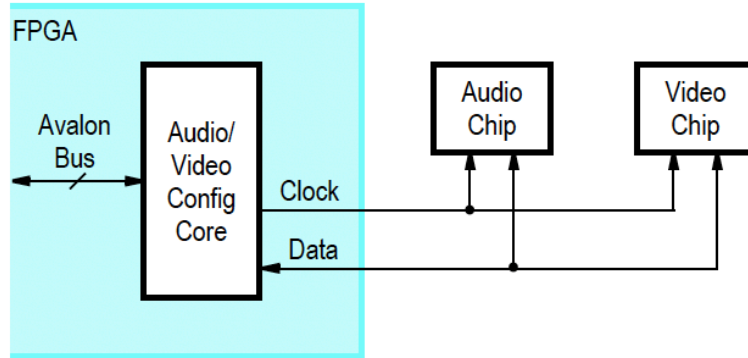
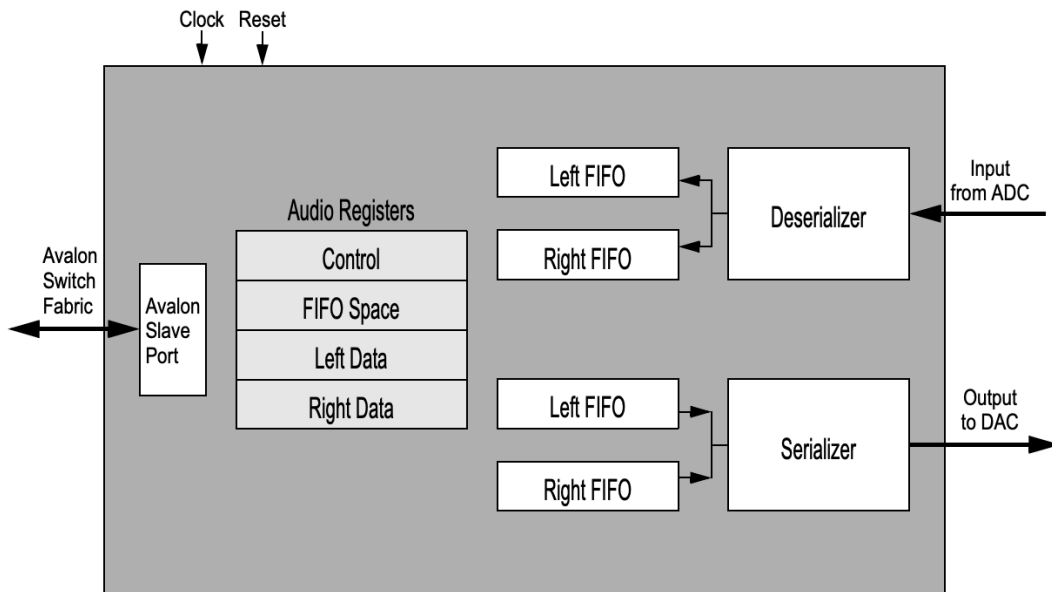


Figure 5: Block diagram for Audio core with Memory-Mapped Interface (image source Reference [5]).

AUDIO CORE FOR INTEL® DE-SERIES BOARDS

For Quartus® Prime 18.0



To facilitate multiple audio clips played together simultaneously the audio will first be divided by a power of two based on the number of clips being played at the same time. For example, if two clips are being played together both clips will be divided by 2 (shifted right one time). If

three clips are being played together all clips will be divided by 4 (shifted right two times). After reducing the amplitude the clips will be added together and placed in the audio core's left 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 sample at 8kHz. 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.

7 VGA Output

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 6 shows the signals connected between the FPGA and ADV7123 VGA DAC (Reference [3]).

Figure 6: Connections between the FPGA and VGA(image source Reference [3]).

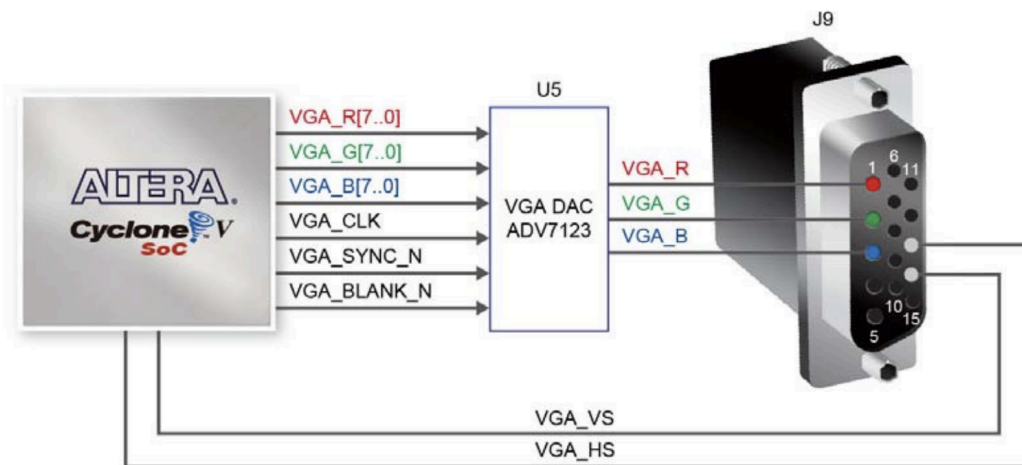
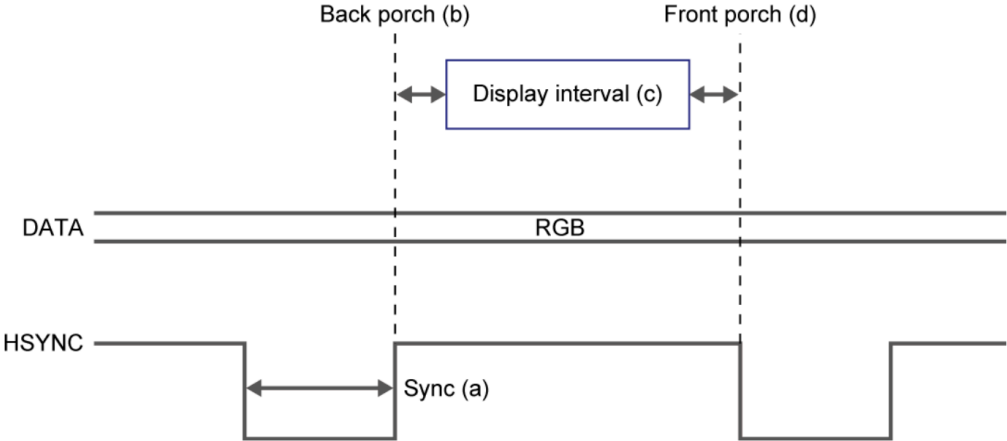


Figure 7 outlines the key timing for displaying a row on a VGA screen. An active-low hsync pulse indicates the end of one row and the start of the next. After this pulse, the RGB data must be off during the back porch, then on during the display interval as pixels are activated across the row. The RGB 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 (Reference [3]).

Figure 7: VGA horizontal timing specification(image source Reference [3]).



8 Game Controller

The player will interact with the game using a iNNEXT joypad, which is shown in Figure 8. The controller sends and receives over the usb connection and the software uses the libusb library to communicate.

Figure 8: Image of iNNEXT Joypad.



The buttons on the joypad and their corresponding functions are shown in Table 9. 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 9: Buttons and corresponding functions.

Button	Function
Left arrow	Move Left
Right arrow	Move right
Start	Start the game
A	Attack

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 10.

Table 10: Communication parameters of the device over libusb.

idvendor (VID)	0x0079
idProduct (PID)	0x0011
Packet size	1*8 bytes

The full detail of the communication protocol is demonstrated in the Table 11 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 11: Communication protocol packet definition of button presses over libusb.

Byte	0	1	2	3	4	5	6	7
default	01	7F	7F	7F	0F	00	002	202
A					2F			
B					4F			
X					1F			
Y					8F			
Up arrow				00				
Down arrow				FF				
Left arrow			00					
Right arrow			FF					
L2						01		
R2						02		
Start						20		
Select						10		
Left arrow + A			00		2F			
Right arrow +A			FF		2F			

9 Milestones

Milestone 1

The first milestone will be to get the sprites generated on screen at positions that software can command via the Avalon Bus.

Milestone 2

The second milestone will be to get the sound clips to play out of the speakers where the software can start/stop commands via the Avalon Bus.

Milestone 3

The third milestone will be to implement the gameplay software algorithm. In addition, the algorithm will process the iNNEXT controller commands to control the poodle and move the cats on the screen.

10 References

- [1] Space Invaders. (2024, March 19). Wikipedia. https://en.wikipedia.org/wiki/Space_Invaders
- [2] Project, F. V. G. (n.d.). Free Invaders. FreeInvaders.org.
https://freeinvaders.org/?_ga=2.126103371.754523267.1708803339-1764679121.1708803339
- [3] Terasic Technologies Inc. (2014). DE1-SoC_User_manual_v.1.2.2. In <https://www.terasic.com.tw>. Retrieved March 26, 2024, from https://www.terasic.com.tw/cgi-bin/page/archive_download.pl?Language=English&No=836&FID=ae336c1d5103cac046279ed1568a8bc3
- [4] Wolfson microelectronics, “WM8731 / WM8731L Portable Internet Audio CODEC with Headphone Driver and Programmable Sample Rates”, Rev 3.4, April 2004, from <https://www.cs.columbia.edu/~sedwards/classes/2008/4840/Wolfson-WM8731-audio-CODEC.pdf>
- [5] Intel FPGA University Program Audio core for DE-Series Boards, Intel Corporation, June 2018, from https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/18.1/University_Program_IP_Cores/Audio_Video/Audio.pdf
- [6] Altera Audio/Video Configuration Core for DE-Series Boards, Altera Corporation, July 2010, from https://ftp.intel.com/Public/Pub/fpgaup/pub/Intel_Material/9.1/University_Program_IP_Cores/Audio_Video/Audio_and_Video_Config.pdf
- [7] Avalon Memory-Mapped Interface Specification, Altera Corporation, 101 Innovation Drive, San Jose, CA 95134, May 2007, from https://www.cs.columbia.edu/~sedwards/classes/2008/4840/mnl_avalon_spec.pdf
- [8] Memory initialization file (.MIF) definition. Intel. (n.d.-b).https://www.intel.com/content/www/us/en/programmable/quartushelp/17.0/reference/glossary/def_mif.htm