University of Toronto
Faculty of Applied Science and Engineering
ECE532
Final Design Report

Date April 13, 2017

Project Title Motion Control Raiden Game

Prepared By Sheng Zhan
Yi Tian Lu
Zhe Wu

Table of Content

1. Overview

1.1 Background and Motivation

Since 1990s, video game had became part of our lives for many years. Some of the classical games were never being forgotten. The team members are interested in video games and very excited to build such a game on our own. Our team decides to develop and reproduce one of the classical scrolling shooter game, Raiden, on the Xilinx Nexys 4 DDR FPGA Board, with real-time motion detection as the controllers. We would like to incorporate the techniques learned from this course, such as video processing, movements tracking and bus protocols between digital modules

Raiden is one of the most classical scrolling game. **Figure 1** is a flyer of the original Raiden game. Players can control the Raiden aircraft to dodge enemy bullets and destroy enemy's aircraft. Enemy aircrafts will spawn and move randomly, and player will gain scores if the enemy aircrafts are destroyed. The mission will fail if the player’s aircraft is hit by the enemy.

In this project, our team will develop some more features on top of the basic plays. First of all, player will be able to control the movement of the aircraft by moving a physical object (such as a color block). Also, different difficulty levels and rolling background will be implemented to strengthen the playability. As the game enters different stages, audio effects and animation will also be generated. In addition, the game can support multiple players (maximum of 2). **Figure 2** is a brief illustration of the game mechanism.

Fig. 1. Japanese arcade flyer of Raiden

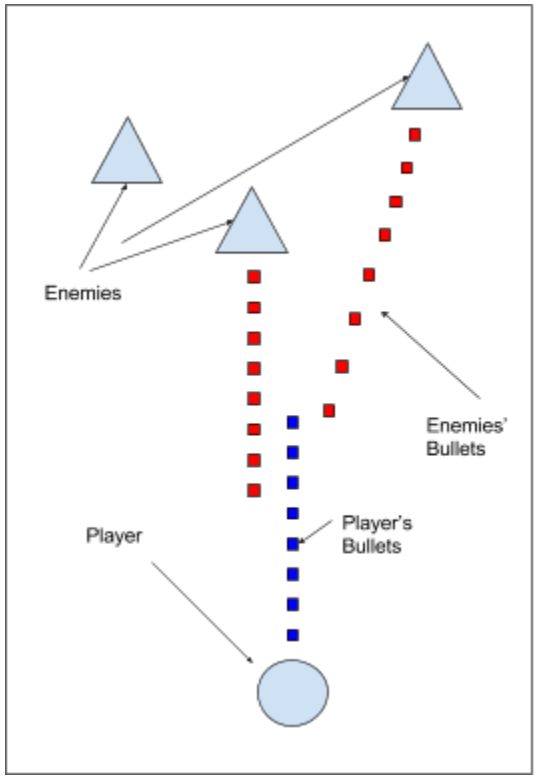

Fig. 2. Illustration of the game

1.2 Goals

The table below provides a summary of the functionalities and features, as well as the acceptance criterias which describes how it will be satisfied for each item.

Functional Requirements / Features	Acceptance criterias
Player aircraft is controlled by real-time motion	The player's aircraft should follow the movement of the physical object captured by the motion detector and be able to move in different directions with certain angles. In addition, the latency of the game should be less than 200ms, and ideally 100ms.
Enemies spawn and move randomly	At least two enemy aircrafts should be generated at random locations, and they should be directed to move in different paths. The aim is over 10 enemies.

Player aircrafts and enemy aircrafts are eliminated by collision with each others' bullets	The enemy should disappear when it is overlapped with player's bullet, and player's aircraft also should disappear when it is overlapped with enemy's bullet.
Player will gain scores when the enemy aircraft is eliminated	The score should be incremented by the value of the eliminated enemy. Different enemies should have different values.
Mission fails when the player's aircraft is hit by the enemy	The game should be terminated and display a "GAME OVER" figure, and shows the final score.
Background music is played when game starts	The background music should be played when the game starts and different BGM should be played with each difficulty levels.
The speed and quantity of the enemy unit can vary with difficulty levels	For every difficulty level, the movement speed of the enemy's aircrafts and bullets should increase by 20%.
The background of the video output will be scrolling.	A long picture should scroll downwards as background.
Two players can play at the same time (optional)	Two player aircrafts can be controlled respectively by each player moving distinct colour block.
Sound effects are generated when enemies is eliminated (optional)	A short explosion sound should be played when the enemy is overlapped with player's bullet.

1.3 Block Diagram

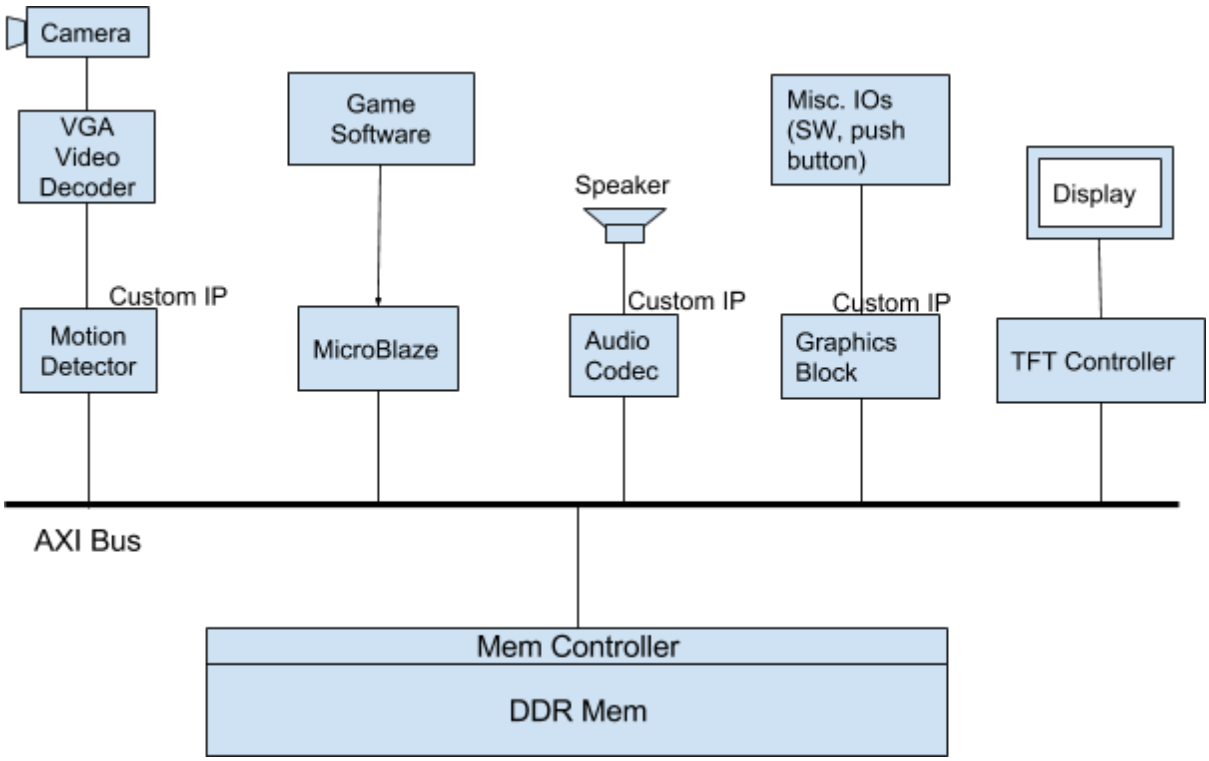

Fig. 3 System block diagram

1.4 Brief Description of IP

IP Block	Function	Origin
Video Decoder (OV7670)	Convert into digital video signals	Existing IP
Motion Detector	Read video frames, process and identify the controller location, then provide a location data stored in register	Custom IP
MicroBlaze	Processor core	Existing IP
Audio Codec	Convert digital stream into audio	Custom IP
TFT Controller	Read video frames from DDR memory to display	Existing IP

	on VGA	
Memory Controller	Regulates R/W memory operations	Existing IP
Game Software	The game features will be written in this software block. It reads the player unit location from the Motion Detector, calculate the enemy unit location, ballistic trajectory and collision between units and bullets. Then store the calculated data into memory.	Custom IP
Graphics Block	Accept drawing request from game software, read image data from local BRAMs and write to video buffer in DDR memory.	Custom IP

2. Outcome

2.1 Results

All the planned functional requirements are met. Below is a list of the requirements from Section 1.2.

Functional requirements and Features

1. Player aircraft is controlled by real-time motion (Done)
2. Enemies spawn and move randomly (Done)
3. Player aircrafts and enemy aircrafts are eliminated by collision with each others' bullets (Done)
4. Player will gain scores when the enemy aircraft is eliminated (Done)
5. Mission fails when the player's aircraft is hit by the enemy (Done)
6. Background music is played when game starts (Done)
7. The speed and quantity of the enemy unit can vary with difficulty levels (Done)
8. The background of the video output will be scrolling (Done)
9. Two players can play at the same time (Extra)
10. Sound effects are generated when enemies is eliminated (Extra)

2.2 Future work

All the features defined in proposal have been implemented properly and the game is pretty playable. The next step would be adding more cool features and make the game more exciting.

A. For example, we can add more motion detection to allow bullet direction control.
B. We can let graphics block support reading from DDR memory to draw large objects and a real delicate background.
C. We can enhance the audio quality and play some real music.

3. Project Schedule

The original plan from project proposal was vague, while the actual schedule is more detailed and more measurable. Three team members were assigned individual tasks and deadlines. Therefore the project was progressing gradually and was work with all features in the end. Below is a table of comparison between original plan and actual work.

Milestones	**Original Plan**	**Actual Work**
Milestone #1	Finish testbenches for camera, video decoder, VGA, audio codec blocks	Infrastructure built (Microblaze, AXI interface, DDR memory) Basic function done for camera input, decoder, VGA(TFT block). Basic testing also finished - able to draw through memory writes.
Milestone #2	Testbench demo. Finish all testbenches and start design.	Graphics block slave side done, start working on master write channel. Audio block: Created a custom AXI slave IP block, able to generate 1 second of a tone with 7 different frequencies.

		Detection block: Developed the average filtering algorithm in verilog, and created a simple test bench to verify the output.
Milestone #3	Finish camera, video decoder, VGA blocks, generate a bgm using the audio block.	Graphics block write address channel done, able to draw color blocks. Audio block: The AXI slave is able to take the song selection from the game software and play the song. Detection block: Simulated the created test bench using real image input, and probed the camera input using debug probe.
Milestone #4	Finish detector block and make software interact with FPGA. Upgrade the Audio IP to generate background music compatible with sound effect.	Graphics block write data channel done, able to draw images. Audio block: Successfully Implemented the sound effect (played by writing to reg1) with bgm (played by writing to reg0) Detection block: Connected the camera and verify the detection block can keep track of the colored object. Improved the accuracy.
Milestone #5	Mid-Project Demo. Most features should be working, but may associate with some bugs.	Do hardware integration and write software.
Milestone #6	Debugging and deal with possible performance issue.	Finish game software. Game is ready to play.
Milestone #7	Final Demo Done! Everything should be working	Polish the design. Ready for demo.

4. Description of the Blocks

4.1 Motion Detector Block

The motion detector block consists of three sub modules. The detailed description for each sub modules are provided in the following sections.

4.1.1 pmod_input

This module is in charge of converting the OV7670 camera input into pixel data stream. The pixel data stream is then processed and calculated by the position locator block. This module is modified based on the source code provided in 532_pmod_camera_dist.zip from ECE532 Piazza. The pixel datas are written into the FIFO in arriving order. When vsync signal is raised, a special constant will be written into the FIFO to indicate the last pixel of a frame is arrived.

4.1.2 FIFO

The FIFO module buffers the pixel data stream written from pmod camera. Then the buffer data will be read by the position locator block to calculate the object's positions in a frame. The FIFO module has two different clock domains for read and write. The write clock is 25MHz generated by camera, whereas the read clock is 100MHz system clock.

4.1.3 Position Detector

The purpose of position detector is to calculate and determine the objects' positions in a frame by reading the pixel datas from the FIFO. The position detector is able to keep tracks of two different colors: blue and red. This is achieved by implementing the average filtering algorithm to find the center locations of both blue and red colours. Two different RGB threshold configurations are used for both colors to determine if the pixel is the desired color or not. The values of average coordinates is stored in the AXI Slave registers.

The position detector module has the following states:

FSM states	Description
WAITING	● Moves to COMPUTING state if there is any new pixels available in the FIFO, otherwise stays in WAITING state
COMPUTING	● If the input data is real pixel data, checks if the pixel is within the RGB threshold range for both blue and red colors. If yes, updates the accumulated X,Y values and number of counted pixels accordingly. Current X and Y positions are also updated. Remains in COMPUTING state if there are datas to read. ● If the input data is not real pixel data but an indication of end of a frame, moves to CREATING_OUTPUT state
CREATING_OUTPUT	● Divides the accumulated X, Y values by the number of counted pixels to calculate the center locations for both blue and red colors. When the division is done, moves to OUTPUT_READY state
OUTPUT_READY	● Copies X,Y values of the center location into AXI Slave registers. Moves to RESETTING state
RESETTING	● Resets all the values to 0. Moves to WAITING state

The next diagram shows the datapath of position locator:

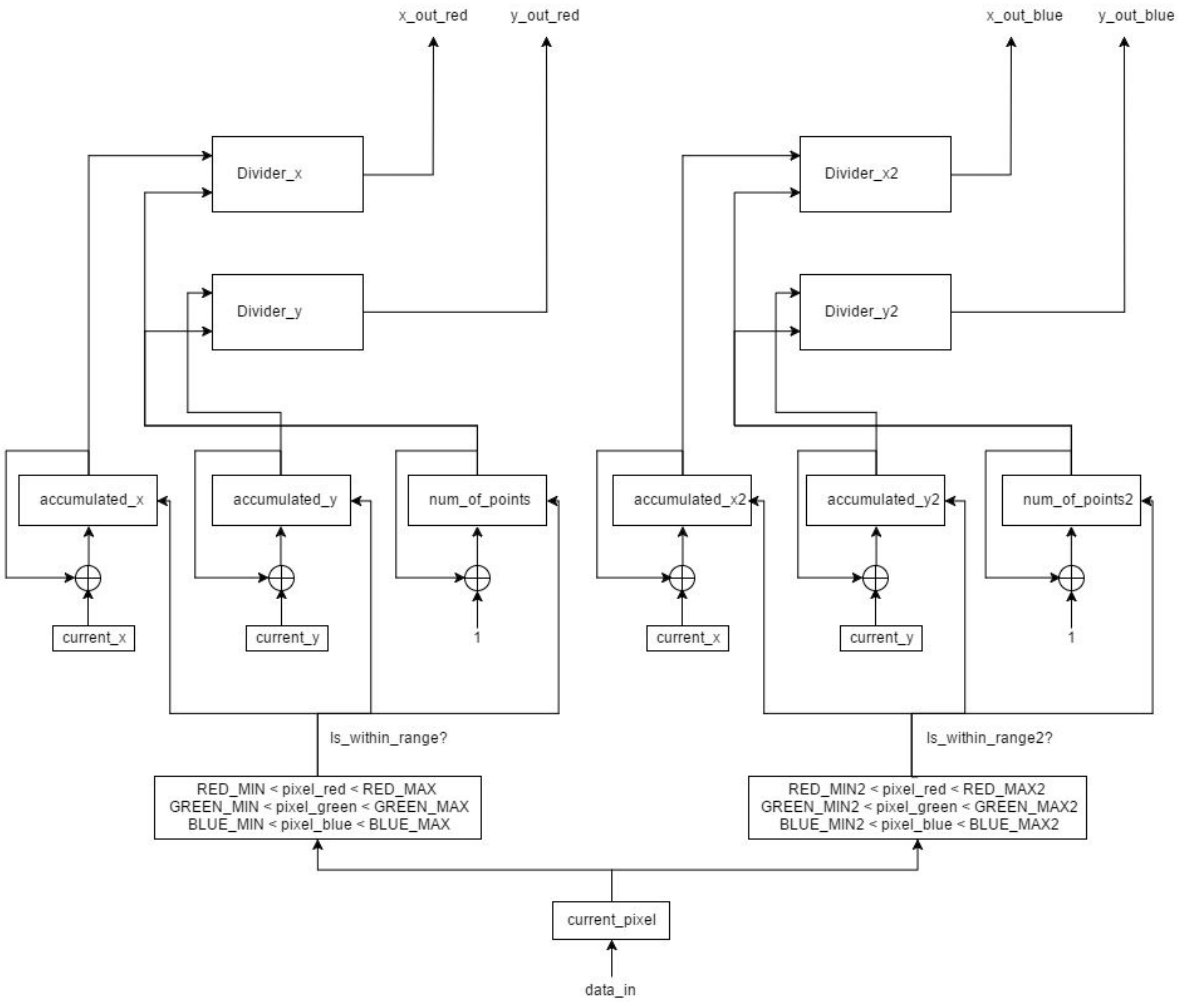

Fig. 4 Position Locator Datapath

In total, three 32-bit AXI slave registers are used, these registers are accessible by microblaze processor using AXI protocol.

Register Name	Offset	Description
Blue_output	0x0	Read Only. It stores X,Y coordinates of the blue object ● [4:0] - Unused ● [15:5] - X coordinate of the blue object ● [20:16] - Unused ● [30:21] - Y coordinate of the blue object ● [31] - Ready bit. This bit is used to indicate the validity of the register value
Red_output	0x8	Read Only. It stores X,Y coordinates of the red object ● [4:0] - Unused ● [15:5] - X coordinate of the red object ● [20:16] - Unused

		● [30:21] - Y coordinate of the red object ● [31] - Ready bit. This bit is used to indicate the validity of the register value
Blue_threshold	0x4	This register is used for debugging purpose. This register can be written to configure the RGB threshold values to determine if a pixel is blue or not. ● [3:0] - Blue minimum value ● [7:4] - Blue maximum value ● [11:8] - Green minimum value ● [15:12] - Green maximum value ● [19:16] - Red minimum value ● [23:20] - Red maximum value ● [31:24] - Unused

4.2 Graphics Block

The Graphics IP Block consists of an AXI slave and an AXI master. The Fig. 5 below illustrates its structure.

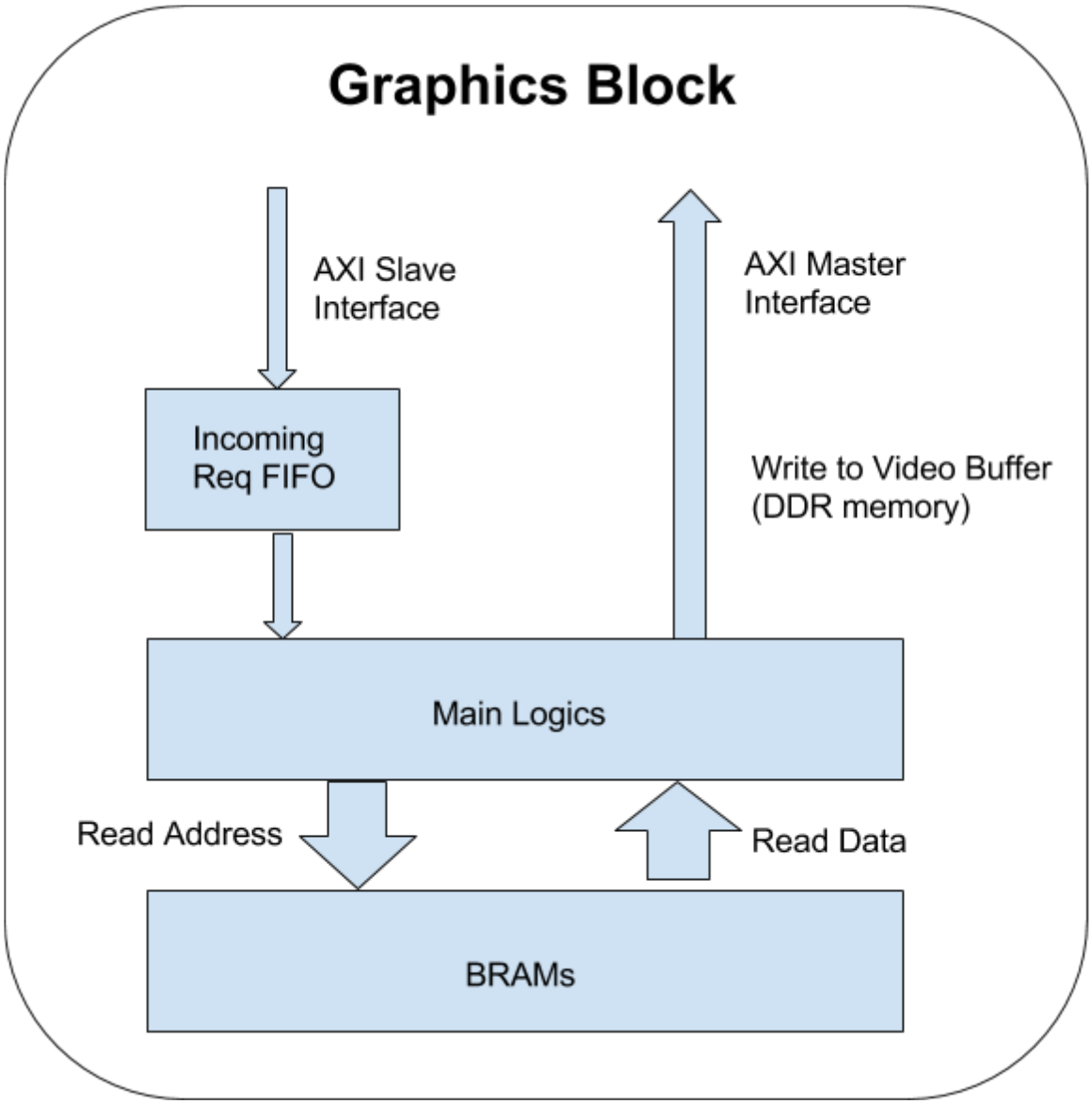

Fig. 5 Graphics Block Diagram

The memory mapped slave receives drawing request from Microblaze processor and stores into a FIFO. Then the requests are processed in sequence to calculate the drawing image information and the drawing location on screen. Image data is read from local BRAMs to minimize latency. Finally the block implements AXI control signals, such as address valid, data valid, and burst size, and writes the data to video buffer in DDR memory through AXI master interface.

To test this block, we first ran system simulation to model some drawing requests and compare with the expected results. Then we download to FPGA and visually check if the images are drawn on screen correctly.

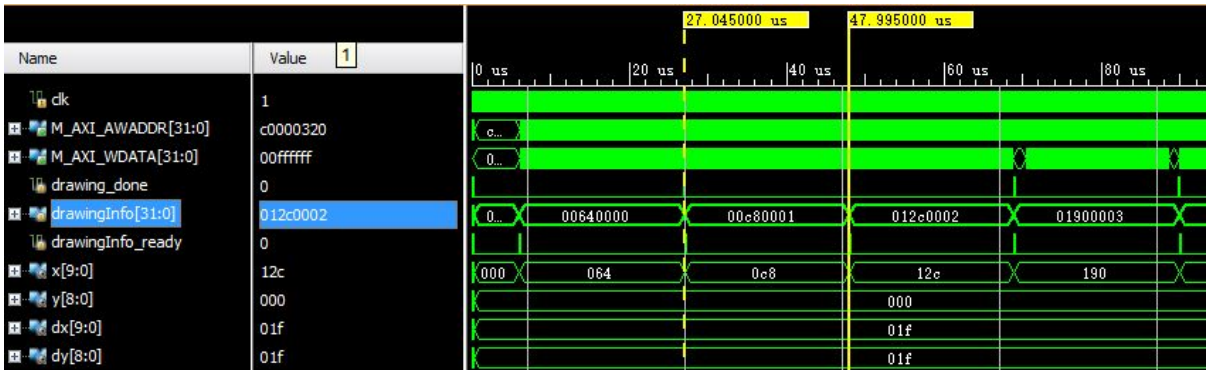

Fig. 6 Graphics Block Simulation

The block is very efficient using pipelining and burst writes. The simulation from Fig. 6 shows it takes about 20us to draw a 64x64 image. We did some stress tests to draw 100 objects moving on the screen and the video was still very fluent.

4 32-bit registers are used in this block as shown below.

Register Name	Offset	Description
Control Register	0x0	Bit 0 is used as enable. Bit 1 is used as software reset(active high). Bit [31:2] are drawing target address. (here is set to video buffer in DDR memory)
Request Register	0x4	Drawing requests are written to this register.
Custom Counter	0x8	Can be used to count frames per second.
Game Control Register	0xc	Read-only register connected to SW. SW[0] is used as game restart.

4.3 Audio Block

The Audio Block is a AXI Slave to generate the required back ground music and sound effects as a enhancement to improve the gaming experience. **Fig. 7** is a brief representation of its structure.

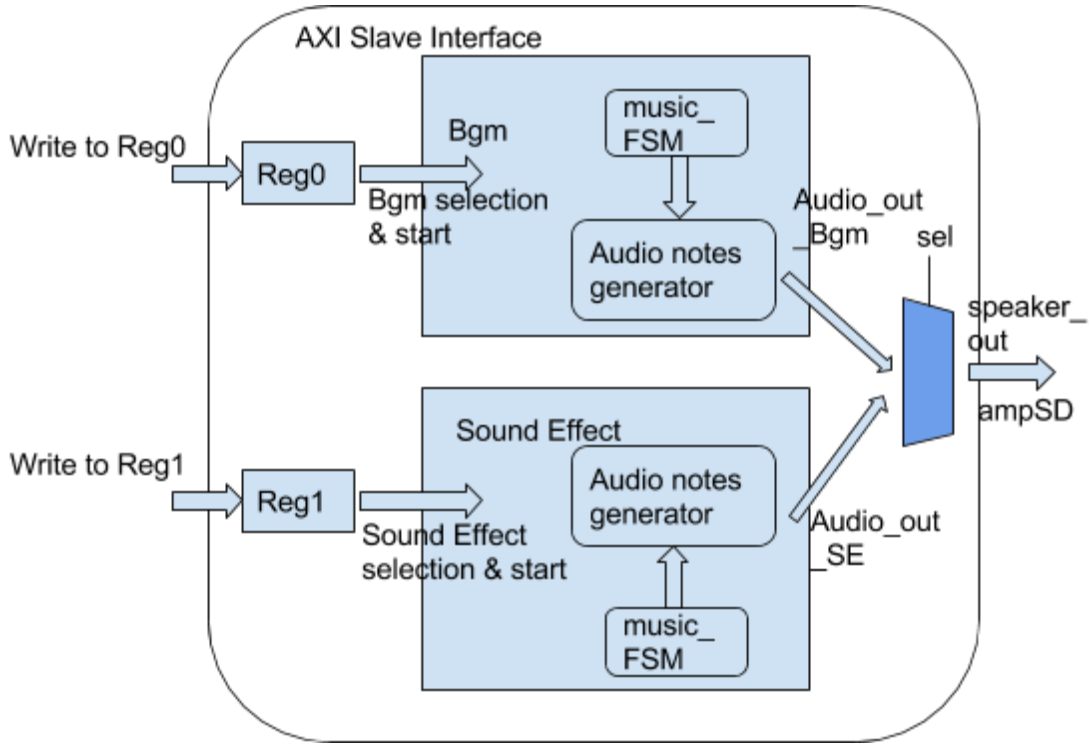

Fig. 7 Audio Block IP

There are two major data path in the audio IP, one is to play the background music, the other one is to play the sound effect.

To play the bgm, the gaming software is supposed to write the bgm index(0-7) to reg0 of the AXI slave interface. Once a write operation to reg0 is detected by the hardware, the bgm block would start to run through one of the music FSMs with the required index stored in reg0. At this point, one FSM (index 1) has been implemented to play a fraction of the music ‘Tinkle Star’ repeatedly. To stop playing the bgm, index 0 should be written to reg0.

At each state of the FSM, the control signal for the Audio notes generator block would be updated to play a single note. These control signals includes choosing

which note to play, the play time of this note and the output volume of this note. Same method is used in producing the sound effect as well.

There are two output signal of the audio block IP. One is ‘speaker_out’ as the audio output and the other one is ‘ampSD’ as the enable signal to turn on the low-pass filter on the board. To output the sound effect while playing the bgm, a multiplexer is used to select the connection of the bgm module or sound effect module. The bgm module is always connected to the output as the default setup, unless when the sound effect module is producing output. After the sound effect is over, it would switch to play bgm automatically.

4.4 Software

The software is downloaded to Microblaze processor for game logic control. For every frame, it reads player movement from motion detector block. Then calculates the resulting movement of aircrafts. Finally issues request to graphic block to display on the screen. Game rules such as hitting enemies with bullets and incrementing scores are also implemented in software. The location and other related information such as velocity are stored for different objects.
The table below provides descriptions of the key functions.

Function Name	Description
insert_bullet_”a”(int x, int y)	Generate a bullet object at x,y location. “a” represents different perspectives such as player1, player2, enemies, or final boss.
draw_bullet(void)	Draw all the available bullets within the bullet object array.
erase_and_calculate_bullet(void)	Erase the bullets at the current locations, then calculate the new locations in the next frame based on the x, y coordinates and given velocity. Main game logic is implemented here
show_score(int start_x,int start_y)	Display the score at a given x,y location
update_score(int score,int x, int y)	Update and redraw the score at a given x,y location

game_over(int start_x, int start_y)	Display the “game over” page at a given location
success(int start_x, int start_y)	Display the “success” page at a given location
welcome(int start_x, int start_y)	Display the“welcome” page at a given location

4.5 Other Existing IPs

TFT Controller:

This is an existing IP which continuously reads a frame data from a specified memory location. We use it as VGA controller.

Microblaze:

This is the processor we are using. It executes software game logic code, reads detected position, and sends requests to graphics block and audio block.

AXI Bus Interconnect:

This is the bus connects most IPs together.

DDR Memory Controller:

The video buffer is located in DDR memory and this is the memory controller.

5. Description of Design Tree

Repository Structure

Docs: Contain the proposal and final report for this group

Src:

- ECE532_integration:
    - ECE532_integration:This folder contains the main project, both digital system and the SDK software. Contains all files needed to run the Raiden game.
    - ip_repo: Contains the IP modules used in this project.

- audio_out_1.0: Contains the custom audio IP
- Graphics_IP_1.0: Contains the custom graphics IP
- pmod_input2_1.0: Contains the video decoder IP, provided in ECE532 Piazza
- position_locator_1.0: Contains the motion detector IP

6. Tips and Tricks

AXI burst write - notice what maximum burst length the target slave supports.
FIFOs - choose FIFO types other than built in FIFO. Otherwise cannot simulate without building a model.
IP names - be careful with IP(like BRAM, FIFO) names, may conflict in global scope after integration.

Audio Block debug - The audio output frequency (range from Hz to KHz) is much lower than the 100MHz system clock frequency. When performing the system simulation, the test feature is needed to increase the audio output frequencies to MHz range. This could save a lot of time waiting for the simulation output.

Test bench simulation - If you have any related files that is required by the test bench simulation such as picture files or binary files, you will need to add these files by “add sources” into your project. Otherwise they can’t be found during simulation.