ECE532 - Final Design Report: Super Skule Fighter

Mohammad Tabrizi
Marco Chung
Xingyu Wan
Team #10

April 13, 2017

Table of Contents

1 Overview

1.1 Background

The project will be to create an augmented reality fighting game using image processing and object tracking for hit detection and augmented image display. The main idea is to use a camera to collect video from two players standing at a safe distance from one another and display a modified video feed where they appear to be in a video game like Street Fighter™.

In such a game, a major component is the detection of player attacks against the other player and correct health tracking in order to display who is winning and losing. The other components are immersion and safety. By replacing the background to classic video game stages, players feel as if they are characters within a video game, making the game more enjoyable. At the same time, safety is also extremely important and will be guaranteed by having players stand at opposite sides of the camera at a safe distance and editing out the space between them to make it look like they are hitting each other when, in reality, they are completely safe from harm.

1.2 Goals

The main goals of this project are:

- Receive HDMI video at 720p/60fps and output augmented feed without loss of quality
- Motion detection of two players through tracking of specific colored gloves (red and blue) in video stream
- Green screen background that immerses players in different environments
- Removing safety gap between two players from output video stream (optional pending mid-project review)
- Display player health bars and announce winner when a player runs out of health
- Create a basic augmented reality video game, as seen in Figure 1

Figure 1. Goal

1.3 Block diagram

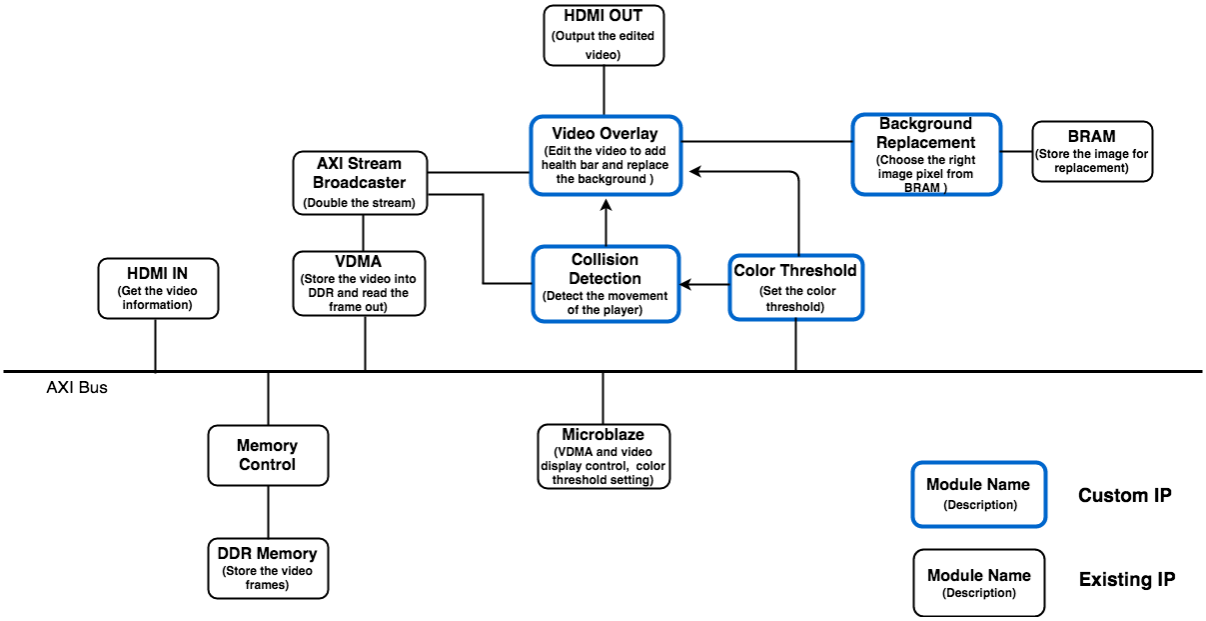

Figure 1.2 Block diagram

Figure 1.2 shows the block diagram of our project. At first, the video is fed into the AXI bus and stored in the DDR memory through VDMA. Then, the video frame information is read out by VDMA from memory and sent to AXI Stream Broadcaster block that duplicates the input stream into two output stream signals. One of the streams is sent into the Collision Detection block for detecting the movement of two players; another one is sent into the Video Overlay block for creating player health bars and replacing the green screen background. Then the edited frame signals will be sent out to be displayed on the monitor. The Background Replacement module grabs the required background image pixel from the BRAM and sends it into the Video Overlay block. Moreover, different lighting conditions require different color thresholds for the Collision Detect and Video Overlay blocks to detect Red, Blue, and Green. In order to account for this, the Color Threshold module was created to allow the user to set the threshold values using software running on the Microblaze system to write to memory mapped registers.

1.4 Brief Description of IPs

The following is a list of all of the IPs used in this design, their function, and their source of origin.

Module	Function	Source
Collision Detection	● Receive color of the pixel through the VDMA ● Judge whether or not there is a hit, based on flags and colors in the current and previous frame	Custom
Background Replacement	● Upscale a 640x360 image to 720p and send correct replacement pixels from the BRAM	Custom
Video Overlay	● Edit the video output to add health bar and replace the green background	Custom
Color Threshold	● Set threshold values, color minimums, health bar lengths, and initial health for the design	Custom
BRAM	● Store the image used in background replacement	Xilinx
AXI Stream Broadcaster	● Duplicate the AXI stream signals	Xilinx
HDMI IN (Multiple IPs)	● Contains DVI to RGB Video Decoder block and Video In to AXI4-Stream block ● Decode the input video signals and convert them to AXI stream signals	Xilinx
HDMI OUT (Multiple IPs)	● Contains AXI4-Stream to Video Out block and RGB to DVI Encode block ● Convert AXI stream signals to the video output signals and encode them	Xilinx
VDMA	● The buffer to the video frame stored in memory ● Controlled through software to output video frames	Xilinx
Microblaze	● Initialize the VDMA and set the parameters for video output, like frame size ● Set the color threshold parameters	Xilinx

2 Outcome

2.1 Results

The final design has the features originally planned for completion. Players can stand within the camera's view at opposite sides and use motions of their red and blue gloves to score hits on the other player and decrement their health. Once a player has lost all of their health, the game is over and the winning player is declared on the screen.

Throughout the design process, there were deviations from the original proposed design. Originally, the design had tasks split up across the components accordingly.

- Camera: collect video from two players and streams it into the system
- FPGA: read the frame in, store it in memory, and perform image processing
- MicroBlaze processor: track game data and draw overlays onto the frame (including green screen detection and replacement)

In the final design, the camera still captures video and streams it in as hdmi and the FPGA is still responsible for the image processing and green screen detection in the proposal. However, the MicroBlaze processor is now responsible for adjusting the thresholds and minimum intensity values. Tracking game data and performing video overlay was moved to hardware. Our group came to this decision by milestone 2 when we attempted to draw the health bar on top of the output video. The result was that the MicroBlaze processor couldn't keep up with the hardware at 60 fps. Instead of throttling the whole design to around 24 fps by calling the frames in software, we chose to move the video overlay to hardware.

Also, instead of cutting out the middle section of the video so the two players appear closer together, background replacement was completed first. The reasoning was that there was no good method to deal with the missing real-world background that is cut out without first replacing the background. The two players would look still look like they were spaced apart due to their real-world backgrounds not matching up. Once the background replacement was complete, there was not enough time to add a frame cut out.

The final system itself has no operational issues. The Nexys video board needs to be connected properly first, with the camera connected as hdmi input and a display connected to the hdmi output. The project should be loaded onto the host computer. The host computer will program the bitstream into the board and run the software through the SDK. The thresholds for color detection can be controlled through the terminal.

Figure 2 shows the final system while its running with two players. Figure 2a shows the players standing in front of the green while figure 2b shows the augmented output of the game on the monitor with background replacement and health bars. Figure 2c shows the announcement of the red player as the winner since the blue player's health has run out.

Figure 2a

Figure 2b

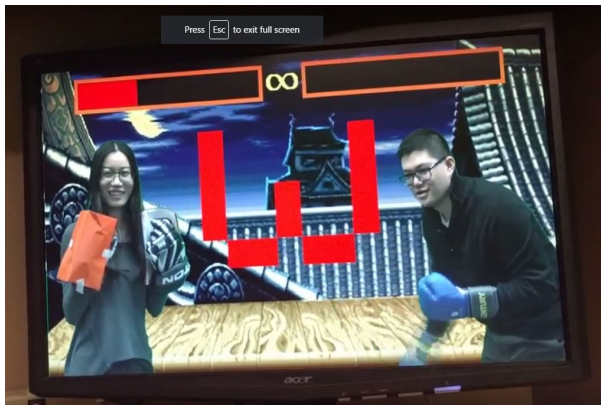

Figure 2c

Figure 3a shows background replacement with an input video of the green screen while 3b shows a partial replacement as the green screen only occupies part of the screen.

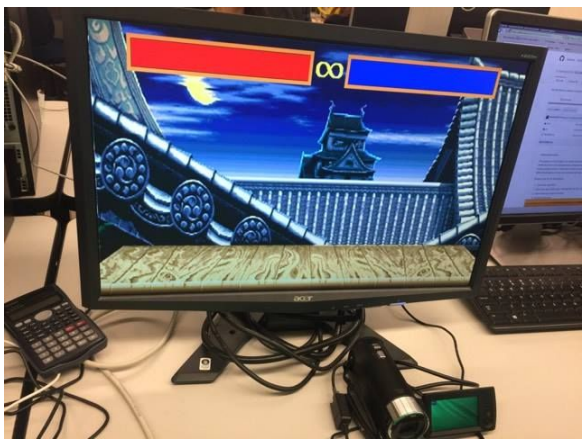

Figure 3a

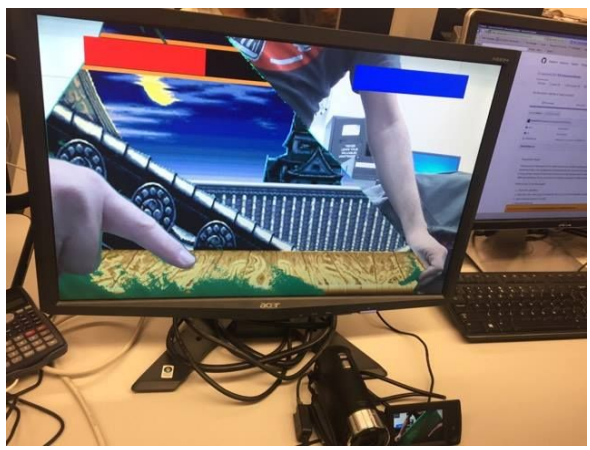

Figure 3b

2.2 Possible Future Improvements

The system has several areas for improvement. A more robust algorithm could be used for color detection of the red and blue. Sometimes, depending on the distance from the camera and the lightning of the room, red and blue may not be detected properly.

The next logical step to improve the design is to implement the cut out feature. Since the background replacement is completed, the middle section between the players can be cut out and the background behind each player will look continuous.

In addition, the system can be upgraded to recognize specific moves from the user, instead of just the presence of color or not. Specifically, this would be within the collision detection block. The block would need to keep track of user's approximate location every frame and then decide whether or not a special input action has been performed.

3 Project Schedule

The following is a table that shows group's weekly progress. There is a description for both the proposed tasks at the start of the week and the tasks completed by the end of the week.

Week 1	**February 10**
Proposal	**Actual**
Implement HDMI IN/OUT and DDR Memory modules	● Converted Digilient Nexys Video Board HDMI demo from 2015.4 to 2016.2 ● Demo included HDMI Input and Output using DDR memory using a VDMA module
Decisions: Moved test bench demo 1 week back due to change in course structure, swap milestone 2 and 3	**Milestone Demo:** Showed a working system with a camera video feed going from input to output on to monitor
Week 2	**February 17**
Proposal	**Actual**
Show testbench with HDMI input video being fed directly to output	● Test bench demo was moved to week 3 by course coordinator, milestones were modified ● Developed collision detection algorithm in MATLAB ● Developed FIFO logic for passing collision data to software ● Studied HDMI demo C code and hardware and attempted to draw health bars using C
Decisions: Will move video editing to hardware as software is too slow handle edits at 720p/60FPS	**Milestone Demo:** Showed working MATLAB code and FIFO logic diagram. Displayed health bar being written in C but it was too jittery due to the slower microblaze speed
Week	**March 3**
Proposal	**Actual**
Start Collision Detection module and Microblaze with Game Logic	● Finished implementing Collision Detection module ● Created test bench for verifying Collision Detection functionality ● Attempted to insert Collision Detection into top level project and used ILA's to debug connectivity issues
Decisions: None	**Testbench Demo:** Showed working Collision Detection test bench and the ILA testing of its integration
Week 4	**March 10**
Proposal	**Actual**
Finish Collision Detection module and Microblaze with Game Logic (Evaluate progress and make changes to project plan)	● Started work on game logic and health bar overlay in hardware ● Used AXI Stream Broadcaster to send video stream to the Game Logic and Collision Detect as well as the output video feed

Decisions: Stopped trying to use a second VDMA for video duplication and used stream broadcasting. Decided to complete background replacement first instead of video editing because the real-world backgrounds were not continuous after video edits.	**Milestone Demo:** Showed system with video streaming from input to output while showing the collision detect and game logic using LEDs to indicate player health and using colored paper to produce damage to players
Week 5	**March 17**
Proposal	**Actual**
Get Video Processing module working (middle extraction)	● Finished game logic module and health bar overlay ● Added changeable thresholds for each color to Collision Detect to help calibrate to different lighting conditions ● Was not able to integrate health bar overlay module and Collision Detection together within top level project ● Started work on the Background Replacement module
Decisions: Include an actual green screen in the demo of the final project	**Mid-Project Demo:** showed 2 separate demos. One that displayed health bar overlay without jitter and one for demonstrating the color thresholds and sensitivities used in the Collision Detection block
Week 6	**March 24**
Proposal	**Actual**
Combine all modules	● Found issue involving failed system integration in previous week due to AXI Stream and Video Timing Control timing ● Modified game logic and health bar overlay ● Integrated Collision Detection along with game logic and health bar overlay into working top level project ● Wrote Python script to convert image to .coe file ● Created BRAM module to hold a 640x360 image in .coe format ● Finished green detection for Background Replacement Logic
Decisions: None	Milestone Demo: Showed integrated system with working Collision Detection and active health bar overlay with updates as well as replacement of green screen background with a single color (black)
Week 7	**March 31**
Proposal	**Actual**
Test/Debug final system	● Finished Background Replacement using an upscaled image ● Added logic for announcing winner ● Integrated entire system ● Finished project, everything fully working
Final Demo: Demonstrated fully working system with Background Replacement, Collision Detection, and active health bar overlay and winner announcement	

4 Description of the Blocks

The following are detailed descriptions of each of the IP blocks that the group used. For each block, their function is given as well some insight into the logic within the block.

4.1 Collision Detection

This is a custom IP block that reads from the AXI Video Direct Memory Access block (axi_vdma_0) and outputs the correct game data to another custom IP block for video overlay (video_overlay_0). It is created from the Create and Package IP Wizard in the Vivado IDE and contains 3 registers that store data from the vdma in AXI Stream format. The block expects the 8-bit red, blue, and green intensities of the incoming pixel and determines whether the pixel meets the requirement for the corresponding colour by comparing it to thresholds and minimums set in software-controlled registers. For the red player, if there is sufficient redness in the right two-thirds of the input stream, a flag goes up that the red player has entered the hitzone for the blue player. Next, if there is not sufficient redness detected in the area in the next frame, the design will believe a punch has been completed and will decrement the blue player's health. The same is done for the blue player, the only difference being that sufficient amounts of blue need to be detected in the left two-thirds of the input frame. The block then outputs each player's health to the video_overlay block for display on the screen.

4.2 Video Overlay

The main function of Video Overlay block is to add the health bar and perform background replacement on the original video. There are several input signals for this IP: AXI stream signals (frame information), background image pixel signal and health bar length signals. The coordinates of input video pixel signal on the frame is calculated by using the stream signals (tlast, tuser, tvalid, tready). When the input video pixel is on the position that we designed for health bar, the output data will be red or blue pixel. When one of the player wins the game, there will be a large "W" with the color of the winner displaying on the screen. Moreover, this module also performs green color detection by applying the same algorithm as the Collision Detection module. When the input video pixel is determined to be green, the output video pixel will be a pixel from the background image instead. Aside from the above two situations, the output video data will always be the same as the input video data.

4.3 BRAM

The BRAM was used to store a 640x360 pixel image with 24bits/pixel in RGB format. The BRAM was configured as Read-Only Memory configured to be always enabled with no clock enable port and with a direct output with no registers in the path to allow for the least number of clock cycle delays from request to output. A python script was written to convert an image into a .coe file which was used to initialize the BRAM with the contents of the desired background image.

4.4 Background Replacement

This module is used to address the BRAM that contains the background image for pixel data that is to be used by Video Overlay to replace detected green screen pixels. The module receives the current position within the frame that is currently being streamed from Video Overlay as two values, horizontal and vertical position, and uses combinatorial logic to address the BRAM for the *next* pixel that has not been received yet by Video Overlay.

This is done so that by the time the new pixel is in, the corresponding replacement pixel is ready if the green screen is detected by Video Overlay. After the correct address is received by the BRAM, we must wait for a positive clock edge before we receive the corresponding pixel and due to this delay of one clock cycle, we must always be addressing the BRAM for a replacement pixel one cycle before the pixel has come in.

Due to the limited number of available BRAM bits, the largest image we were able to store in the BRAM block was 640x360 which is half the size of a 720p (1280x720) image across both dimensions. Because of this, the background replacement logic also had to upscale this image by repeating every pixel within the background image 4 times, 2 times across a row and 2 times across a column. This was done by using combinatorial logic to convert the current horizontal and vertical positions within the frame received from Video Overlay to their corresponding position within the smaller image by, in essence repeating, every background image pixel in a line twice and repeating every line twice.

4.5 AXI Stream Broadcaster

The Collision detection block and Video Overlay block both need the AXI stream signals that carry the input video information. This block helps to duplicates the AXI stream signals into two same AXI stream signals.

4.6 Other IPs

The IPs related to HDMI IN and HDMI OUT are borrowed from the Nexys Video HDMI Demo from Digilent [1].

5 Description of Your Design Tree

The following is a description of the design tree posted on GitHub.

- **src:** the project files
    - **Projects/hdmi:** the main project folder
        - **proj:**
            - **hdmi.xpc:** the vivado project file
            - **hdmi.srcs/sources_1/new/**: location of two standalone verilog files
                - **BGR.v:** background replacement verilog file
                - **bypass.v:** video overlay verilog file
            - **hdmi.sdk/video_demo/src:**
                - **video_demo.c:** the C file for VDMA initialization, video display control and color threshold setting
        - **src**: source folder for synthesis
            - **bd**: block diagram directory
            - **constraints/NexysVideo_Master.xdc**: constraints file
    - **collision_detection:** the IP for blue and red detection and hit judging
    - **color_threshold:** the IP for setting threshold value for color detection
    - **n_BG.coe:** the .coe file of image for background replacement

- **doc**: the documentation that supports the project
    - **Presentation.pdf:** powerpoint file used during final demo presentation
    - **demo_video.wmv:** video demonstration of the project in operation
    - **Group Report:** report file

- **input_data:** contains images, photoshop, and scripts used for generating .coe files

- **README.md:** general description of the project

6 Tips and Tricks

For editing the video, adding another VDMA is not the only solution. You can edit the frame on the input of the AXI4-Stream to Video Out. An issue is that the input of AXI4-Stream to Video Out should be synchronized with the video timing control signals. One method is to use combinatorial logic for pixel data modification to make sure there are no delays in the pixel data path

7 References

[1] “Nexys Video HDMI Demo,” [Online]. Available: https://reference.digilentinc.com/learn/programmable-logic/tutorials/nexys-video-hdmi-demo/start [Accessed 11 Apr. 2017].