ECE 532 DIGITAL SYSTEM DESIGN

Professor: Paul Chow

TA: Vincent Mirian

Group Report

First Person Shooter Project

Patrick Judd ([redacted]
Ian Katsuno ([redacted]
Bao Le [redacted]

Table of Contents

Overview 2
- Project Description 2
- Project Motivation 2
- Project Background 3
- Project Functional Requirements 3
- System Block Diagram 4
- Brief Description of IP 5

Outcome 6
- What Works: 6
  - First demo: 6
  - Second demo: 6
- What Does Not Work: 7
- Improvements to the Current System 7
- Future work 7

Detailed Description of IP Blocks 8
- IP_to_Mem and mem_to_ip 8
- IP_to_mem 8
- Target Locator Logic 8
- Mem_to_ip 9
- Microblaze 9
- Multi Port Memory Controller and DDR Memory 10
- TFT controller 11
- XPS_Ps2 Core 11
- Mouse Handler 12
- Central Software Controller: 12
- Target Remover: 12

Description of Design Tree 13
- Project Folder #1: Target_Detector_No_Mouse 13
- Project Folder #2: Target Remover_Mouse 13
- Doc Folder 13

Overview

Project Description

The goal of this project was to create a "live action" first person shooter game. It uses a stationary video camera connected to the Xilinx XUP Virtex 2 Pro FPGA board to capture the live video feed. This live video feed is modified and displayed on a computer monitor using the VGA output of the FPGA board. Using a mouse connected to the FPGA, the player of the game look at the computer monitor and shoot at “targets” which move within the camera's field of view. To start the game the user must first take a reference shot of an empty section of a room to serve as the stage for the game. Any person who then appears on the stage in the cameras field of view becomes a “target.” If the player is successful in 'shooting' (clicking on) a target then the target is be removed from the screen. Thus, it is similar to a first person shooter except for the fact that the both the game environment and the game targets are live-action.

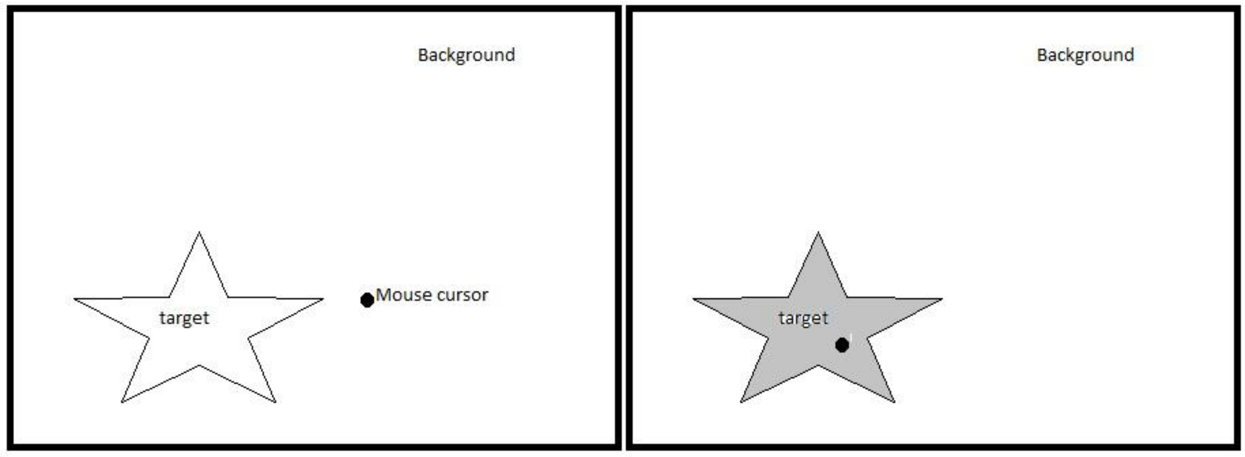


Figure 1: Grayed Out Target Indicates it has been “shot” by mouse

Project Motivation

The motivation for this project stems from three trends in today's consumer computing applications:

1. **First-Person Shooters:** One of today's most popular video game genres is the first-person shooter. These can be described as action games in which the player must shoot targets on a screen from the first-person perspective. Examples include the Call of Duty series and Counter-Strike[1]. The gameplay mechanics in this project closely resemble the gameplay of these franchises.
2. **Augmented/Diminished Reality:** Augmented or diminished reality applications have become increasingly common, largely due to the ubiquity of cameras in smart-phones. These are applications in which a live-action video feed is modified to give the user a changed view of reality. These applications are deemed augmented if they add information or diminished if they remove information [2]. Our project could be classified as both. It is diminished reality because we are removing targets from the feed, but it could also be augmented if a score or life bar were added to the video with more development.
3. **Motion Gaming:** With the introduction of the Nintendo Wii [3]. It was proved that there was a

lucrative mainstream market for Motion gaming. Motion gaming is when players affect the game though motions of their body rather than by pushing buttons. Our project includes motion gaming not from the shooter side, but from the target side. Targets could be real people which move in an attempt to make themselves harder to hit.

Our project incorporates aspects of all three of these trends. The heightened interest and profit potential which surrounds each of them serves to motivate our project.

Project Background

The framework of this system draws from that of a past ECE532 project, namely the Color Replacement project from 2009 [4]. The most important concept borrowed from this project was the manner in which the targets would be made to disappear. The Color Replacement project was able to make certain colours invisible. This was done by first taking a still reference shot with a stationary camera. Then, they would examine pixel data to search out for the colour they wished to make invisible. If they detected a pixel of that colour they would replace its data with that of the reference picutre, thus, making it “invisible”. Our project works in a similar manner, except that instead of replacing pixels of a specific colour, we replace all pixels which are part of a target that has been hit (clicked on). Pixels are determined to be part of a target if they differ from the reference picture. A more detailed description of the target detection algorithm can be found in the detailed IP descriptions.

There are two major aspects which set our project apart from the Colour Replacement Project and other games which have been created for this course:

1. This game uses a pointing device as a communication tool between the user and the system. This allows the interaction more precise and responsive.
2. Our video processing would be color independent. We believe this allows our game to be more liberated from the environment and thus makes the game more exciting.

Project Functional Requirements

From the outset, our project had the following requirements:

1. The system must be able to handle 30 frames per second of video
2. The system must be able to handle a video feed with a 640x480 resolution
3. The system must be able to display the input video frames which have been modified by the internal logic
4. The system must be able to identify a “target” in the input video feed
5. The system must also be able to track the movement of the “target.” The system must decide if an identified “target” is the same “target” identified in the previous frame, or if it is a new target in the frame.
6. The system must be able to display the mouse cursor/crosshair on the output image
7. The system must allow targets to be selected using the mouse

System Block Diagram

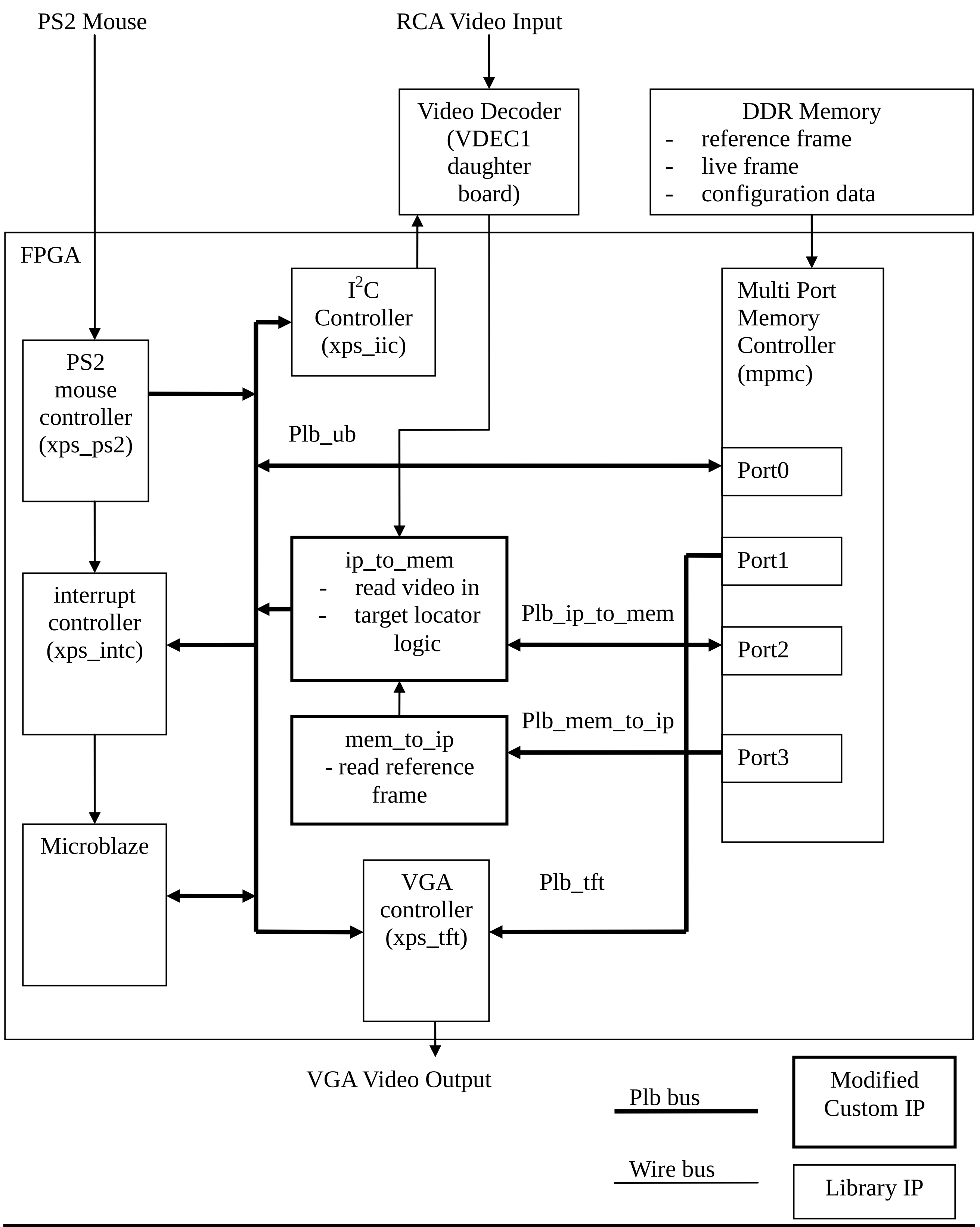


Figure 2: Hardware Block Diagram

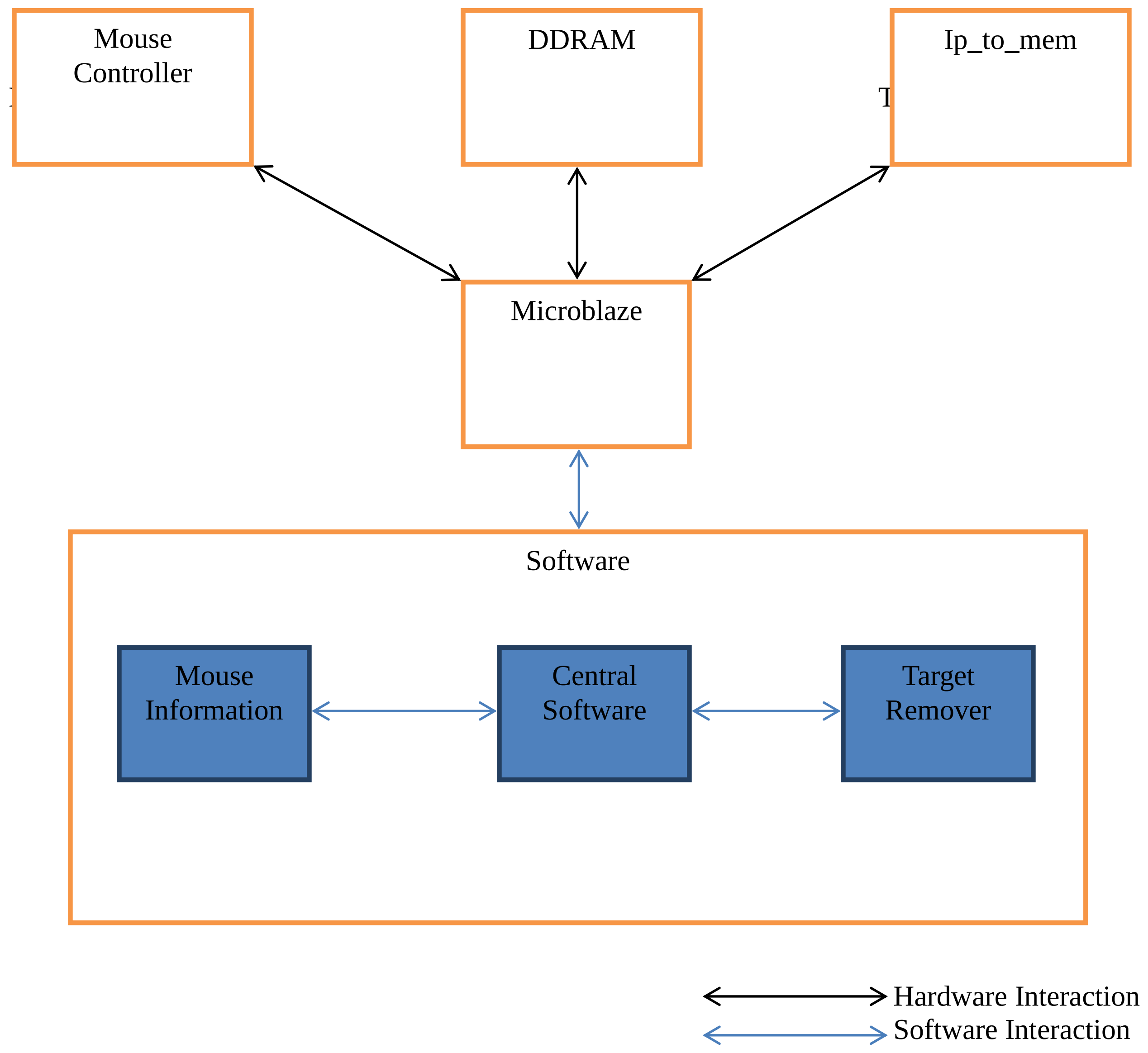


Figure 3: Software Block Diagram

Brief Description of IP

Block	**Description**	**Origin**
	Hardware IP Blocks in FPGA	
IIC	Sends commands to the video decoder	Xilinx IP
ip_to_mem	Detects targets, writes video data to memory	Modified from Colour Replacement Project
mem_to_ip	Reads reference image from memory, supplies it to	Imported from Colour Replacement Project

	ip_to_mem	
Microblaze	Executes software in final_software.c	Xilinx IP
MPMC	Memory controller which has multiple ports	Xilinx IP
PLB	Bus used for communication between Microblaz and peripherals, as well as transferring video data	Xilinx IP
TFT	Reads frame buffer from memory and outputs to VGA	Xilinx IP
XPS_Ps2	Provides interface for transmitting data over PS2 Port	Xilinx IP
Software Blocks*		
Mouse Handler	Draws mouse cursor on screen	Custom
Hit/Miss Detector	Detects hit or miss on click	Custom
Target Remover	Removes Target from video	Custom
Hardware Outside FPGA		
VGA Monitor	Displays Video	School Supplied
PS2 Mouse	Controls Cursor	Self Supplied
Video Camera	Captures Video	Self Supplied
Video Decode Daughter Board	Decodes Video	School Supplied

Table 1: Brief Description of IP Blocks
*Note: These are logical Software blocks, all software is contained in file final_software.c

Outcome

What Works:

We will have two demonstrations on the demo day:

First demo:

- A mouse is used as a pointing device, its cursor is drawn on screen, and its information is stored in the software.
- The software extracts the target data from hardware. For this part, the target coordinates are hard-coded in hardware. In other words, the target detection logic is not employed.
- The software will detect if a user click inside the area occupied by the target. If so, the target will be removed from the video feed, the background is displayed at that area instead.

Second demo:

- A new object appears on screen. This whole object will be painted blue.

These two demonstrations show that we have had all of our hardware, software blocks functional.

However, when we integrate the mouse controller block with the two modified custom IPs, we have problem that makes our whole project not fully working.

What Does Not Work:

When we integrate the mouse into with the two custom IPs, the video input signal becomes garbled [5]. Basically, random lines are drawn on screen. Consequently, our target locator outcome becomes falsified as it always detects these lines as targets. We have no luck to find the source of this noise in the video feed, and the reasons why the existence of the PS2 block creates this noise. This phenomenon can be witnessed in the first demonstration.

As a result of this effect, we separate them for the sake of our demonstrations. We want to show that our target locator logic in correct video in stream would output the correct target coordinate. And when the target coordinate is correct, our system will behave in a correct way.

Improvements to the Current System

The first improvement would be to discover and remove the source of the noise that is introduced when the ps2 core is added to the project. This would allow our target locator to work with the mouse cursor and provide the basic game play mechanic of our design. Another improvement to our design would be easier configuration of the target locator. Currently the parameters Minimum Target Width and Pixel Difference Threshold are set by writing to memory using XMD. We could improve this configuration interface by using the push buttons to increase and decrease each of these parameters.

Future work

We had initially envisioned a much more sophisticated system for our project. However, we decided to scale back some of the features in order to meet time constraints. We feel that adding these features would be technically challenging and greatly improve the final product.

The first feature we removed was support for multiple targets. We feel this would create a more interesting and challenging user experience. Being able to detect and distinguish between multiple targets would require much more sophisticated algorithms than that used to detect a single target. There are a lot of cases to take into account involving how the targets move around the screen and interact with each other.

The second feature we removed was the use of a light gun as the input device instead of the mouse. Our idea was to have the game function similarly to Nintendo’s Duck Hunt, where the user would aim a light gun and the screen to shoot targets. The light gun works as follows: when the user pulls the trigger the screen flashes a black and white pattern. The black represents the background while the white represents a target. The light gun can sense the color on the part of the screen that it is aimed at and thus can determine whether or not you have hit the target. The challenge in adding the light gun peripheral would be the hardware interface. The light gun would likely not have a connector that would connect to the FPGA board so some extra interfacing hardware would be required. The system would also need to be modified to produce the black and white flash.

Detailed Description of IP Blocks

Note: Only IP blocks which have non-trivial configurations or functions will be described.

IP_to_Mem and mem_to_ip

The ip_to_mem and mem_to_ip cores were borrowed from the Real Time Color Replacement project [7]. These cores contained much of the functionality needed by our project. This includes reading video from the video decoder, writing the image data to memory and reading back image data from memory. This served as the necessary framework for our target locator logic.

IP_to_mem

The ip_to_mem core reads the video data from the video decoder and writes that data to the memory. It also reads configuration data from the memory using the same PLB port that it uses to write the video data. For information on the FSMs that controlled the read and write operations see the Real Time Color Replacement documentation [7]**.** We modified the core by removing the color replacement logic and implementing our own target locator logic. We also reused the configuration data address but changed the configuration encoding. Our configuration encoding is show in table 2. Software accessible registers and a slave PLB port were also added to communicate with the microblaze.

Configuration Address	Bits	Description
0x9020d000	[31:22]	Pixel Difference Threshold
	[21:10]	Reserved
	[9:0]	Minimum Target Width

Table 2: Configuration Encoding

Target Locator Logic

We define a target to be anything in the live video frame that is different from the reference frame. The function of the target locator is to determine the x and y coordinates of a box that encloses the target. Specifically the target box is defined by the top left, (x_min, y_min) and bottom right (x_max, y_max) corners of the box (see figure 2). These four values are then stored in 4 software accessible registers. The microblaze can read these registers via the microblaze plb connected to the slave plb port.

Figure 1. Reference Frame

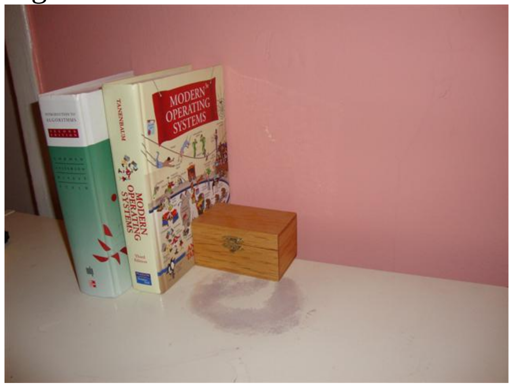
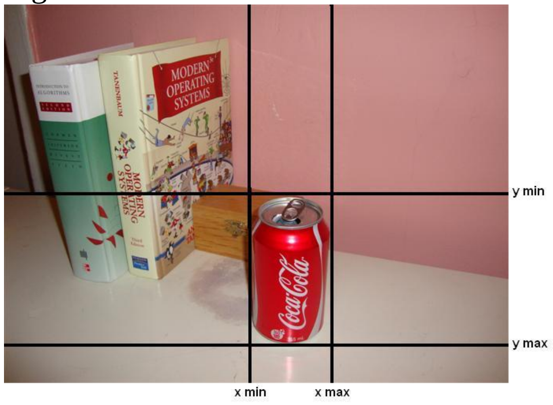


Figure 2. Live Frame



The target is located by comparing pixel values of the live image with the reference image. If the difference between the reference pixel and the live pixel is greater than a certain threshold then that pixel is considered part of the target. The threshold was introduced to account for noise in the video signal and slight variations in lighting which could cause the live video to differ from the reference frame even though nothing has entered the frame. The threshold can be set using the configuration memory as described in table 2. Pixel data is compared by comparing the individual color channels and summing the differences. A pixel is determined to be part of the target if the following expression holds.

| live_red – ref_red | + | live_green – ref_green | + | live_blue – ref_blue | > threshold

One issue we had was with the noticeable jitter of our input video feed. This caused errors in our detection logic around sharp edges in the background because pixels seemed to be moving back and forth in the horizontal direction. This caused our detection logic to falsely identify target a target due to pixels at the edges drastically changing in color. To overcome this we added a minimum target width requirement. A minimum target width section was added to the configuration data as shown in table 2. The logic was modified so that a consecutive set of "different" pixels would not register as a target until it reached the minimum target width.

Mem_to_ip

The mem_to_ip core reads reference image data from memory and stores it in a local buffer. That buffer is connected to the ip_to_mem core so that ip_to_mem can read the reference image data. This core remains unchanged from the Real Time Color Replacement project. For information on the read FSM see the Real Time Color Replacement documentation [7].

Microblaze

The Microblaze instance in this design is version 7.20.d and is configured to run at an operating clock speed of 100MHz. It is connected to a PLB bus which allows it to communicate with peripherals.

Multi Port Memory Controller and DDR Memory

The multiport memory controller (mpmc) is a Xilinx EDK IP which allows multiple port connections to an external DDR memory attached to the FPGA. In our project the DDR memory is used for three purposes:

1. store the live video frame
2. store the reference frame
3. store configuration data for the ip_to_mem block

The addresses for each memory object are summarized in table 1

Object	Address Range	Size
All allocated DDR memory	0x90000000-0x9FFFFFFF	256 MB
Live Video Frame	0x90000000-0x90200000	2 MB*
Configuration Data	0x9020d000-0x9020d000	4 B
Reference Frame	0x90400000-0x90600000	2 MB*

Table 3. DDR memory address map

* video data should be stored in 1024x512 word memory region (1 pixel = 1 word = 4 bytes) [6]

The mpmc has 4 ports, which are each connected to a different core via a plb bus. Each core that requires memory access was given its own port and plb connection in order to allow simultaneous memory accesses to different addresses.

Port 0: microblaze plb

The microblaze needs to be able to write to the live video frame in order to draw the cursor and read the reference frame in order to perform the target removal.

Port 1: tft plb

The vga controller needs to be able to read the live video frame in order to display it on the vga monitor

Port 2: ip_to_mem plb

The ip_to_mem core required memory access for two reasons. First, it needs to be able to read the configuration data stored in memory. Second, it needs to be able to write the input video data from the video decoder to the memory.

Port 3: mem_to_ip plb

The mem_to_ip core needs to be able to read the read the reference image from memory so that ip_to_mem can compare it to the input video. This was done in parallel with ip_to_mem writing the live video data in order to increase throughput.

TFT controller

The tft controller allows image data stored in memory to be displayed on a monitor attached to the FPGA board via VGA. The tft controller reads image data from the DDR memory via a dedicated PLB bus attached to its master PLB port. The tft controller is also connected to the microblaze plb via its slave PLB port. By default the tft controller is set to read the live video frame from memory. However, the microblaze can configure the tft to read the reference frame using the slave PLB port. This feature is useful for debugging.

XPS_Ps2 Core

XPS_Ps2 v1.00a is a Xilinx IP core which allows the microblaze processor to communicate with the PS/2 mouse.. This IP connects as a slave to the same PLB bus which the microblaze processor. It has two sets of registers. One corresponds to the mouse PS/2 port, and one to the keyboard PS/2 port. In this design we have no keyboard so only the mouse port is used.

The following XPS_Ps2 registers are used in our design:

Register Name	**Description**	**Base Address + Offset**	**Type of Access**
SRST_1	Software Reset Register	C_BASEADDR + 0x0000	Write
STATUS_REG1	PS2 Core Software Register	C_BASEADDR + 0x0004	Read
RX1_DATA	Receive Data Register	C_BASEADDR + 0x0008	Read
TX1_DATA	Transmit Data Regiser	C_BASEADDR + 0x000c	Write
GIE_1	Global Interrupt Enable	C_BASEADDR + 0x002c	Read/Write
IPISR_1	Interrupt Status Register	C_BASEADDR + 0x0030	Read/Write
IPIER_1	Interrupt Enable Register	C_BASEADDR + 0x0038	Read/Write

Table 4: XPS_Ps2 IP Memory Mapped Registers Used in Design

Initializing the PS2 core and configuring the PS2 mouse according to the PS2 protocol are central functions of the microblaze software. Pseudo code for these actions is shown below:

```
//SETTING UP THE PS2 DRIVER INSTANCE:
     Declare Driver Instances of the ps2 core;

     call Xps2_LookupConfig();
     call CfgInitialize();
     call Xps2_SelfTest();

     Specify the interrupt handler using Xps2_SetHandler();
//CONNECT THE INTERRUPT CONTROLLER TO PS2 ISR:
     Declare interrupt controller driver instance;
     call Xintc_Initialize();
     call Xintc_Connect();
//MOUSE INITIALIZATION PROTOCOL:
     send reset code 0xFF;
     spin until receive acknowledge code 0xFA;
     spin until receive self-test-passed code 0xAA;
     spin until receive Device ID 0x00;
```

```
        send “set to stream mode” code 0xFA;
        send “enable data streaming” code 0xF4;
    //ENABLE INTERRUPTS AND START INTERRUPT CONTROLLER
        call Xps2_IntrEnable();
        call Xps2_IntrGlobalenable();
        call XintC_Enable();
        call XintC_Start();
```

Mouse Handler

The mouse handler is software code that will receive data packet from the mouse at interrupts. It then updates the x and y coordinates and the mouse status accordingly. At every interrupt, the mouse send out three bytes:

Byte Index	Containing Data
Byte 1	Mouse status: Left Click/ Right Click/ X Direction / Y Direction
Byte 2	Relative Coordinate in x Direction
Byte 3	Relative Coordinate in y Direction

Table 5: Mouse Data Packet Table

Central Software Controller:

This piece of software is used to do all the software routine work. Every iteration starts of by checking the current mouse coordinates still in the boundary of the screen size (640x480). If a user moves the mouse out of the boundary, its location is adjusted accordingly. After that, it draws the current mouse cursor to the live video frame. Now if a user is currently clicking inside the area covered by the target, the target remover is called; otherwise, go to the next iteration of this routine.

- Pseudo Code :

```
if ((click) && (x_min  <  mouse_x  <  x_max) && (y_min < mouse_y < y_max))
    Target_remover(x_min,x_max,y_min,y_max);
```

Click: - indication of a user is clicking the mouse.
x_min, x_max, y_min, y_max are coordinates of the rectangle covers the target.
mouse_x, mouse_y: mouse location.

Target Remover:

Target remover is called when a target must be hit already. This software simply replaces the pixels in the live frame with pixels from the reference image. These actions are only performed inside the area covered by the target as we only want to remove the target from the video output.

- Pseudo Code:

```
For i = y_min to y_max
    For j = x_min to x_max{
        Int color = read_pixel(reference, i , j);
        Write_pixel(current_image, color, I, j);
    }
```

reference: the starting address of the reference image 0x90400000
current_image: the starting address of the video output 0x90000000

Description of Design Tree

Due to integration problems our design is split into two projects, each of which demonstrates a function of our project. The archive root directory has three folders:

- Target_Detector_No_Mouse
- Target_Remover_Mouse
- Doc

These three Directories are described below:

Project Folder #1: Target_Detector_No_Mouse

Folder	Description
System	This project contains the XPS project which detects the target but has no mouse. **Key files:** **1. ip_to_mem custom logic** **\system\pcores\ip_to_mem_v1_00_a\hdl\verilog\user_logic.v** **2. mem_to_ip custom logic** **\system\pcores\mem_to_ip_v1_00_a\hdl\verilog\user_logic.v**

Project Folder #2: Target Remover_Mouse

Folder	Description
System	This project contains the XPS project which can not detect target but has a mouse and can replace targets. **Key files:** **1. ip_to_mem custom logic** **\system\pcores\ip_to_mem_v1_00_a\hdl\verilog\user_logic.v** **2. mem_to_ip custom logic** **\system\pcores\mem_to_ip_v1_00_a\hdl\verilog\user_logic.v** **3. final_software.c** **\system\sw\final_software.c**

Doc Folder

Key Files:

1. **Project Proposal: Proposal.doc**
2. **Group Report: Group_Report.pdf**
3. **Project Presentation: Project_Presentation.pptx**

References

[1] First-Person Shooter.
http://en.wikipedia.org/wiki/First-person_shooter

[2] Augmented Reality.
http://www.howstuffworks.com/augmented-reality.htm
[3] Wii.
http://www.nintendo.com/wii
[4] 2009 Projects.
http://www.eecg.toronto.edu/~pc/courses/532/2011/pastprojectsnode5.html

[5] Problems caused by interlacing
http://en.wikipedia.org/wiki/Interlaced_video

[6] Xilinx, “XPS Thin Film Transistor (TFT) Controller (v1.00a) Product Specification”
page 9, July 21 2008. available:
http://china.xilinx.com/support/documentation/ip_documentation/xps_tft.pdf

[7] C. Liu, A. Bajenov, J. Sham. “Real Time Color Replacement” April 9, 2009
available: http://www.eecg.toronto.edu/~pc/courses/432/2009/projects/colorreplacement.zip