ECE532 Digital Systems Design

Winter Semester 2016

The Turret Master 5000

Group Report

Tuesday 12th April, 2016

Submitted by

Roberto Bortolussi

Michael Halamicek

Emily Ng

Patrick Payne

University of Toronto

Contents

1 Introduction

Within the realm of video games, a sentry turret is a staple within several genres of gaming including first person shooters and real-time strategy games. Two video games that have such turrets are Portal and Team Fortress 2. The turrets from Portal (pictured in Figure 1) are stationary and their only moving parts are their side-mounted guns. The key inspiration we drew from these turrets was their method of aiming. The way these turrets take aim is as follows: When an object (the player) enters the turret's field of view, it uses its laser to determine where exactly the object is. Once it has successfully targeted the object, it fires its guns. Our group wished to replicate this method of targeting and shooting using a self-contained mock turret system that utilizes a stationary camera, an articulating laser and image processing to effectively take aim and (pretend to) shoot at pre-determined targets within the camera's field of view. To do this with a high definition camera, hardware acceleration is a necessary. As such, the brains of the system were implemented on a Xilinx FPGA.

There exist similar projects that implement motion-sensing turrets in various ways. One of these turrets [1] uses LIDAR to determine the distance of objects from the turret, detecting differences in these distances over time, and shooting when these changes are detected. This technique requires the use of an expensive LIDAR device but allows for a simple algorithm to be used, allowing for the use of a microcontroller instead of an FPGA.

Figure 1: Portal 2 Sentry Turret

2 Overview

2.1 Project Goals

The aim of this project is to develop a turret based laser-pointer with two degrees of freedom, that can identify specific objects, target and then "fire" at them.

Video processing is done by a custom IP. This IP performs object recognition and laser detection. A MicroBlaze processor determines which target to fire at and how to adjust the turret position in order to do so. The physical mechanical system incorporates a laser pointer on a mount with two servos to alter the polar and azimuthal angle of the laser pointer. The system also produces turret sound effects faithful to the original game at specific trigger points determined by the MicroBlaze processor. The audio files are stored in a raw I2S format and is output to an external speaker.

2.2 Specifications

Features and Functions

- Real time video processing, involving object detection and laser point detection
- Tracking identified objects with the laser pointer
- “Firing mechanism” consisting of sound effects triggered when an object has been targeted
- Live feed showing turret’s view of the environment

Peripheral Requirements

- Camera
- Monitor
- Speakers
- Laser pointer
- 2 Servo motors with a pan/tilt bracket and a power supply

2.3 System Overview

The video processing IP has two functions. Firstly, it detects the current position of the laser. Secondly it detects the current position of objects. As shown in our process diagram in Figure 2, both functions are performed in parallel. The process diagram shows the stages of computation and their dependencies. The feedback in the process comes from the fact that we effect a change on the environment by moving the laser. A VDMA moves the incoming video stream over the bus to the video processing IP (see block diagram in Figure 3) and extracts the processed output. The IP outputs the co-ordinates of the objects it detects as well as the laser position. This information is polled by the MicroBlaze over the AXI Lite interface. The MicroBlaze then chooses one of the objects to set as the target.

The system has three main outputs: sound effects, HDMI video output, and control signals for orienting the laser. The MicroBlaze processor polls the positions of the target and the laser and uses that to calculate the relative change in position required. It then generates the appropriate PWM signals for controlling the motors. The MicroBlaze also reads sound files off the SD card and outputs control signals to the audio controller.

There are a number of debug features. The SD card has test images that can be fed to the IP to test the IP under a controlled condition. HDMI video output is included to show the stages of object detection. A UART based terminal interface is used to communicate with the MicroBlaze in order to issue commands and view debug messages.

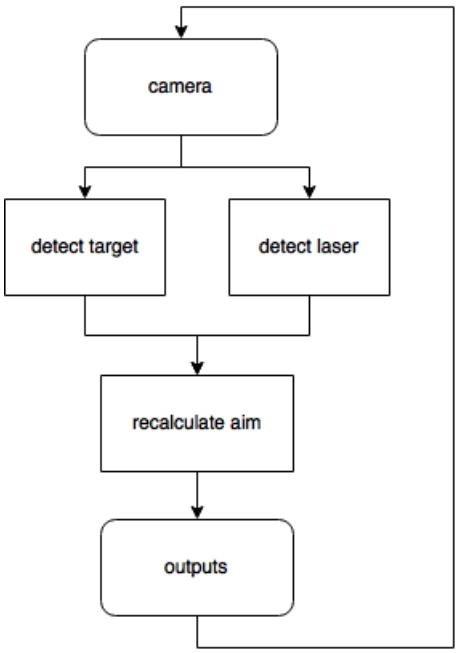


Figure 2: Process diagram, showing stages of calculations.

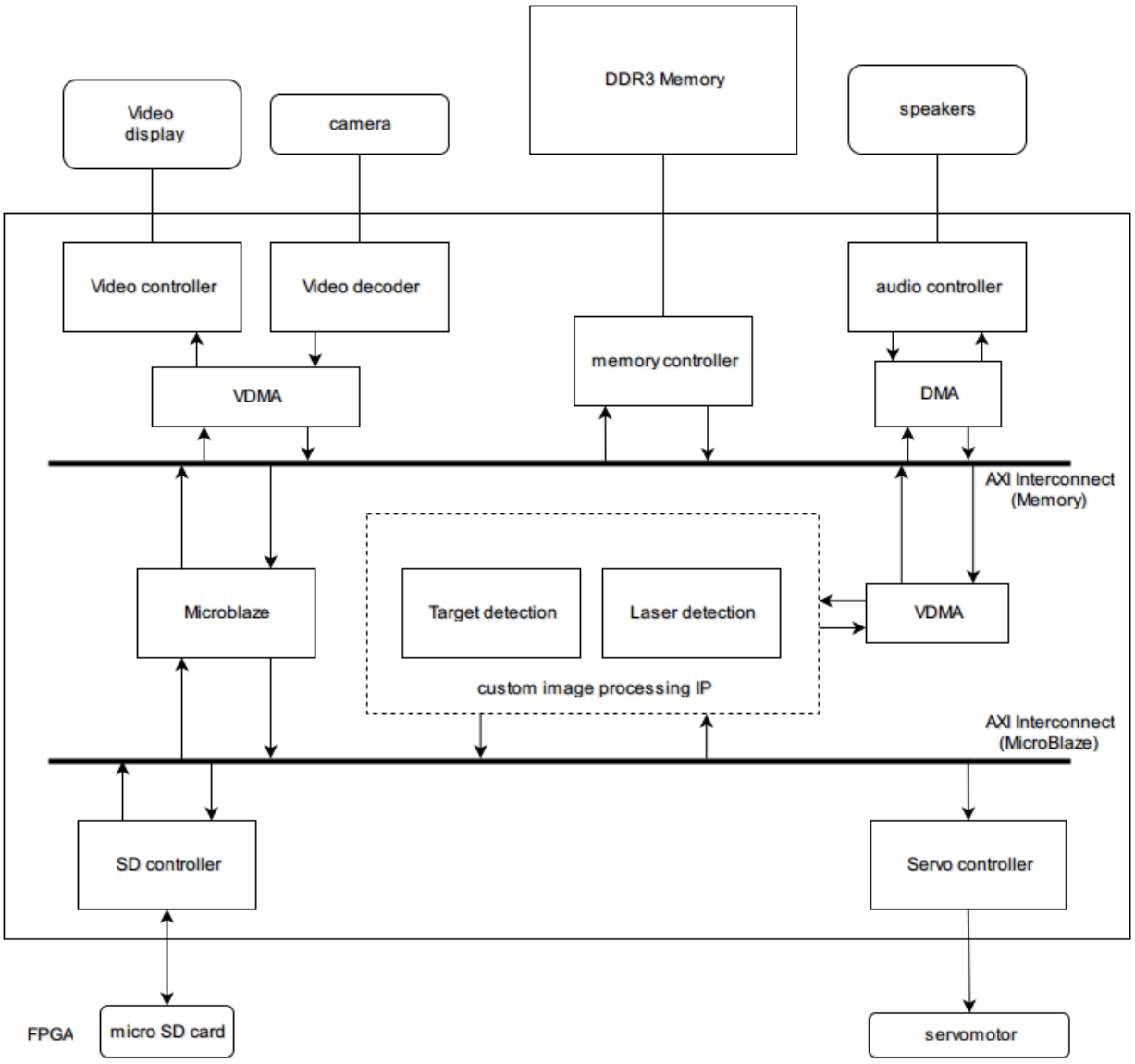


Figure 3: Block diagram of major components and connections.

3 Block description

3.1 Image Processing IP

The image processing IP receives and transmits video data using the AXI Stream protocol. This works through a series of ready/valid handshakes that occur at both the IP RX side (memory map to IP stream) and TX side (IP stream to memory map). Video data is streamed from DRAM using a VDMA, which sends a frame one line at a time. The input data is stored in a circular FIFO. In addition to this RX FIFO, there is also a TX FIFO at the output that receives the processed image pixels and transmits them upon request from the VDMA. Internally, the object and laser detection modules only pull from the RX FIFO and load the TX FIFO when they are empty or full respectively, and when these modules are enabled.

The AXI Lite protocol was used for communicating with the MicroBlaze. The full register list is available in Appendix A.

3.1.1 Object Detection

Our aim was to identify objects and be able to recognize friends and foes. The objects were to be on a plain monochrome background and be non overlapping. This task was broken down into the following smaller tasks: (1) locate non-background objects (2) perform a binary classification of friend or foe.

This was performed in the following steps:

- grayscale
- edge detection
- flooding
- connected components analysis
- data extraction

The full top level diagram showing all these components is available in Figure 11 in Appendix B.

Grayscale Grayscale was calculated using the ITU-R Rec. 601

$$I = 0.299R + 0.587G + 0.114B$$

Image Filter Image filters can produce interesting effects, such as blur, sharpen, emboss, or outline. This diverse range of effects is achieved with a very simple and compact computation. It is performed by convolving an *image kernel* with an input image to produce an output image. The kernel is a small matrix that defines the resultant pixel as a weighted

summation of the pixels in the window surrounding it. The output is computed by convolving at each input pixel a window of the input with the kernel.

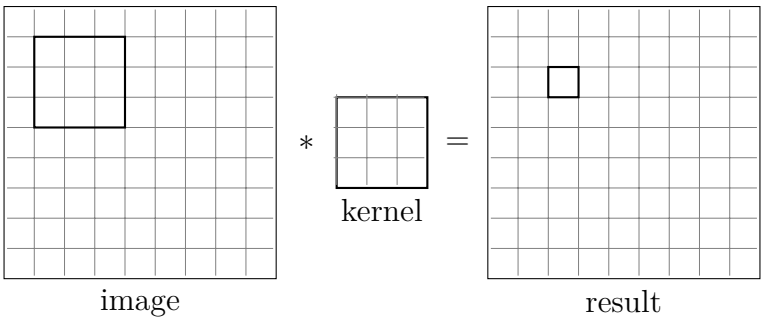


Figure 4: Image kernel convolution.

Sobel The Sobel operator is an image filter designed to extract edges by computing the derivative at each pixel. Gradients correlate well with edges because an even surface contains pixels that are very similar to each other whereas pixels along edges show a high level of difference, or gradient, along the direction of the edge.

$$G_y = \begin{bmatrix} -3 & -10 & -3 \\ 0 & 0 & 0 \\ +3 & +10 & +3 \end{bmatrix} * A \tag{1a}$$

$$G_x = \begin{bmatrix} -3 & 0 & +3 \\ -10 & 0 & +10 \\ -3 & 0 & +3 \end{bmatrix} * A \tag{1b}$$

$$G = |G_x| + |G_y| \tag{1c}$$

$$q = \begin{cases} 1 & G \geq \text{threshold} \\ 0 & G < \text{threshold} \end{cases} \tag{1d}$$

Performing the computation requires access to a 3 x 3 neighbourhood of pixels. This requires two line buffers to access the pixels from the previous two rows, shown in Figure 5. Once two rows have been filled up, the output can be computed at a rate of one pixel per clock cycle.

Flooding When the edge detection was tested in hardware, two main issues arose. First, there were many small speckles of noise. Second, the outline appeared grainy, like in a sketch where some of the paper still shows through a penciled outline. A filter was applied that takes a population count of the window, by applying an image kernel of all one's. Then, if

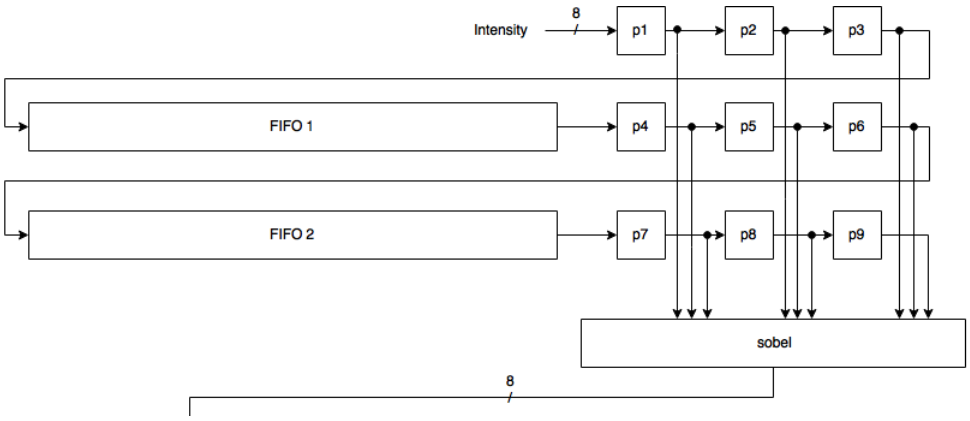


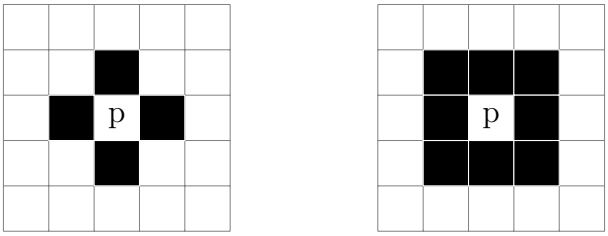
Figure 5: Generating the pixel array for Sobel computation.

this count meets a threshold the input pixel is accepted as part of the outline, otherwise it is rejected and marked as part of the background.

$$\text{count} = \begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} * A \tag{2a}$$

$$q = \begin{cases} 1 & \text{count} \geq \text{threshold} \\ 0 & \text{count} < \text{threshold} \end{cases} \tag{2b}$$

Connected Components Analysis Connected components analysis is a method to extract information about objects in a *binarized* image. *Connectivity*, illustrated in Figure 6, refers to pixels that share (1) edges for 4-connectivity or (2) edges or vertices for 8-connectivity A *binarized* image is an image where there are only two pixel values, which represent background or non-background. This method was used to locate the different objects in view.



(a) 4-connectivity (b) 8-connectivity

Figure 6: 4- and 8- connectivity

The analysis works by scanning the image in raster order, scanning each row from left right from the top to the bottom of the image. At the time that pixel p is processed, the

A	B	C
D	p	

Figure 7: Neighbourhood of input pixel p. Labels A through D are the labels of the pixels that have already been processed through connected components analysis and p is an input pixel from the binarized image.

pixels before it have already been labeled, as shown in Figure 7. The label for p is selected based on the labels A through D.

There are several possible scenarios:

1. p is a background pixel.

   It is not given a label.

2. p is not a background pixel and A through D are background pixels.

   It is the start of a new object, so it gets a new label.

3. p is not a background pixel, and among A through D there is a single label.

   It is part of this object, it copies that label.

4. p is not a background pixel, and among A through D there are two or more labels.

   These labels need to be merged. p is assigned the smallest of these labels.

Scanning in this fashion, there comes a point where two regions may need to be merged. Consider a 'U' shape. The two arms of the 'U' will be labeled separately, and it is not until the bottom right of the 'U' that the two labels appear in the same neighbourhood. At this point, the data for the two objects can be merged. This is valid since the entries are just a summation of data points for each pixel considered to be part of the object.

Bailey and Johnston [2] illustrate a single pass connected components analysis implementation on an FPGA. This approach was followed with some simplifications. The main components are shown in Figure 8. There is a label selection module, a merge table, and a data table.

The first two stages select the label for the current pixel. The first stage is label selection. This block is all combinational logic. The next stage reads from the merge table. It looks up the selected label in the merge table which is in BRAM to see if this pixel really belongs to another label. At the end of this stage, we know the correct label for this pixel.

The next two stages update the data for this label. The features being extracting from the objects are its moments, up to the third order. In the third stage, the data for the resolved label is read out of the table, updated and in the fourth stage written back.

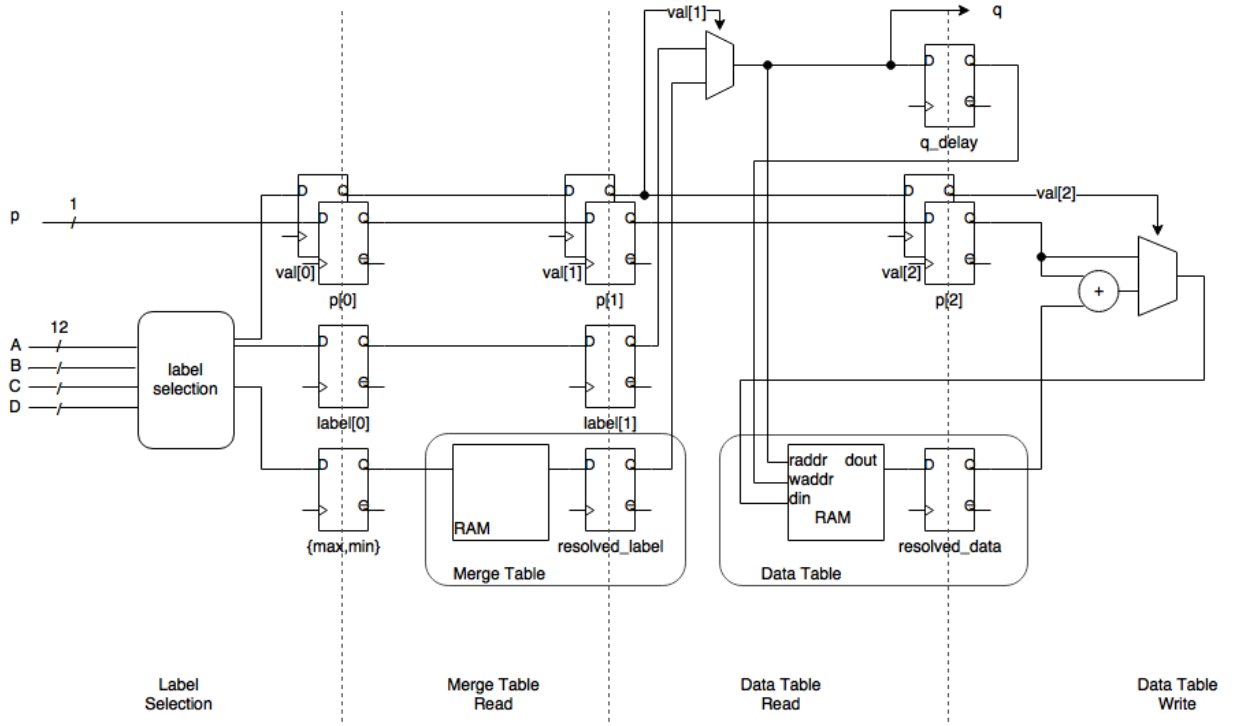


Figure 8: Connected components pipeline.

Feature Extraction In the one pass method described by Bailey and Johnston [2], data is extracted on the fly as the objects are being labeled.

Location Locations of objects were found by finding the centre of mass of the outlines. As the objects were being labeled, their moments up to the third order were accumulated in a data table.

The locations were then extracted using Equations (3). Due to the nature of division in hardware, this was left to the MicroBlaze. When it was attempted in hardware, it resulted in an astounding 75 ns delay on a single path.

$$area = m_{00} \tag{3a}$$

$$\bar{x} = \frac{m_{10}}{m_{00}} \tag{3b}$$

$$\bar{y} = \frac{m_{01}}{m_{00}} \tag{3c}$$

Debug Features In order to facilitate debugging, each stage of image processing can be displayed to video output. In particular, the connected components labels are coloured in a colour-by-numbers fashion so the different outlines can be easily observed. Internally, they

are labeled $1, 2, 3, ...$, which would appear as various shades of grey if displayed directly. The output mode is set by the MicroBlaze in a control register. Table 1 enumerates the output modes and corresponding codes.

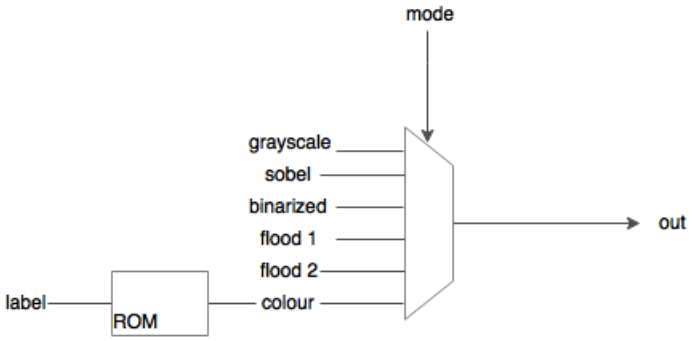


Figure 9: Output modes

i	mode
0	pass
1	gray
2	sobel
3	thresh
7	flood1
8	flood2
4	cc
5	color
6	laser

Table 1: Table of output modes and designated codes.

3.1.2 Laser Detection

Laser tracking is done through detection of a high concentration of red pixels. A filter subtracts the maximum of the B and G channels from the R channel. If this value exceeds a certain threshold, it is deemed to be red. If enough of these red pixels are found consecutively, the IP has found the object. Detection based on locating the brightest point was also explored. However, both of these algorithms suffered from the flaw that the laser would often be lost when it traversed over a white area. This is because the laser generally appears as an intense red-white to the camera. Similarly, checking only the red channel resulted in white objects also being picked up by the filter.

3.2 Turret Control

The turret consists of two servo motors mounted on a pan-tilt bracket. The servo motors are controlled by two pulse width modulated (PWM) signals that set the angles of the motors. The motors require a 20ms period pulse that is generated with counters that counts up to 2,000,000 on a 100MHz clock. The allowable pulse widths vary between 1ms and 2ms, corresponding to $-90°$ to $+90°$ respectively. These are generated with counters of values between 100,000 and 200,000, which are set by the MicroBlaze. The laser is simply controlled by an enable received from the MicroBlaze. All signals are output through a Pmod port and are buffered and level-shifted before driving the actuators and laser.

3.3 Video Subsystem

The Video input/output subsystem is an adaptation of the HDMI demo project provided by Digilent [4]. It has been modified to work with our block design and our code structure. These modifications include backporting the project to Vivado 2015.1 (it was originally 2015.3). Code-wise, major changes include changing some of the interrupt handling code to better integrate into our code and changing the framebuffer size for 720p video. Software was also added to simplify the interface between the existing video capture/display drivers and the rest of the code.

3.4 Audio Subsystem

The blocks used for the audio I/O are adapted from the DMA Audio Playback/Record demo provided by Digilent [3]. It has been modified to integrate into our block design. These modifications include backporting the project to Vivado 2015.1 (it was originally 2015.3). In terms of software, there were changes to the interrupt handling code and a new interface was created to make integration with the other code more streamlined. In addition, instead of using GPIO signals to trigger sounds and recording, these actions are triggered using software hooks. Instead of using a fixed audio buffer in memory, sound files may be loaded from an SD card into memory and played by specifying a base address and length.

3.5 MicroBlaze Subsystem

The MicroBlaze subsystem consists of 4 main components, including memory management, the SD card system, debugging and control facilities, and turret control.

3.5.1 Memory Management

In the block design, there are two VDMAs and one DMA. One VDMA is responsible for the video input from an external source and output to a monitor and it was originally borrowed

from the Digilent HDMI demo project [4]. It is configured to write to one frame buffer (`fb0`) and read from another (`fb1`). Aside from the change in framebuffer configuration, this VDMA is more or less the same as the one in the demo.

The other VDMA is set up to interact with the custom image processing IP. It feeds `fb0` into the IP and reads the IP output into `fb1`. Unlike the other VDMA, it is not configured with an *fsync* signal and it is set to copy a single frame of data in either direction (frame counters are set up to trigger after one frame worth of data). This frame counter set-up includes the use of interrupts, for which handlers may be registered at run-time. These callbacks are set-up to keep track of the status of the IP so that the rest of the system can wait for the IP to finish working. The VDMA can also be configured to not read the output of the IP and not write to `fb1`.

The final DMA is used to interface the audio codec with DRAM. It originates from the Digilent Audio DMA demo project [3]. The software has been modified so that a callback may be registered. This callback is used to inform the main code that audio transfers have been completed.

In addition to the input and output frame buffers for the webcam video, we also needed to allocate space for the various sound and image files we were using. In our use case, we never needed to resize, modify, or remove files from memory, so a complex memory allocation algorithm such as that used by `malloc()` would be overkill. Instead, we implemented a simple contiguous allocation scheme, where we would place the file buffers one after another in a large region in DDR3 memory. A separate fixed-size table was used to keep track of the location, size, and status (loaded or not) of each file.

3.5.2 SD card communication

Micro SD cards support two different interfaces: native SD communications and SPI. The native SD protocol is not publicly available, and the IP provided in Vivado requires a purchase. As a result, we opted to use the more obscure, slower SPI protocol which is publicly available. To do this, we used the AXI quad SPI IP available with Vivado. Low level code for initializing, reading from, and writing to the SD card were written using the register interface to the SPI IP, rather than the Xilinx provided drivers. This was necessary since the Xilinx drivers do not provide the low-level control over the SPI bus needed for the protocol to work.

In order to implement the FAT32 filesystem layer, we made use of the FATFS library, available open-source under a BSD-like license. This library uses the low-level IO functions we implemented using the SPI SD protocol, and presents a unix-like file IO API. If the SD card is formatted using a FAT16/FAT32 filesystem, we can read and write files using an embedded variant of the `fopen()`, `read()`, and `write()` system calls.

3.5.3 Debugging and control

We implemented a terminal-like interface to the MicroBlaze using the AXI UARTLite IP provided within Vivado. In order to avoid having to poll the device, we set up interrupts. We configured this IP to throw an interrupt whenever a character was received from the attached computer. Our system supports string commands (for example "runip"), which gets executed when the user hits the enter key. This is implemented using an internal buffer in which we place characters we obtain during the interrupts. Similarly, we also support commands which accept arguments, for example supplying a threshold value for the Sobel operator. Using this command interface, we can dynamically configure the IPs within the design, as well as read debug information from various IPs.

As mentioned earlier, we implemented debug features allowing us to view visually the output of various stages within the video processing IP. In addition, we also annotated this output with more information useful for debugging the output of the IP. This includes the location, area, and moments of objects detected by the IP, as well the location of the laser according to the IP. This allows us to easily view this information without having to cross-reference it with the textual debug output provided through UART. It also provides information that helps guide the tuning of the IP parameters, including the various thresholds.

3.5.4 Turret control

As mentioned earlier, we implemented a basic IP for configuring the PWM signals sent to the pan and tilt servos. In addition, we wrote a software module to abstract this low-level interface in order to ease turret control. This interface allows us to set and move the motors using angular degrees rather than PWM values. Using this interface, a simple manual turret control mode was written, where pushbuttons allow us to move and fire the turret.

We also wrote an automated turret control module, that uses the output of the video processing IP in order to try to move the laser to point at a target. Given the pixel-positions of the target and the laser, we determine the angle at which we need to pan and tilt in order to bring the laser point closer to the target. Due to the fact that we do not have a direct translation between the pixel position and angle (since the camera distance is variable), we do not move immediately to the target, but rather incrementally get closer. Once the laser and target are within a certain threshold of each other, we fire by playing a sound file.

4 Outcome

4.1 Results

Our project successfully shoots at the largest non-background object in its view. It is limited in a few ways. Our turret can only operate in dark rooms on non-reflective surfaces. Further, it cannot shoot at objects that are too light due to the fact that we are projecting the objects

using a data projector. The light of the object obscures the red hue of the laser. The turret loses track of where the laser is because it appears simply as a bright white spot.

If we could give our past selves some advice, the first would be to get HDMI passing through the AXI IP as soon as possible. The second would be to acquire all peripherals as the next priority and feed realistic test data to the software prototypes. The third would be to consider simplifying the project sooner to perhaps discriminate between objects by colour.

Although we initially wanted to determine objects by shape, we could have tried identifying them by colour. This would have allowed the MicroBlaze interface to be integrated and tested sooner. It could have been done relatively easily and it would have made our demo that much more impressive. We did not initially opt for identification by colour because it seemed too easy, but as we started slipping from our project schedule, we could have considered modifying the requirements.

A potential alternative to using a visible spectrum laser is to use an IR laser. This would mitigate the issue that we have where the laser and object detection conflict with each other. However, this would be expensive and also require a second HDMI input stream.

4.2 Improvements

We would like to make our project more robust and be able to operate under normal lighting conditions and on plain, but not pristine surfaces. Currently, if the paint on the wall is too shiny, or there are too many chips or smudges on the wall, the turret does not work very well. In order to do this, it would likely be necessary to implement more sophisticated noise rejection, such as background subtraction or variance normalization.

Secondly, we would like to complete our goal of discriminating between targets and non targets by implementing our comparison method using moment invariants [5].

5 Project Schedule

	Planned	Actual
1	Select video processing algorithm. Acquire components for turret. Basic UART command interface. Working SD SPI interface.	Experimentation done with OpenCV. Turret construction completed. Single char commands supported. Low-level SPI data access done.
2	Prototype targeting algorithms in SW. Play basic audio from codec. Read and write files from FAT32 SD card. Assemble turret with circuitry.	Selected algorithms implemented in SW. Partially functional Audio design done. FAT32 file read/write IO done. Turret Control IP implemented.
3	uB has full control over turret. Live video feed forwarding. Dummy video processing IP. Testbench for video processing IP.	uB can use control IP to move turret. Audio and UART improvements. Initial tb with bitmap read/write done.
4	Target detection working in simulation. uB aim laser at arbitrary location. Hardware testing of laser detection. Improve laser detection IP.	Basic HDMI pass-through working. HW and code version-controlled. Edge detection done in simulation.
5	Hardware testing of object detection IP. uB tracks targets and laser. Write projection loop for demo. Improve IP performance.	Command-line UART interface. Can play sounds from SD card. CC labeling partially done in simulation. HDMI integrated into project
6	Refine targeting and firing software. Play sound files upon lock on target. Improve IP performance.	Manual turret mode. Memory management for loaded files. CC labeling done in simulation. Edge detection done in hardware.
7	Integration and debugging.	Pass images from SD card into video IP. Turret uses IP to centre laser on screen. Can pass arguments to UART commands. Object locations done in hardware.
8	Integration and debugging.	Turret uses IP to point laser at target. Improved robustness of object detection.

6 Design Tree

Directory	Description
bd/	Tcl scripts for generating block designs.
constraints/	Pin and clock constraint files.
doc/	Helpful documents for understanding our code and hardware.
hdl/	Misc. other HDL files, usually block design wrappers.
ip_repo/	A managed IP repository for our custom IPs.
video_design/	The script for generating the vivado project.
video_sdk/	The microblaze code for running our project.

Users wishing to run our project would run the project generation script in video_design. The design can then be fully compiled, and the resulting bitstream programmed onto a Nexys Video board. The code in video_sdk can be run on the microblaze in order to implement our automated turret.

Our work is available at the following Github repositories:

- `https://github.com/ngemily/G3_Turretmaster5000`
  Complete project.
- `https://github.com/ngemily/opencv-object-detection`
  Software prototype of object detection.
- `https://github.com/ngemily/sample-bmp-tb`
  Sample test bench for image processing blocks.

7 Tips and Tricks

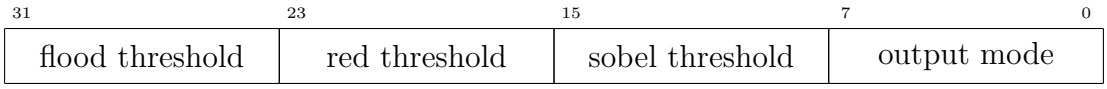
Read the tips on the course website and follow them!

A IP Documentation

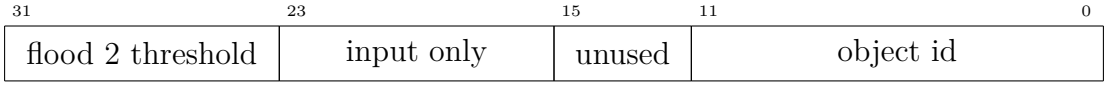
A.1 Register list

Table 2: Table of AXI registers. 'w' indicates read and write register, 'r' indicates read only register.

address	r/w	description
5'h00	r	obj_m11
5'h01	r	obj_m12
5'h02	r	obj_m21
5'h03	r	rx_read_pointer
5'h04	r	tx_write_pointer
5'h05	r	tx_read_pointer
5'h06	r	rx_fifo_track
5'h07	r	tx_fifo_track
5'h08	r	mm2s_tready
5'h09	r	mm2s_tvalid
5'h0A	r	s2mm_tvalid
5'h0B	r	s2mm_tready
5'h0C	w	ctrl_reg1
5'h0D	w	frame_resetn
5'h0E	r	laser_xy
5'h0F	w	ctrl_reg2
5'h10	r	num_labels
5'h11	r	obj_area
5'h12	r	obj_x
5'h13	r	obj_y
5'h14	r	obj_m20
5'h15	r	obj_m02
5'h16	r	obj_m30
5'h17	r	obj_m03



(a) Control register 1.



(b) Control register 2.

Figure 10: Control register layout.

B Object Detection Block Diagram

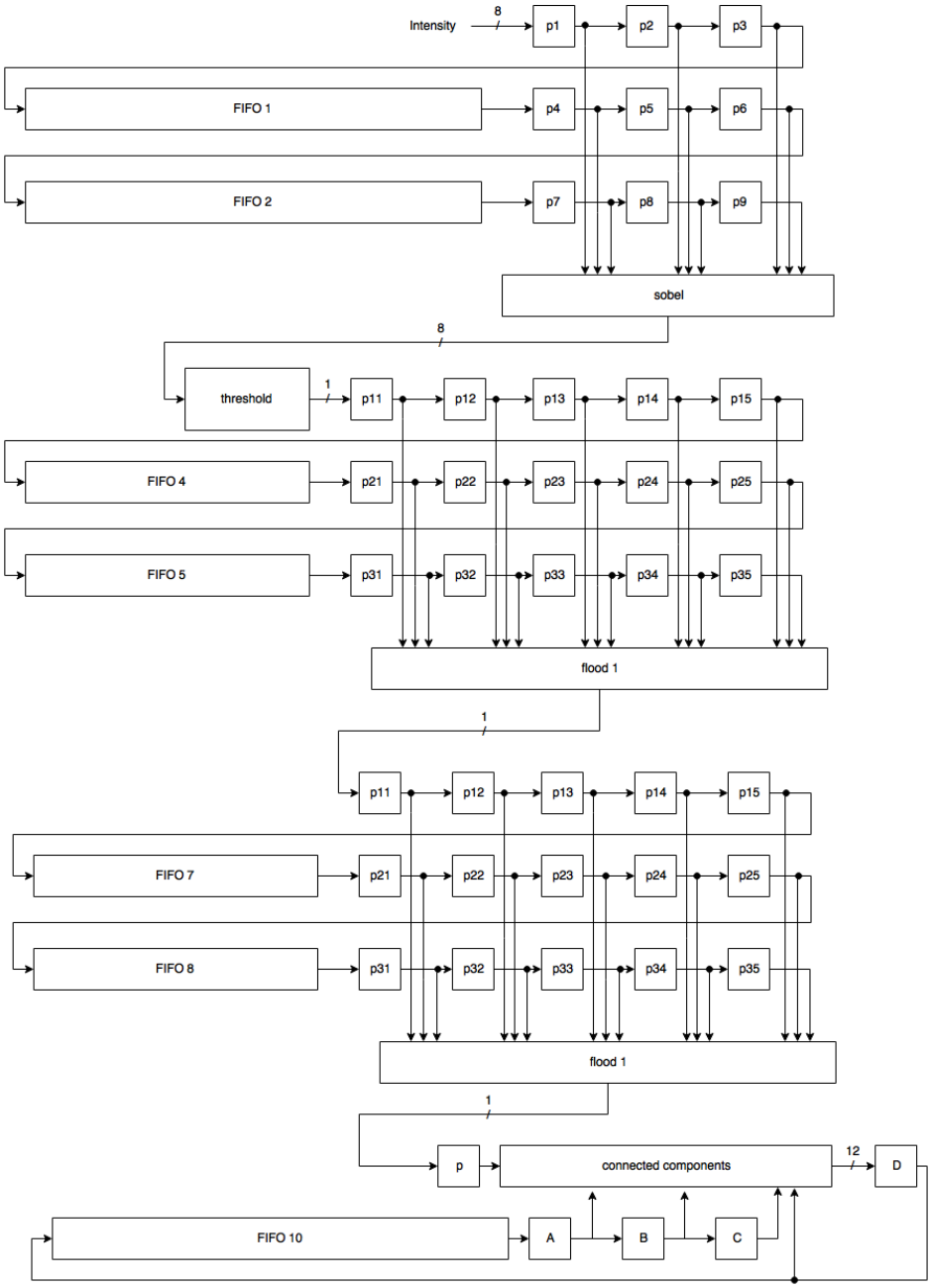


Figure 11: Object detection top level block diagram, highlighting line buffer structure.

References

[1] *Autonomous Sentry Turret*. 2016. URL: http://www.instructables.com/id/Autonomous-Sentry-Turret/.

[2] D.G. Bailey and C.T. Johnston. "Single Pass Connected Components Analysis". In: *Proceedings of Image and Vision Computing* (2007), pp. 282–287.

[3] Digilent. *Nexys Video DMA Audio Demo*. 2016. URL: https://reference.digilentinc.com/nexys-video:dmaaudiodemo.

[4] Digilent. *Nexys Video HDMI Demo*. 2016. URL: https://reference.digilentinc.com/nexys-video:hdmi.

[5] Ming-Kuei Hu. "Visual pattern recognition by moment invariants". In: *IEEE Trans. Inform. Theory* 8.2 (1962), pp. 179–187. DOI: 10.1109/tit.1962.1057692.