Simultaneous Personnel Localization and Mapping

A Major Qualifying Project Report Submitted to the Faculty of
Worcester Polytechnic Institute

In partial fulfillment of the requirements for the
Degree of Bachelor of Science in Electrical & Computer Engineering

By:

Georges Gauthier
John DeCusati

July 22, 2016

Advisor:
Professor R. James Duckworth

Contents

List of Figures

Abstract

The overall goal of this project is to create a device capable of generating detailed maps and imagery of an area in real-time. This device will rely on an FPGA, and will use image processing algorithms capable of detecting, localizing, and tracking human beings. The device will gather imagery from the visual light and infrared-spectrums, as well as localization data and distance measurements from an IMU and a rangefinder. Along with gathering and processing data, the device will also serve as a long-range wireless access point, and will be able to transmit all generated maps and imagery in real-time. A major deliverable of this project is that the transmitted data will be fully processed, allowing it to be viewed remotely on less-powerful, mobile devices.

This device will be especially useful for first responders. It is intended to be mounted on a small remote control vehicle, allowing any connected user to wirelessly traverse dangerous and remote locations in search of people in need. Since this device transmits data in real-time, it will be able to provide first responders with an accurate representation of not only a 2-D floor plan of an area such as a building, but also where any people are located. An anticipated use of this device would be in the event of a building in danger of collapsing. Since it would be dangerous to physically enter the building, first responders could locate any people trapped inside and find the fastest route to them using the wirelessly transmitted floorplan. The first responders would also be aware of any dangers in their way by making use of the real-time augmented video stream. This video stream will consist of image data with overlaid with object indicators and location information on any human beings detected by the image processing algorithms.

1 Introduction

Currently there are many applications that rely on a simple video camera setup in order to gather information on remote and inaccessible locations. Although this is an effective strategy for simple surveillance, it is limited in many ways. Using current imaging and sensor technology, it is possible to gather camera images and 3D depth information on a given area as both a cost-effective and information-rich alternative to using a camera module on a device. If a product were to be created that gathers this information as a replacement to using a standalone camera module, it would be possible to use high speed data processing techniques in both hardware and firmware that would allow for the creation of an augmented real-time video feed.

This type of technology is known as Simultaneous Localization And Mapping, or SLAM. The purpose of SLAM is to compute the location of an agent within its environment, and allows for the creation of self-aware robot systems that are able to respond to their surroundings. SLAM is a common area of research in the image processing and high-speed computing field, and has been applied mainly to autonomous vehicles. We would like to propose the creation of a SLAM-like system that is capable of monitoring and mapping its environment in real-time, as well as detecting and localizing objects, such as human beings. For the purposes of our project, we would like to define our desired objective as Simultaneous Personnel Localization And Mapping, or SPLAM.

A device that is capable of both mapping its surroundings using SLAM and performing human detection in real-time would be applicable to many different fields. We are especially interested in creating a proof of concept sensor suite capable of performing these tasks that can be added to existing robotic systems as a stand-in replacement for a video camera. This type of technology would allow for people such as firefighters or first responders to wirelessly traverse dangerous and remote locations in search of people in need. We envision our sensor suite being able to process data so that its users would be provided with a 2D ?floorplan? of the area being traversed by the sensor suite, as well as an augmented video feed with

imagery containing indicators for any detected human beings in the area.

One type of technology that would be useful for performing the high speed data processing necessary for SPLAM is a Field Programmable Gate Array , or FPGA. FPGAs pose several advantages over using standard computing or microcontroller technology for real-time data processing, as they have the ability to manipulate digital information in parallel using hardware only. This allows for extremely high-speed performance, as calculations can be run in parallel and are only dependent on their data inputs as opposed to waiting for specific tasks or scripts to run on a microcontroller or computer software interface.

Although FPGA technology is highly applicable to performing SLAM-like tasks due to its high speed, there are currently few existing commercial products that use FPGAs for the purpose of performing SLAM. Most current SLAM implementations rely on the use of a sensor suite connected to a computer or system on chip (SoC) computing device that performs data analysis using software or a real-time operating system (RTOS). This means that data must first be collected by a sensor suite, and then transferred to an external computing device that is only capable of processing it serially based on its arrival time. Although this type of setup is acceptable for performing real-time situational awareness analysis, we propose that an embedded, FPGA-based SPLAM device would be a much more elegant and higher-speed solution.

2 Background

A major concern with real-time image processing, especially in first responder situations, is speed. Because FPGAs have the ability to process data in parallel, they are ideal for this type of application. Using an FPGA for this system will enable all data inputs to be processed at the same time, thereby dramatically increasing throughput speed. Dealing with each input separately makes it easier to combine everything together, especially because each component functions at a different clock speed. Also, since everything is running in parallel, more cameras can be added to the system to increase the field of vision of the device without introducing any latency in the system, as long as enough memory is available.

SLAM is a widely expanding field with much potential for improvement. One application of such a system is a proof of concept of camera-based SLAM systems, presented by Andrew Davison of Oxford University in a research paper entitled "Real-Time SLAM with a Single Camera" [1]. This system is handheld and relies on a computer using a 2.2 GHz Pentium processor connected to a single camera and laser rangefinder. The system requires prior knowledge of the area being analyzed before it can successfully localize and map. It implements edge detection, but is limited to the narrow field of vision of the rangefinder, so it is only able to map an object directly in front of it. This system carries a latency of around 33 milliseconds. An output frame of the device is shown in Figure 1.

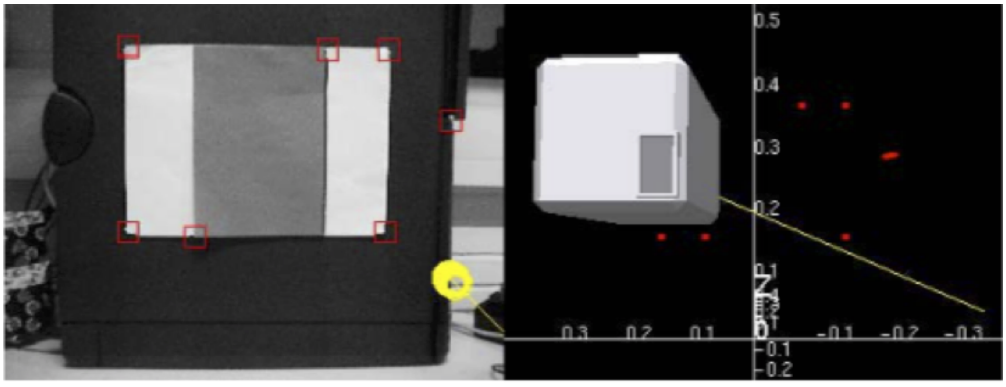

Figure 1: Real-Time SLAM with a Single Camera [1]

The frame on the left in Figure 1 is the video feed with 6 points of a paper target input as

prior knowledge, along with successfully marked identifying features (marked as red squares), and another identifying feature that is not marked for measurement (marked by a yellow circle). The frame on the right is a localization graph displaying the positions of all red squares.

A more commercial device similar to our concept is Serveball?s Squito^{TM} [3]. Squito is a wireless, throwable, 360° panoramic camera that implements target detection to stabilize the video feed from its many cameras. It is shown in Figure 2 below.

Figure 2: Serveball's Squito [3]

Squito utilizes a microprocessor receiving input from cameras, as well as orientation and position sensors in order to transmit a real-time stabilized video of its adventure. The device is still in the prototype stage and is receiving interest from first responders. The image in Figure 3 shows the input from the Squito's four cameras on the left, and the corresponding stitched output on the right.

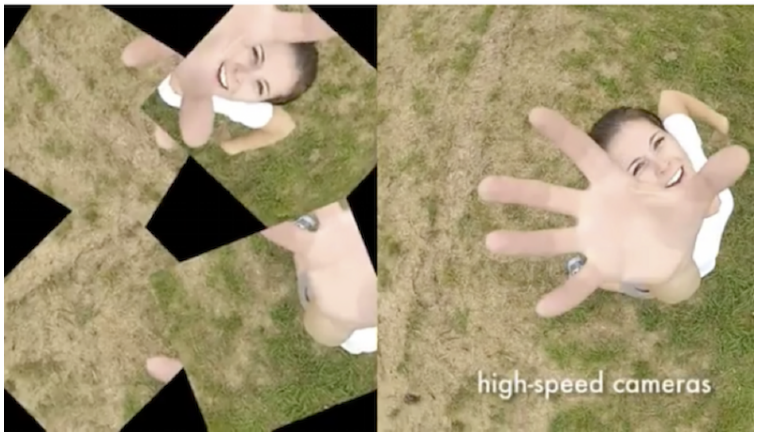

Figure 3: Serveball's Squito Input and Output [3]

By using multiple camera sensors in a sensor suite, it is also possible to determine depth information from corresponding images of an area. This technique is known as stereo imaging, and the process of gathering depth information from a pair of stereo images is known as disparity mapping. University of Bologna researchers Stefano Mattoccia and Matteo Poggi have worked to implement a real-time disparity mapping algorithm on an FPGA, and an example of a stereo image disparity is shown in Figure 4 below [4]. Using their stereo vision algorithm, the researchers are able to generate real-time image data showing the relative locations of objects within an image frame using color gradients. Based on this depth information, it is also possible to detect objects located within the field of view of the stereo imaging system, as shown in Figure 4. An implementation similar to this would be extremely useful in a SLAM-like system, as it would allow for the localization of objects and creation of 2D "floorplans" of an area in real-time using only two camera sensors.

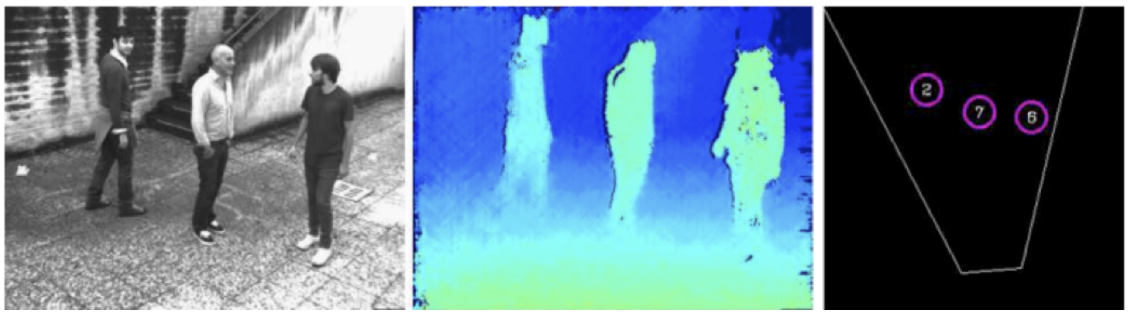

Figure 4: From Left to Right: Original Image, Disparity Map, Object Detection Results [4]

Many security systems implement human detection and human body tracking in order to increase their effectiveness. These devices process real-time images in order to identify human characteristics, and are limited to the field of vision of a stationary or rotating camera. An example of this type of system is explored by the Mitsubishi corporation in a research paper entitled "Human Body Tracking by Adaptive Background Models and Mean-Shift Analysis" [6]. The paper explores a stationary image processing system implemented on a PC platform with a 1.8GHz processor that yields a maximum processing time of 100 milliseconds. An output frame of the system is shown in Figure 5 below.

Figure 5: Human Body Tracking by Adaptive Background Models and Mean-Shift Analysis [6]

Our proposed device will combine the ideas of the four systems examined. It will be able to simultaneously localize and map an area, as well as implement human detection algorithms. The device will be capable of generating real-time 2D maps of any area it has traversed with humans' locations labeled, and an augmented video feed of what the device is recording with any humans' positions marked.

In order to successfully implement this system, we propose the creation of a device that will rely on two stereo cameras, a laser rangefinder, and an inertial measurement unit (IMU) as its sensor suite, as shown in Appendix Item 2. Limitations of previous art have been in their ability to combine human detection with real-time localization and mapping of a

large field of vision. Little to no existing commercial products are also capable of processing their gathered data locally and in real-time, with their gathered data usually requiring post-processing on external computing devices. Stereo cameras will allow our device to calculate disparity, just as human eyes do. Although disparity is useful for localization, it is not enough for accurate mapping because it only accurately provides the relative distance between objects. The inclusion of a rangefinder will allow for precise base distance readings, and an IMU will be used to spatially reference all gathered data. All of this data will be combined with the disparity maps and image data in order to create flawless localization and mapping. All time-dependent processing required for the device will be mainly done in parallel using hardware on an FPGA. An overall functional block diagram of our intended implementation is shown in Figure 6 below.

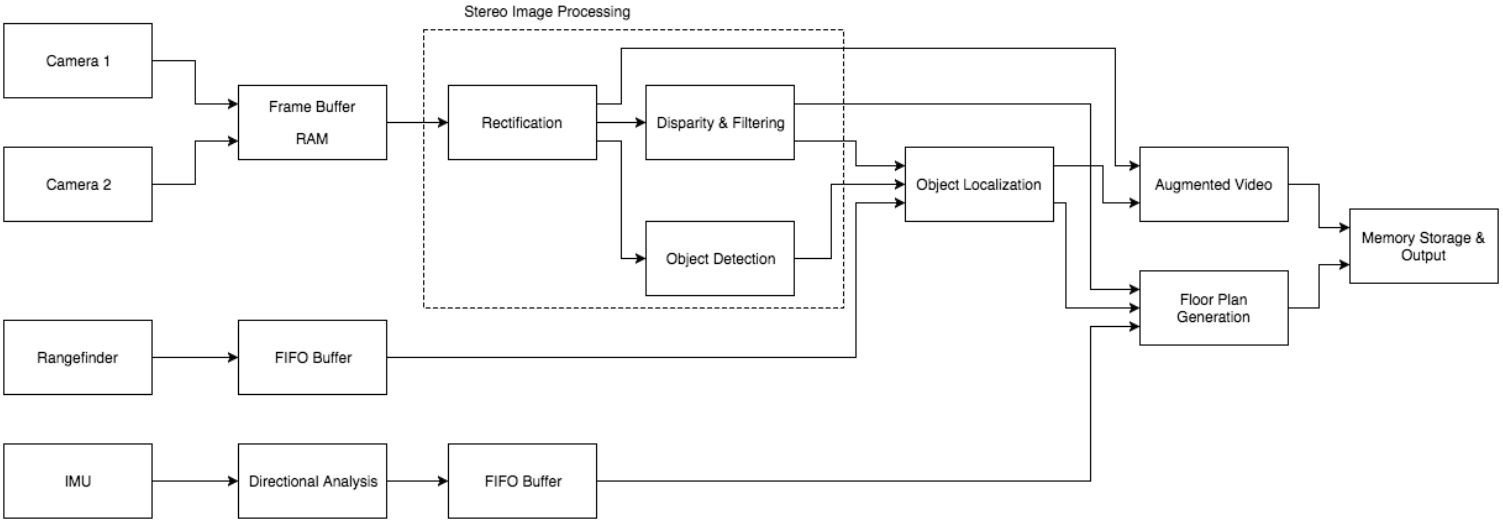

Figure 6: Functional Block Diagram

Most applicable previous camera-based systems have also focused on object detection from a stationary point, or edge detection from a mobile platform. Our project aims to combine these concepts, by creating a mobile device that detects people which will be especially of use in many first responder situations. In addition, this device will receive data from the visible light and infrared spectrums in order to identify people quickly and accurately in a way that has not been previously implemented.

As our research has progressed over time, our project objectives have continually evolved. We originally envisioned the creation of a device that used laser rangefinders to create 3D maps of its surroundings, similar to that of a Carnegie Mellon University device created in order to volumetrically map abandoned mines [7].

As our research progressed, we believed that we could use a visual light and thermal imaging camera set to gather information on an area, and supplement that data with IMU and rangefinder readings in order to produce detailed maps of our sensor suites? surroundings. Eventually we came upon the concept of disparity mapping and generating depth information from image data, and decided that we would again like to shift the overall setup of our device to rely mainly on stereo image data. Due to our overall budget and the resources that have been made available to us, in the coming terms we plan to use an electronic rangefinder, IMU, and stereo camera pair to generate real-time SPLAM video and floorplan information. Although we were also originally planning on including a thermal camera in our sensor suite as well, we have decided to eliminate the module in favor of higher quality cameras due to its prohibitive cost, low resolution, slow sampling rate, and small field of view. More information on this decision can be found in Appendix item 4.3.

3 Methodology

3.1 Camera Testing

After obtaining two of the MT9V034 cameras chosen through the process referenced in Appendix item 4.3, several steps were taken to obtain test data from each camera. These steps are outlined in the following sections.

3.1.1 Camera Operation

In order to gather working images from each camera module, we first needed to understand what circuitry our camera module breakouts contained so that we could interface with them. MT9V034 camera breakouts were purchased from Leopard Imaging Inc. Although these camera module breakout boards are intended to be used with Leopard Imaging's LeopardBoard ARM development board, the breakouts were found to contain only the supporting circuitry recommended in the MT9V034 datasheet, and we decided that they would be ideal for our application [2, 5].

Once the schematics of each camera module breakout were known, it was then possible to design a basic control interface for each camera. According to the MT9V034 datasheet, each camera module needs to be supplied with an external Master Clock and Output Enable signal in order to operate [5]. A simple Verilog module for the Nexys3 Spartan-6 FPGA board was created in order to supply the camera module with a 24MHz master clock signal, and a switch was used to toggle output enable. With this module implemented, the camera module's default outputs could then be observed. In order to interface the camera module with an FPGA, the breakout board shown in Figure 7 was also created to make the module's pins more easily accessible based on functionality.

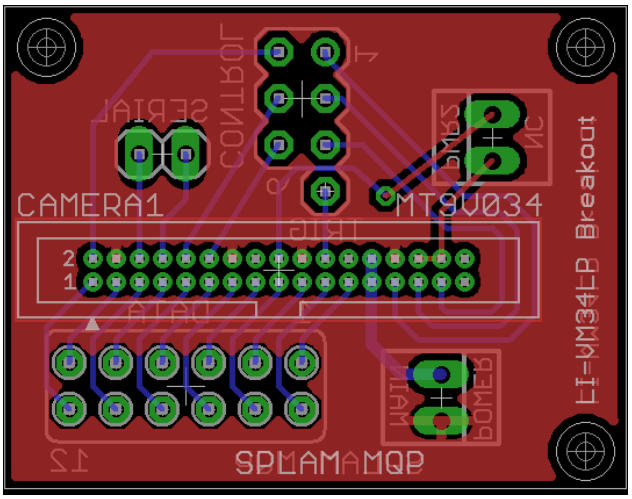

Figure 7: LI-VM34LP Breakout Board

By default, the MT9V034 camera module will continuously gather image data at 60Hz as long as it is supplied with an external clock signal and output is enabled [5]. Several output signals from the camera module are then used to transmit image data. Each image, or frame, is broken up into individual "lines" which correspond to a line of pixels that stretch the width of the frame. Since our camera module captures images at 752x480 pixel resolution, one frame will contain 480 lines of 752 pixels each. The camera module breaks up image data by frame and line, and camera data pins FRAME_VALID and LINE_VALID are toggled to indicate the transmission of a frame or line. The timing diagram shown in Figure 8 shows the operation of these pins while transmitting an image.

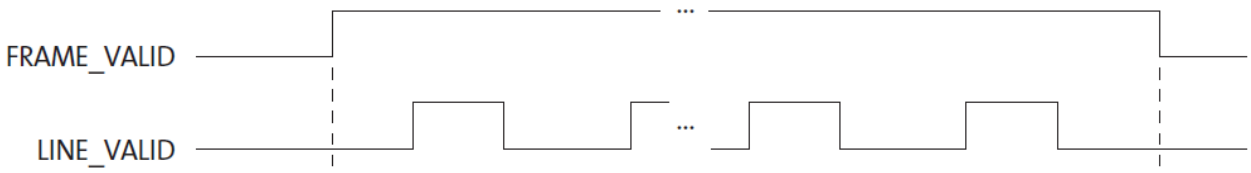

Figure 8: Frame and Line Valid [5]

Since the MT9V034 module transmits image data in parallel and each pixel contains 10 bits of resolution, 10 pins are used to transmit pixel values in parallel. Pixel data is transmitted in correspondence with LINE_VALID and output clock signal PIXCLK. When

LINE_VALID is asserted, the pixel data pins are updated with values corresponding to pixels 0-751 of the given line. Values for each pixel are written out on the falling edge of the camera's PIXCLK pin, allowing for each pixel's value to be read on the rising edge of PIXCLK cycle. A full LINE_VALID data transmission sequence will therefore contain 752 PIXCLK cycles, corresponding to the 752 pixels that make up the given line. A timing diagram of this data transmission scheme is shown in Figure 9.

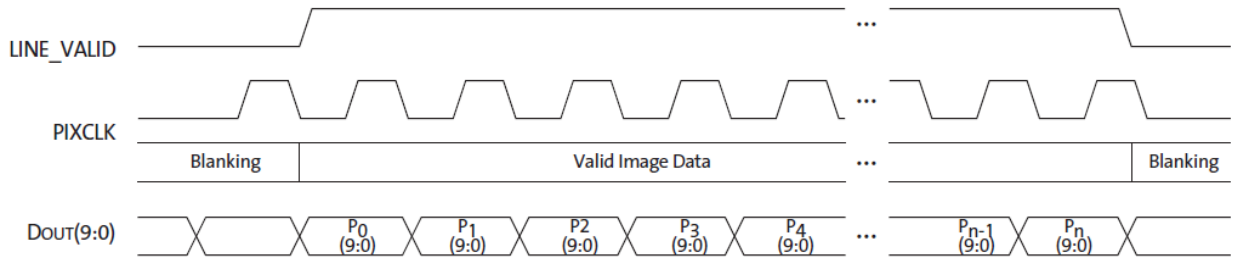

Figure 9: Line Data Transfer [5]

The default camera data transmission scheme was also examined using an oscilloscope, as shown in Figure 10, with channels 1-4 corresponding to camera PCLK, FRAME_VALID, LINE_VALID, and Data[0], respectively. In the case of Figure 10, the camera is initially powered off, resulting in an inactive PCLK signal.

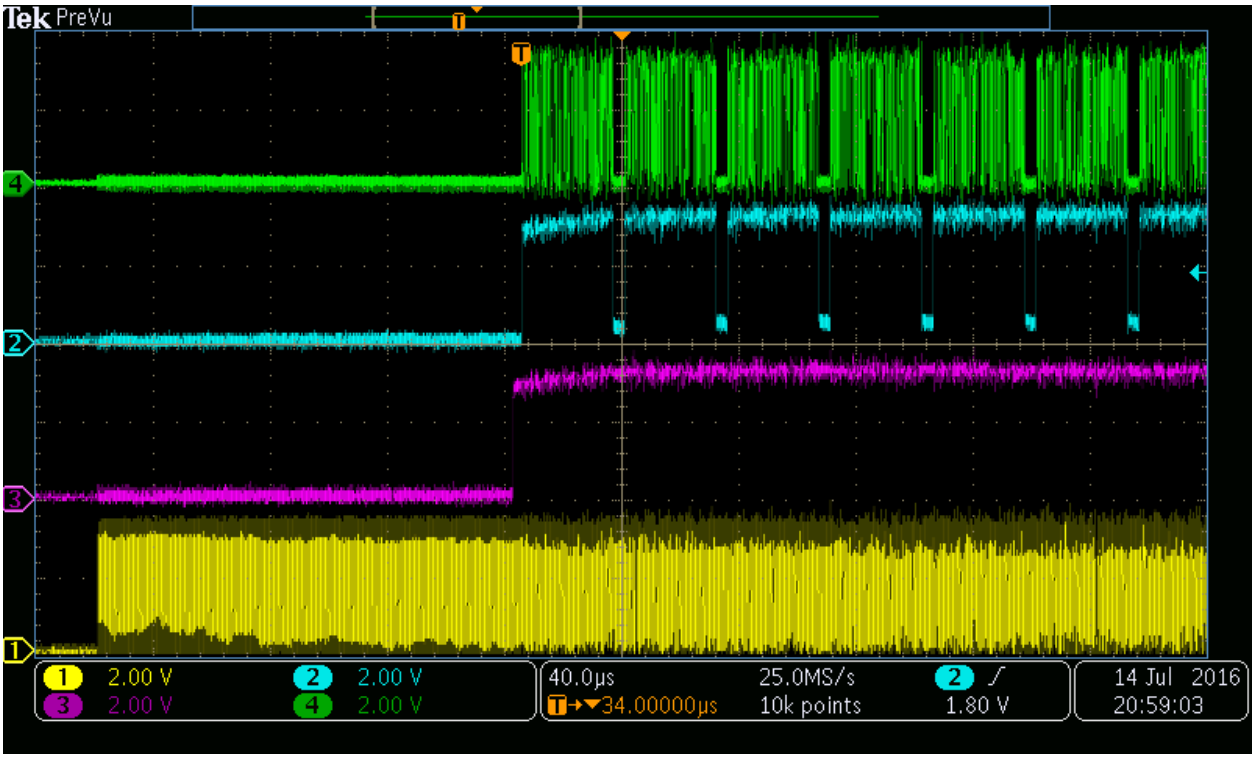

Figure 10: Camera Data Transfer

3.1.2 I²C Control

Started by trying to get the camera version...

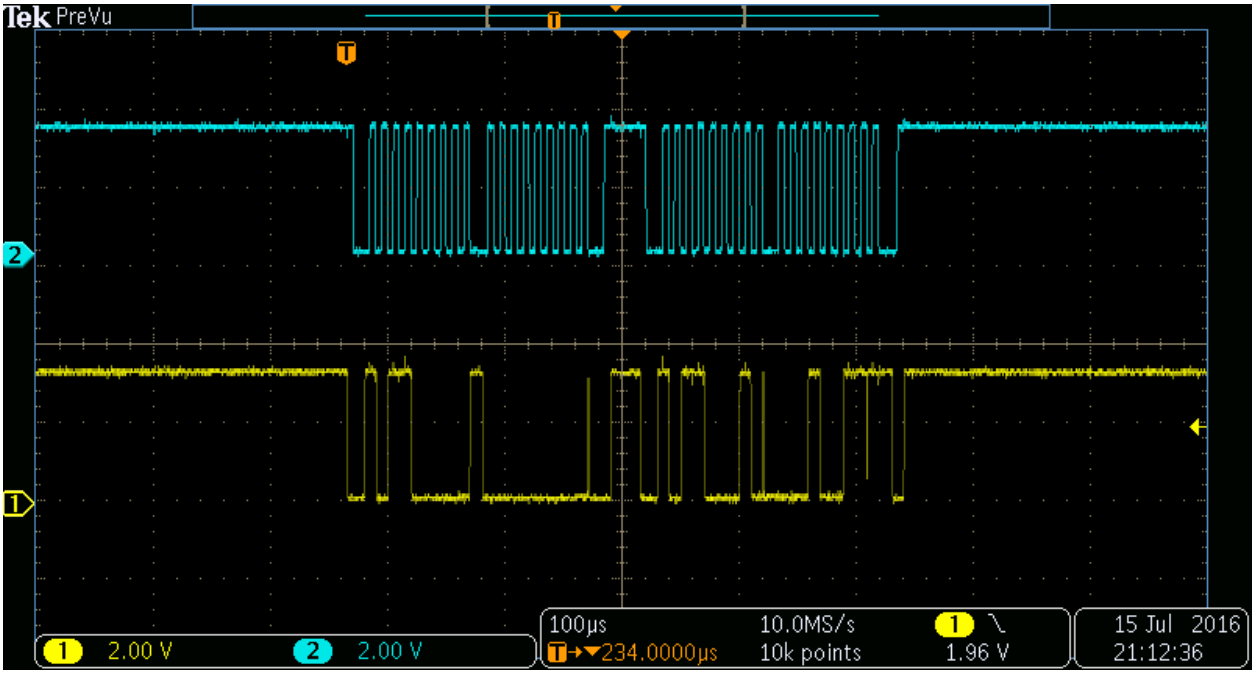

Figure 11: Example I^2C Transfer with Camera

Next, set camera control register for trigger mode and confirmed that it was working. triggered!

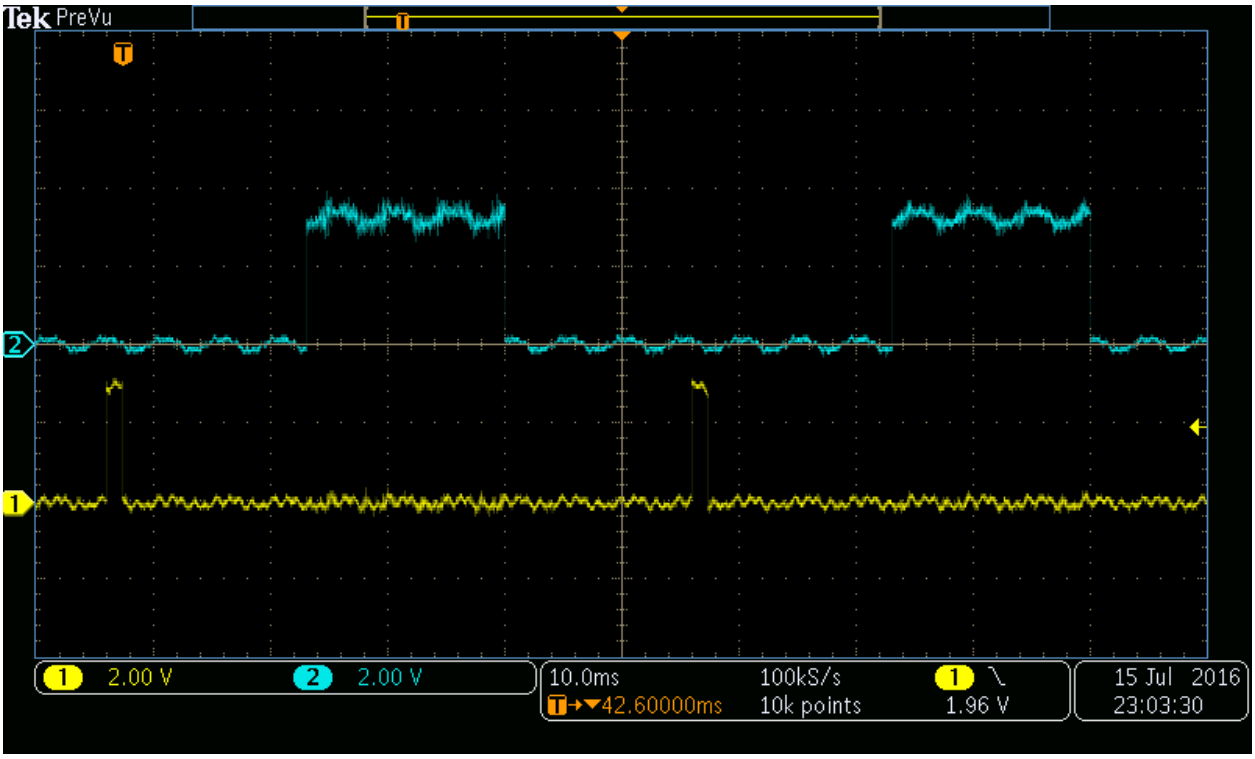

Figure 12: Camera Trigger and FV in Trigger Mode

3.1.3 Data Management

AL422B FIFO, etc...

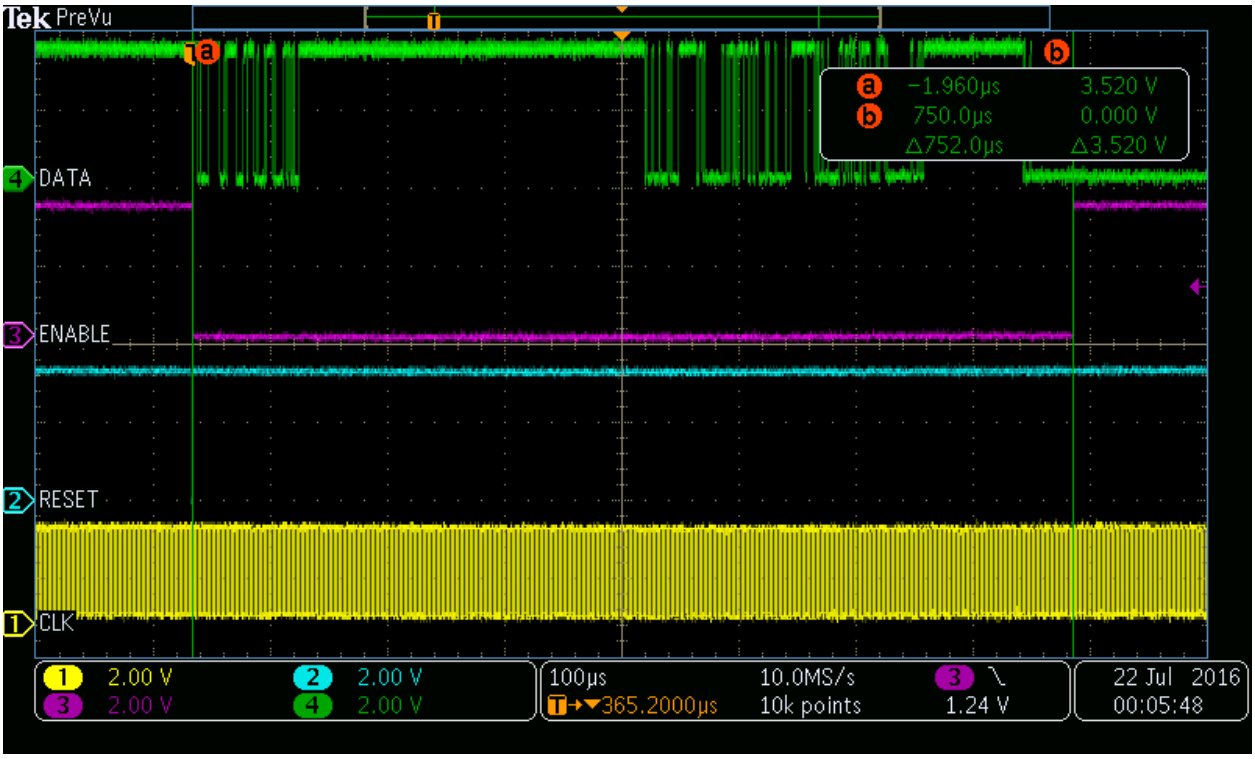

Figure 13: Transferring Line Data from FIFO to FPGA

3.1.4 Transmitting Images Over UART for Analysis

The source code found in Appendix item 4.6.1 was implemented on a Microblaze MCS in order to transmit camera line data from the FPGA's internal line buffer over UART. An example of the microcontroller's UART output is shown in Figure 14.

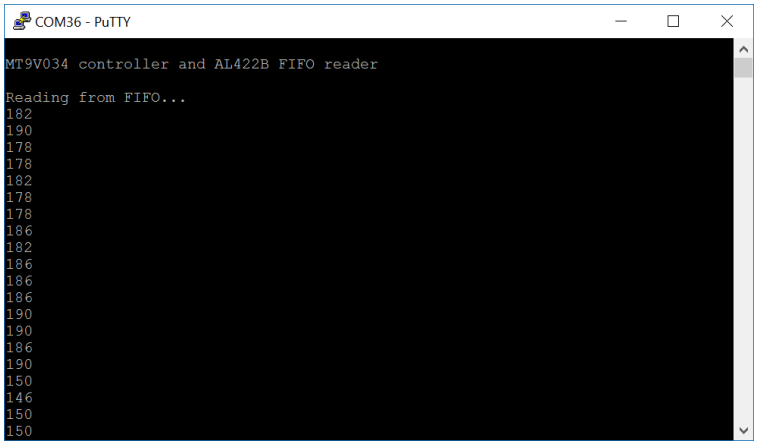

Figure 14: Reading FIFO Data

After the image was recieved through PuTTy, the MATLAB script found in Appendix item 4.4.1 was used to parse the corresponding logfile into a greyscale image. An example image created through this process is shown in Figure 15. Note that the sub-optimal quality of this image is due to the test setup's wiring.

Figure 15: Notebook With Grid and Oscilloscope Leads

References

[1] Davison, A.J., *Real-Time Simultaneous Localisation and Mapping with a Single Camera*. IEEE Computer Vision, 2003. 2(1).

[2] Leopard Imaging Inc. *LI-VM34LP Camera Board*. 2009. Available from: `http://www.leopardimaging.com/uploads/li-vm34lp_v1.1.pdf`.

[3] *Serveball*. Available from: `http://www.serveball.com/`.

[4] Stefano Mattoccia, M.P., *A passive RGBD sensor for accurate and real-time depth sensing self-contained into an FPGA*. in *International Conference on Distributed Smart Cameras*. 2015.

[5] On Semiconductor. *MT9V034: 1/3-Inch Wide-VGA CMOS Digital Image Sensor*. 2015. Available from: `http://www.onsemi.com/pub_link/Collateral/MT9V034-D.PDF`.

[6] Fatih Porikli, O.T., *Human Body Tracking by Adaptive Background Models and Mean-Shift Analysis*. in *IEEE International Workshop on Performance Evaluation of Tracking and Surveillance*. 2003.

[7] Sebastian Thrun, D.H., David Ferguson, Michael Montemerlo, Rudolph Triebel, and C.B. Wolfram Burgard, Zachary Omohundro, Scott Thayer, William Whittaker. *A System for Volumetric Robotic Mapping of Abandoned Mines*. in *IEEE International Conference on Robotics and Automation*. 2003.

4 Appendix

4.1 Useful Resources

This section is intended to serve as complete compilation of all resources gathered throughout D Term 2016 that we believe will be useful as we begin to work on the methodology portion of our project.

- Strother, Daniel. 2011. "Open-Source FPGA Stereo Vision Core Released." https://danstrother.com/2011/06/10/fpga-stereo-vision-core-released/.
- Field, Mike. 2013. "Zedboard OV7670." http://hamsterworks.co.nz/mediawiki/index.php/Zedboard_OV7670.
- "OV7670/OV7671 CMOS VGA CameraChip Implementation Guide." https://www.fer.unizg.hr/_download/repository/OV7670new.pdf.
- "MIPI CSI2-to-CMOS Parallel Sensor Bridge - Lattice Semiconductor." http://www.latticesemi.com/~/media/LatticeSemi/Documents/ReferenceDesigns/JM/MIPICSI2toCMOSPar pdf?document_id=50533.
- "OmniVision Serial Camera Control Bus (SCCB) Functional Specification." http://www.ovt.com/download_document.php?type=document&DID=63".
- Morvan, Yannick. "Multiple-View Depth Estimation." http://www.epixea.com/research/multi-view-coding-thesisse15.html.
- MathWorks. "Depth Estimation From Stereo Video." http://www.mathworks.com/help/vision/examples/depth-estimation-from-stereo-video.html.
- Stefano Mattoccia, Matteo Poggi. 2015. "A passive RGBD sensor for accurate and real-time depth sensing self-contained into an FPGA". International Conference on Distributed Smart Cameras.

- Mattoccia, Stefano. 2013. "Stereo Vision: Algorithms and Applications." http://www.slideshare.net/DngNguyn43/stereo-vision-42147593.

- Szeliski, Richard. 2010. Computer Vision: Algorithms and Applications. New York: Springer.

- Beau Tippets, Dah Jye Lee, Kirt Lillywhite, James Archibald. 2013. "Review of Stereo Vision Algorithms and Their Suitability for Resource-Limited Systems." Journal of Real-Time Image Processing 11 (1). doi: 10.1007/s11554-012-0313-2.

- Jouni Rantakokko, Joakim Rydell, Peter Stromback, Peter Handel, Jonas Callmer, David Tornqvist, Fredrik Gustafsson, Magnus Jobs, Mathias Gruden. 2011. "Accurate and Reliable Soldier and First Responder Indoor Positioning: Multisensor Systems and Cooperative Localization." IEEE Wireless Communications 18 (2):10-18. doi: 10.1109/MWC.2011.5751291.

- Davison, Andrew J. 2003. "Real-Time Simultaneous Localisation and Mapping with a Single Camera." IEEE Computer Vision 2 (1). doi: 10.1109/ICCV.2003.1238654.

- Fatih Porikli, Oncel Tuzel. 2003. "Human Body Tracking by Adaptive Background Models and Mean-Shift Analysis." IEEE International Workshop on Performance Evaluation of Tracking and Surveillance.

- Giovanni Pintore, Enrico Gobbetti. "Effective mobile mapping of multi-room indoor structures." The Visual Computer 30 (6):707-716. doi: 10.1007/s00371-014-0947-0.

- "iRobot 110 FirstLook." iRobot. http://www.irobot.com/$~$/media/Files/Robots/Defense/FirstLook/iRobot-110-FirstLook-Specs.pdf.

4.2 Component Selection

Component	Part Number	Supplier	Cost
FPGA	ZedBoard	Borrowed	N/A
IMU	ADIS16375	Borrowed	N/A
Rangefinder	URG-04LX	Borrowed	N/A
Stereo Cameras†	MT9V034	Mouser	$146

† Note that we originally planned to purchase a flir lepton thermal camera module and accompanying breakout board to support two stereo ov7670 camera modules. After experimenting with the ov7670 camera module on our FPGA board, we began to realize that these camera modules are highly limited due to their low frame rate and poor documentation, and realized that we wanted to search for a different camera module. In addition, at a price of $223 for a thermal camera with an 80x60 degree resolution, 25 degree fov, and 7-9Hz image sample rate, we believe that we are much better off spending our money on better camera modules that will be more usable for our task. For more information see Appendix Item 3.

4.3 Camera Module Decision Matrix

Camera Module	Max Frame Rate (FPS)	Resolution at Max Frame Rate (px.)	Cost	Requires External Adapter	Data Transfer Interface	Shutter	Field of View (deg.)	Rank 0-10
OV7670	30	640x480	$10	No	Parallel	Rolling	25	5
Raspberry Pi Camera	90	640x480	$30	Yes, $53	MIPI (CSI2)	Rolling	49	6
PC1089K	60	720x480	$32	No	NSTC/ PAL	Rolling	Not Given	5
OV4682	330	640x480	$89	Yes, $50	MIPI	Rolling	Not Given	6
MT9V034	**60**	**750x480**	**$73**	**No**	**Parallel**	**Global**	**55**	**9**

For purposes of comparison, the thermal camera module we were evaluating is also shown below.

Flir Lepton	9	80x60	$175	Yes, $40	SPI/ MIPI	N/A	50	3

Shown above is our decision matrix for choosing a camera module. Fields marked in green indicate a positive ranking, while red indicates a negative ranking. Based on the individual

rankings of each item's field, we gave our camera modules an overall ranking of 0-10 in the right hand column, with 10 being an extremely high ranking and 0 being an extremely low ranking.

Based on our decision matrix, we believe that it would be worth both our time and money to use the MT9V034 camera modules for our stereo camera interface. These camera modules are the only low-cost global shutter option we've come across, and would be ideal for taking images in a sensor suite that is susceptible to motion. The MT9V034 also uses a parallel data interface and relies on an external clock and shutter trigger, making the module ideal for interfacing with an FPGA-based stereo imaging setup.

4.4 Matlab Code

4.4.1 Camera Image Parsing

```
% Camera data read through use of AL422B FIFO, arduino, and PuTTy logging
clear all;
close all;
FILENAME =  uigetfile('*.log','multiselect','off');
fprintf('File %s selected\n\r',FILENAME);
fid = fopen(FILENAME,'r');
image = zeros(480,752);
XPOS = 1;
YPOS = 1;
LINENUM = 1;
ERRNUM = 0;
h = waitbar(0,'Parsing image...');
c = fgetl(fid); %get rid of 1st line
while 1
  c = fgetl(fid);
  if ~ischar(c), break, end
  if length(c) > 0
      image(YPOS,XPOS) = str2num(c)/255;
  else
      ERRNUM = ERRNUM + 1;
      fprintf('Error #%d: Line %d contains no data\n\r',ERRNUM,LINENUM)
  end
  if XPOS<752
      XPOS = XPOS + 1;
  else
      XPOS = 1;
      YPOS = YPOS + 1;
  end
```

```
  if mod(LINENUM,36096)==0
    waitbar(LINENUM /360000);
  end
  LINENUM = LINENUM + 1; % current line in file (for debug)
end

figure
imshow(image);

%save the image to a file, overwrite if already saved
[path,name,ext] = fileparts(FILENAME);
imgname = strcat(name,'.png');
if (exist(imgname, 'file') == 2)
    fprintf('File for image already exists... overwriting it\n\r')
    delete(imgname);
end
% change to eps for vector image
saveas(gcf,imgname);
fprintf('Saved figure to image %s\n\r',imgname);

fclose(fid);
close(h)
```

4.5 Verilog Code

4.6 C Code

4.6.1 Camera Image Parsing

```
/*
 * Source code for printing values from the AL422B FIFO / FPGA Buffer over UART
 */

#include <stdio.h>
#include "platform.h"
#include "xparameters.h"
#include "xiomodule.h"

// GPO1
#define GETDATA (1<<2) // load a new line of pixels into the FPGA buffer
#define RRST (1<<1) // reset to address 0
#define LED (1<<0) // LED indicator
// GPI2
#define SW_READ (1<<1)
#define BUF_READY (1<<0)

void print(char *str);
void _EXFUN(xil_printf, (const char*, ...));
```

```
20
21
22  int main()
23  {
24      init_platform();
25      int pixel_position = 0,row = 0;;
26      u8 data=0x00,GPO1=0x00,GPI2=0x00,swState=0x00,prevState=0x00;
27
28      XIOModule gpi;
29      XIOModule gpo;
30
31      // GPI1 = pixel_value(7:0)
32      XIOModule_Initialize(&gpi, XPAR_IOMODULE_0_DEVICE_ID);
33      XIOModule_Start(&gpi);
34
35      // GPO1 = (GETDATA)|(RRST)|(LED)
36      XIOModule_Initialize(&gpo, XPAR_IOMODULE_0_DEVICE_ID);
37      XIOModule_Start(&gpo);
38
39      print("\n\rMT9V034 controller and AL422B FIFO reader\n\r");
40      while(1)
41      {
42          // get switch position
43          GPI2 = XIOModule_DiscreteRead(&gpi,2);
44
45          if((GPI2&SW_READ)!=0){
46              swState = 1;
47              if (row >= 480) GPO1 &=~(LED);
48              else GPO1 |= LED;
49              GPO1 &=~(RRST|GETDATA);
50              XIOModule_DiscreteWrite(&gpo,1,GPO1);
51          }else{
52              GPO1 &= ~(RRST|LED|GETDATA);
53              XIOModule_DiscreteWrite(&gpo,1,GPO1);
54              row = 0;
55              swState = 0;
56          }
57
58          // code below runs only once based on SW state change
59          if (prevState != swState){
60              if(swState){
61                  print("\n\rReading from FIFO...\n\r");
62
63                  GPO1 |= (RRST); // reset FIFO position to 0th index
64                  GPO1 &=~ (GETDATA); // make sure we're not trying to read data
65                  XIOModule_DiscreteWrite(&gpo,1,GPO1);
66                  pixel_position = 0;
67                  GPO1 &=~(RRST);
68                  XIOModule_DiscreteWrite(&gpo,1,GPO1);
69                  GPO1 |= (GETDATA); // make sure we're not trying to read data
70                  XIOModule_DiscreteWrite(&gpo,1,GPO1);
71
72                  u32 pixelsRead = 0;
73
```

```
74                  while(row<480){
75                      // make sure read_sw hasn't been turned off
76                      GPI2 = XIOModule_DiscreteRead(&gpi,2);
77                      if ((GPI2&SW_READ)==0) break;
78
79                      u8 i=0;
80                      // check to see if BUF_READY is good to go
81                      GPI2 = XIOModule_DiscreteRead(&gpi,2);
82                      // wait until it is
83                      while((GPI2&BUF_READY)==0){
84                          if(i==0){
85                              i++;
86                              //print("\n\t buffer not ready \n\r");
87                          }
88                          GPI2 = XIOModule_DiscreteRead(&gpi,2);
89                      }
90                      GPO1 &=~ (GETDATA); // make sure we're not trying to read data
91                      XIOModule_DiscreteWrite(&gpo,1,GPO1);
92
93                      while (pixel_position < 752){
94                          // update pixel position for FPGA buffer
95                          XIOModule_DiscreteWrite(&gpo,2,pixel_position);
96
97                          // make sure read_sw hasn't been turned off
98                          GPI2 = XIOModule_DiscreteRead(&gpi,2);
99                          if ((GPI2&SW_READ)==0) break;
100
101                         // read value at pixel position from FPGA buffer
102                         data = XIOModule_DiscreteRead(&gpi,1);
103
104                         //print the value
105                         xil_printf("%d\n\r",data);
106
107                         // increment to the next pixel position
108                         pixel_position++;
109                         pixelsRead ++;
110                     }
111                     // signal to the FPGA that we want more data!
112                     GPO1 |= (GETDATA);
113                     XIOModule_DiscreteWrite(&gpo,1,GPO1);
114                     pixel_position = 0;
115                     row++;
116                     //xil_printf("Row: %d",row);
117
118                 }
119                 //xil_printf("%d Pixels Read by MCS",pixelsRead);
120             } else {
121                 GPO1 &=~(GETDATA);
122                 GPO1 |= RRST;
123                 XIOModule_DiscreteWrite(&gpo,1,GPO1);
124                 print("\n\rReset for new sequence\n\r");
125             }
126         }
127
```

```
128             prevState = swState; // update prev switch position
129         }
130         cleanup_platform();
131         return 0;
132 }
```

4.7 LaTeX Coding Examples

This section isn't intended to remain here, but can serve as an example for how to set things up later on

4.7.1 Figures

Figure 16: A Test Figure

Using the `\ref` command, I'm able to reference Figure 16 by calling `\ref{wpiLogo}`.

4.7.2 Code Snippet

Code snippets can be created by calling `\begin{lstlisting}[style=<Language_Name>]`, inserting all code, and then calling `\end{lstlisting}`. Also call `\singlespacing` before the code snippet and `\doublespacing` after to keep things from getting too big.

Note that you can also use

`\lstinputlisting{src.ext}[style=<Language_Name>,firstline=lineNumber, lastline=lineNumber]`

with reference to source files and listings will import the code for you. Way less work!

```
//verilog code example
always @ (x, y, z)
  x <= y + z;
```

```
%Matlab code example
clear all;
close all;
FILENAME =  uigetfile('*.log','multiselect','off');
fprintf('File %s selected\n\r',FILENAME);
return;
```

```
//C code example
#include "someheader.h"
```

```
3  XIOModule gpo;
4
5  void GPOWrite(u8 value){
6      XIOModule_DiscreteWrite(&gpo,1,value);
7      xil_printf("Wrote %d to GPO\n\r",value);
8      return;
9  }
```

4.7.3 Using the bibliography

All bibliographic references are contained in `bib.tex`. To cite a reference in the paper, use the `\cite` command.

As an example, I can cite *Serveball* at the end of this sentence by calling `\cite{serveball}`.[3]

To cite multiple references, call `\cite{ref1,ref2}`.[3, 6]