Real Time Pedestrian Detection Using an Infrared Camera with a FPGA

A Major Qualifying Project Report

Submitted to the faculty of the
WORCESTER POLYTECHNIC INSTITUTE
in partial fulfillment of the requirements of the
Degree of Bachelor of Science

By:

Adam Karcs, Electrical and Computer Engineering
Arsene Tchatchoua, Electrical and Computer Engineering
Damani Walder, Electrical and Computer Engineering

Submitted to:
Project Advisor: Professor Xinming Huang,
Department of Electrical and Computer Engineering

Submitted on: March 7, 2016

Abstract

This project focuses on using Infrared technology in partnership with various filtering algorithms to implement a pedestrian detection system on a Field Programmable Gate Array (FPGA). Currently pedestrians are the most vulnerable users of the road. Every day there are millions of vehicles on the road and conditions such as inclement weather, poor lighting, traffic and other road hazards restrict the visibility of drivers which increases the risk to pedestrians. In addition, human error is known to be one of the leading causes of accidents. With the use of a pedestrian detection system which utilizes infrared technology, we are able to remove some of the hazards and provide an additional sense of awareness to the driver to reduce the occurrence of accidents and save lives.

Acknowledgements

On behalf of our Major Qualifying Project team, we would like to acknowledge everyone that helped to provide this opportunity for us to work on this project and the great success that resulted from it.

Our deepest gratitude to our advisor, Dr. Xinming Huang for his guidance, advice and motivation throughout this project. Without his encouraging words and motivating talks during our weekly meetings, this project would not have been successful.

Furthermore, we would like to thank the graduate students in the Embedded Computing Lab at Worcester Polytechnic Institute for their support, particularly Roy Sun without whom this project would not have been possible.

We would like to sincerely thank Worcester Polytechnic Institute (WPI) for providing the facilities and equipment for this research

Without the assistance of these individuals, this project would not have been the success that it was. Thank you!

Authorship

This report was developed through a collaborative effort by all three members of the project team: Adam Karcs, Arsene Tchatchoua and Damani Walder. All sections were created and edited as a team, with equal contributions made by each member.

Table of Contents

Table of Figures

Executive Summary

There are about 2 billion vehicles in the world today. Millions on them are on the road at any given time. In many areas, conditions such as weather, time of day, traffic, and others makes it difficult to keep up visibility and therefore awareness of pedestrians amongst drivers.

Human error is one of the leading causes of accidents. There are always various details in any given environment that a driver must always be aware of. For example, distance between vehicles, lights changing, signs, and pedestrians. Over time, this usually develops into an unconscious skill, but that very fact makes it easier to be surprised by unusual and sudden events.

By using technology to enhance pedestrian detection within vehicles, we are able to remove some of the errors in judgment and thus possibly reduce the number of accidents. With thermal vision, we are able to reduce interfering conditions such as snow, rain, darkness and hazards which may have hindered a driver’s visibility. When thermal vision is used in conjunction with pedestrian detection algorithms, it can provide an additional sense of awareness to the driver and help reduce the number of accidents.

To accomplish this, our project team used an FPGA in accordance with the FLIR Lepton Infrared Camera module to implement a real-time pedestrian detection system.

Chapter 1

1.1 Pedestrian Detection

Among all road users, pedestrians are the most vulnerable [1]. This vulnerability can be attributed to many different factors when compared to other road users. One such factor is that pedestrians are the most difficult road users to observe during both ideal and non-ideal conditions. Another factor which can be observed is that, unlike motorists or cyclists, the common pedestrian has no means to lesson or absorb any impact if they are struck in a collision. As a result, pedestrians involved in collisions are often in danger of suffering severe injuries or being the victim of a road fatality. The danger of pedestrian collisions not only affect the pedestrians, as sometimes the occupants of the vehicle also suffer injuries, either due to the impact or during a maneuver whilst trying to avoid the collision.

According to a recent study by the Governors Highway Safety Association, there was 4,735 reported pedestrian fatalities in the United States in 2013, a 15 percent increase to the number of incidents reported in 2009, when there were 4,109 incidents [2]. This increase in the amount of pedestrian fatalities line up with the growing number of vehicles on the road today.

Studies show that most traffic accidents are related to human errors. Carelessness and erroneous decisions by the driver are the two main factors that cause traffic accidents. Although these types of errors are nearly impossible to eliminate, efforts can be made to decrease them [3].

Many states employ a variety of methods to try to curb the incidents of pedestrian fatalities each year through educational programs, different engineering practices (such as pedestrian overpasses), high-intensity crosswalk (HAWK signals), and enforcement strategies [2].

Recent research in the field of Intelligent Vehicles (IV) has focused on using advances in information technologies to prevent human error. Advanced Driver Assistance Systems (ADAS) try to warn and prepare the driver in the event of possibly hazardous situations [3]. Among ADAS, pedestrian detection is an important issue due to the vulnerability of pedestrians in the event of accidents [3].

In recent years, many auto manufacturers along with other tech companies have done extensive research on pedestrian detection systems, integrating output signals into the control system of cars, as a way to further curb pedestrian fatalities. Most pedestrian detection systems use sensors, which are integrated into the vehicle or attached with an after-market kit, in order to ‘sense’ pedestrians in the nearby vicinity of the vehicle and alert the driver with warnings. Some modern built-in sensor systems are able to output warning signals directly to the car’s control system and apply automatic braking to help avoid or mitigate a crash [4]. Modern pedestrian detection systems use a variety of techniques to ‘sense’ the presence and proximity of pedestrians, such as infrared sensors, visible light cameras, laser and radar technologies. In order to process the data from these sensors, FPGAs, microcontrollers and microprocessors are often used.

In a concerted effort to encourage progress in the field of pedestrian detection, multiple datasets have been created which contain real life video/images of pedestrians in different scenarios. These datasets are updated regularly. One of the largest and commonly used datasets in the computer vision research community is the Caltech Benchmark Dataset. The Caltech Pedestrian Dataset consists of approximately 10 hours of 640x480 30Hz video taken from a vehicle driving through regular traffic in an urban environment [5].

1.2 Future of Pedestrian Detection

Autonomous cars or self-driving cars is a concept of the future which seems closer to being a reality as more strides are made in automobile technology. With the combination of current ADAS such as lane departure warning, blind spot detection, adaptive headlights, park assist, GPS among others, commercial autonomous cars could become a reality within the next 5 years.

Companies such as Google and Tesla are currently leading the charge in the field of autonomous cars. Google is currently testing their autonomous cars on the streets in California, Texas and Washington while Tesla recently added a ‘Summon’ feature to their line of vehicles which will allow a driver to remotely summon their car, and in response, their car will autonomously drive itself to the location of the owner. Though both ventures are in their early stages, it hints at how close commercial fully autonomous cars are to being a reality.

Chapter 2 – Hardware Design

2.1 FPGAs

2.1.1 Introduction to FPGAs

FPGAs, or Field Programmable Gate Arrays, are devices comprised of many logic blocks, most of which can be configured for any number of applications. Configurable logic blocks (CLBs) are the basic building elements of an FPGA, and are usually referred to as slices. These slices, in turn, are mainly made up of flip-flops and lookup tables (LUTs) [6].

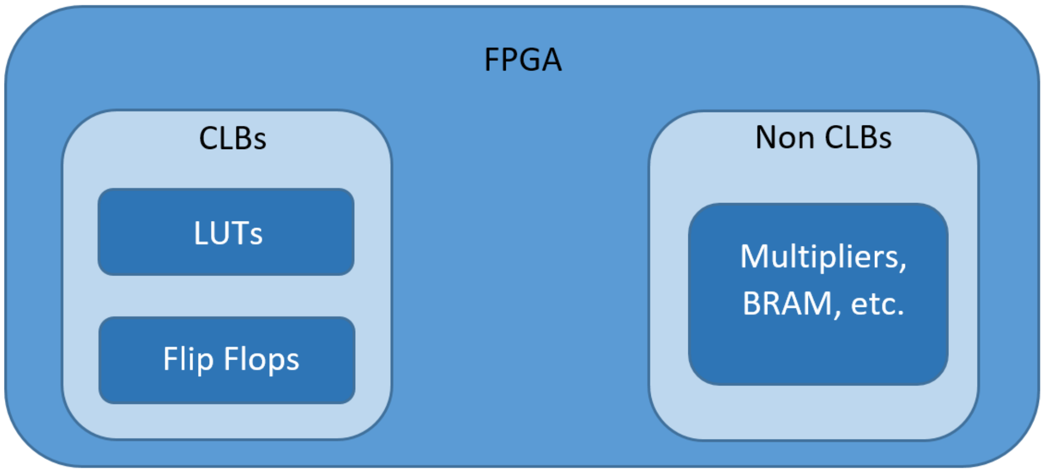


Figure 1 - Simple Representation of FPGA

Flip-flops are the basic memory blocks which can hold a logic state from one clock cycle to the next. LUTs are the physical representation of a truth table in digital logic, essentially a set of relations stored in small amounts of memory. An FPGA also contains other function blocks, e.g. multipliers, block RAM, etc., which are used to optimize the overall application [6].

These FPGA boards differ from Application-Specific Integrated Circuits (ASICs) because they can be reprogrammed after manufacturing, whereas ASICs are given their configuration, and then usually mass-produced that way. In addition, they differ from microcontrollers because they give you control of the software implemented on the board whereas an FPGA gives you control over the hardware itself [7].

FPGAs are used by writing in a Hardware Description Language (HDL), two most common of which are VHDL and Verilog. For this project, we decide to go with Verilog due to the familiarity throughout the group as well as simplicity in the coding scheme.

2.1.2 Why we chose FPGAs

For specialized purposes and applications which would be intended to be mass-produced, FPGAs have many advantages over PCs in terms of image processing. Because PC processors are general purpose, a gate array configured for a certain function will operate at a much faster rate without background processes weighing it down. In the field of digital image processing (DIP), FPGAs are becoming increasingly more common due to these facts [8].

When using FPGA applications, there is relatively little boot-up time compared to a PC because the program is saved on the flash memory on the board. When the board is powered, the FPGA automatically starts the program to process the input and generate output. In addition to the built-in security features of the Xilinx FPGAS (such as JTAG Monitoring and AES 256 encryption), this lack of an operating system can actually be beneficial in various ways [9].

Having custom code instead of a brand name operating system attributes to the security of the FPGA program. Development boards like the Raspberry Pi work a bit more like personal computers, where the file system is usually directly accessible with a simple login. Alternatively, the Nexys does not work the same way. Although it does have non-volatile code programmed onto it, it would not be reconfigured easily. Any malicious efforts would require disconnecting the board, hooking it up to a PC with Xilinx software installed (which requires a license to Synthesize) and the project itself which would not be obtainable directly from the board.

2.1.2 Nexys 3 Spartan 6 Board

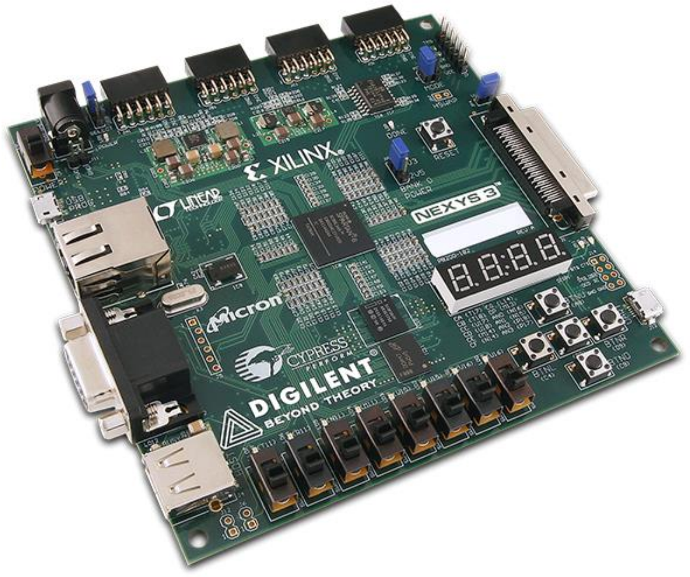


Figure 2 - Nexys 3 - Spartan 6 Board [10]

As the specific FPGA board used for this project, the Nexys 3 Spartan-6 FPGA Board is optimized for lowest-cost logic, which provides many unique I/O ports on a single board, and different types of RAM for the developer to use. Delving further into the resources on the board, the most important components for this project are detailed below:

- Spartan-6 XC6SLX16 capable of 500 MHz clock speed
- 48 MB memory
    - 16 MB Cellular RAM
    - 16 MB SPI PCM (phase-change mem) non-volatile memory
    - 16 MB parallel PCM non-volatile memory

- USB2 port (for programming/power)
- 8-bit VGA port
- 8 slide switches
- 4 Banks of 12 PMOD connectors

The stats shown above are from the Nexys 3 ™ FPGA Board Reference Manual by Digilent [10].

Due to the highly configurable nature of FPGAs, there are such a vast range of applications that these boards can be targeted for, including but not limited to: Security, Communications, Medical Electronics, and Smarter Vision. For this last category, the boards can be used in areas such as Real-Time Analytics, Intelligent Transport, and Product Differentiation [7].

2.2 Infrared Technology

2.2.1 Infrared (IR) Background

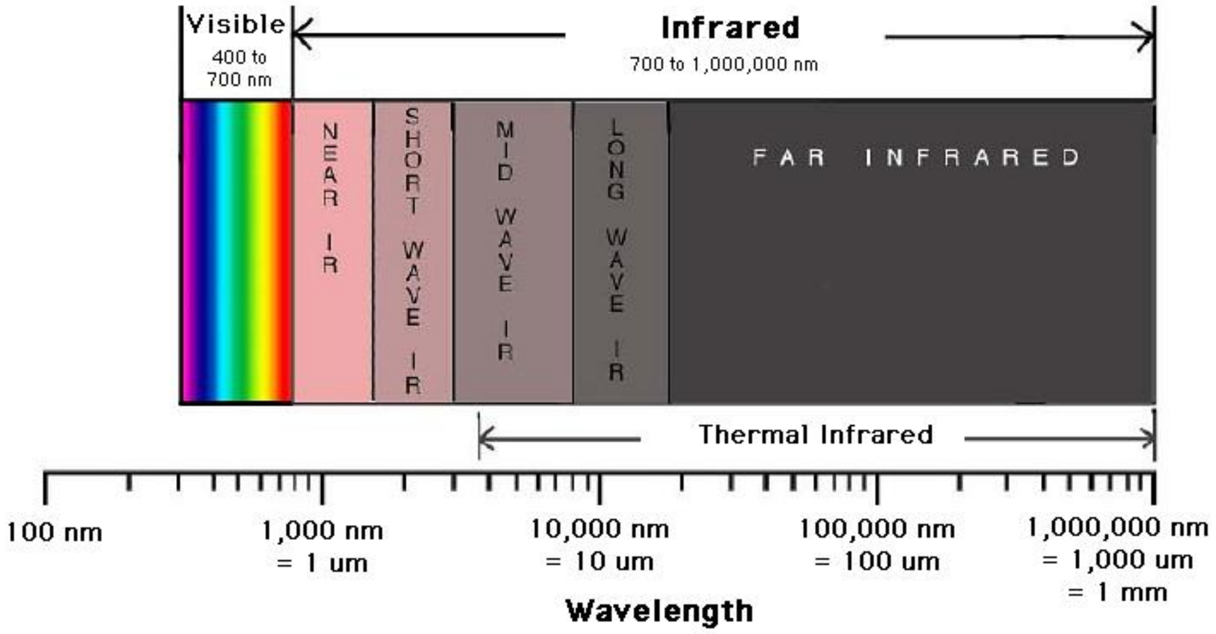


Figure 3- IR Spectrum

Infrared (IR) technology involves/refers to devices which utilize infrared waves in many different ways. So let's take a step back and look at how that works. In a data collection sense of IR technology, referred to in the field as IR wireless, a device emits IR light (in this case, Long Wave IR), and compares the reflection as it bounces off objects in its view. Infrared light borders visible light on the EM spectrum, with frequencies lower and wavelengths longer than that of visible light.

2.2.2 Infrared Camera over a Visible Light Camera

With the help of infrared cameras, we can utilize the ability to see an environment in any light conditions, as well as extract heat signature information in these conditions. Most current pedestrian detection modules and techniques use only a visual light camera, but by using an infrared camera, we can see pedestrians during the day and during the night. In addition, filtering for organic beings is easier to be implemented with a thermal camera, because it is a contrast between the object and the background.

2.2.3 FLIR Lepton Module Camera

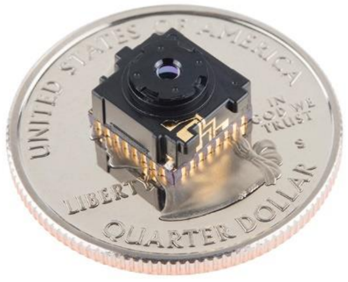


Figure 4 FLIR Camera Module Size Comparison [11]

In this project, we use the FLIR LEPTON© Long Wave Infrared Camera, which is a "passive thermal imaging module for mobile equipment". This camera delivers video data serially of the thermal representation of the environment in an 80x60 resolution. To operate like this, the board must have MOSI and MISO pins, to send data to and from the breakout board [11].

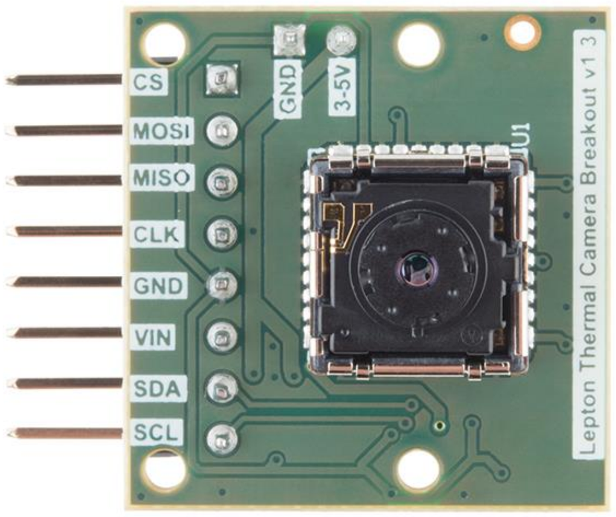
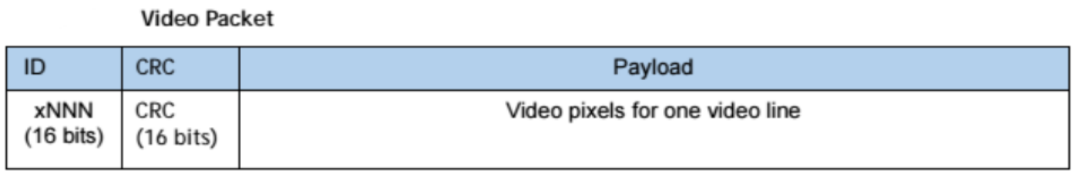


Figure 5 Lepton Camera with Breakout Board [11]

The board shown above requires a VIN of 3.3 V and a clock frequency (CLK) of 20 MHz. Controlling both the Chip Select (CS) and Master In Slave Out (MISO) pins is essential for collecting data from the camera module. More on this is detailed in Chapter 3.



Video Packet

ID	CRC	Payload
xNNN (16 bits)	CRC (16 bits)	Video pixels for one video line

Figure 6 - Representation of how data is passed in from the camera [12]

The data coming in from the camera arrives in packets of 164 bytes. The first 4 bytes are split evenly between the ID and CRC fields (see Figure 6*)*. The remaining 160 bytes are the payload. For a regular packet, the ID field looks like xNNN, meaning the first bit is ignored and the remaining 3 determine the ID. If there is ever to be a packet starting with xFxx, this is a discard packet and does not get used in the processing or display. For our purposes, we are not implementing the CRC checksum calculations due to the timeline of the project.

Each packet refers to a line/row of data in the video feed. Therefore, one frame of video contains 60 packets of 80 pixels. This camera is capable of delivering frames of data at a flexible clock rate between 2.2 MHz and 20 MHz. To keep a valid frame timing and maintain synchronization with the display, one frame must be read within 1/27 of a second [12].

Currently, small thermal cameras such as the Lepton FLIR module are being used in many different fields. Our purpose is to detect pedestrians in any given light conditions on an embedded platform, which can be installed on a car or even an autonomous vehicle. Often, they are being used by both law enforcement and consumers to detect unwanted persons outside a facility at night. They are also being used to detect leaks in the piping of homes, operate automatic doors, and for many other applications [9].

Chapter 3 - Implementation

3.1 Overview of Dataflow

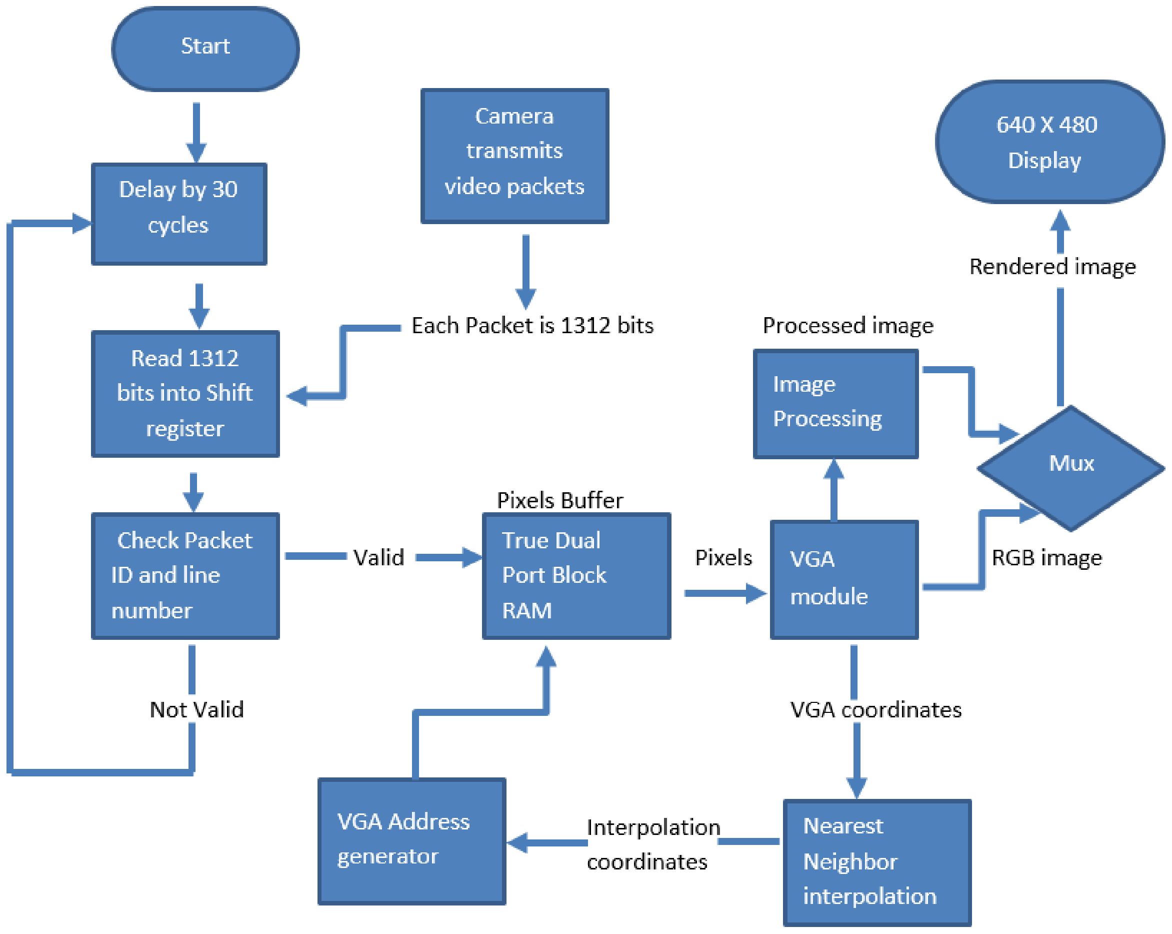


Figure 7 - Flowchart of the dataflow of our system

The lepton camera module is connected to the PMOD pins of the Nexys 3 board and synchronized for video transmission through an SPI interface. This interface is implemented using a Finite State Machine (FSM) design method. The SPI Verilog code can be seen in the appendix. The Nexys 3 FPGA produces the serial clock and chip select signals needed to synchronize with the camera. Since the Nexys 3 board only reads video packets transmitted by the camera, only the MISO breakout pin is asserted. Hence the only SPI pins used in this SPI interface are the Serial clock (SLCK), chip select (CS) and Master In Slave Out (MISO).

3.2 System Integration

3.2.1 Input Camera and Clocking

The camera uses the SPI mode 3 protocol; Clock Polarity (CPOL) is 1 and Clock Phase is (CPHA) is 1. CPOL with 1 means the idle level of the Serial clock (SLCK) should be high (1) and CPHA with 1 means data must be sampled from the camera at the positive edge of the clock. Figure 8 shows diagrams of the appropriate SLCK waveforms for each SPI mode. It is very important to identify the SPI mode of each device before building an SPI interface because not all devices work with the intuitive SPI mode 0.

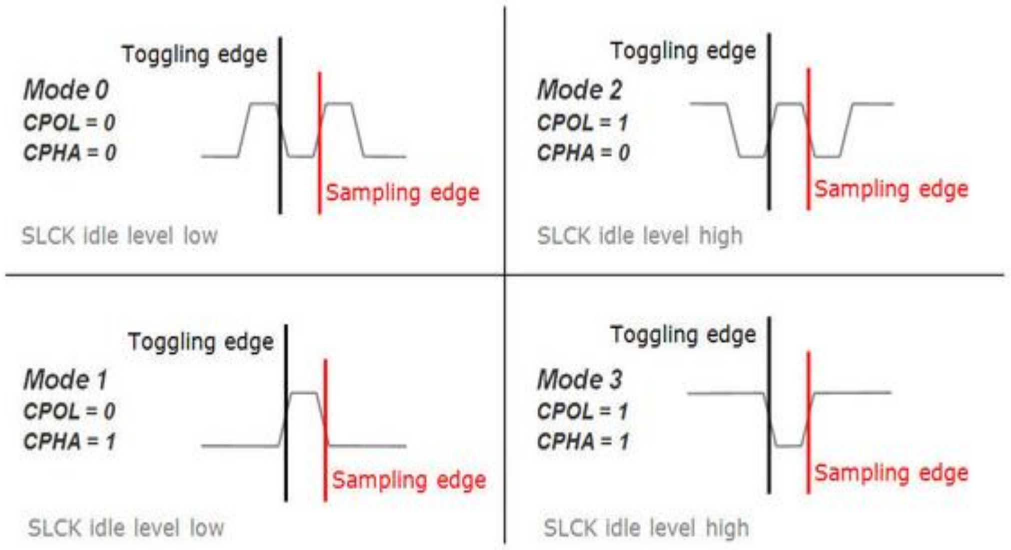
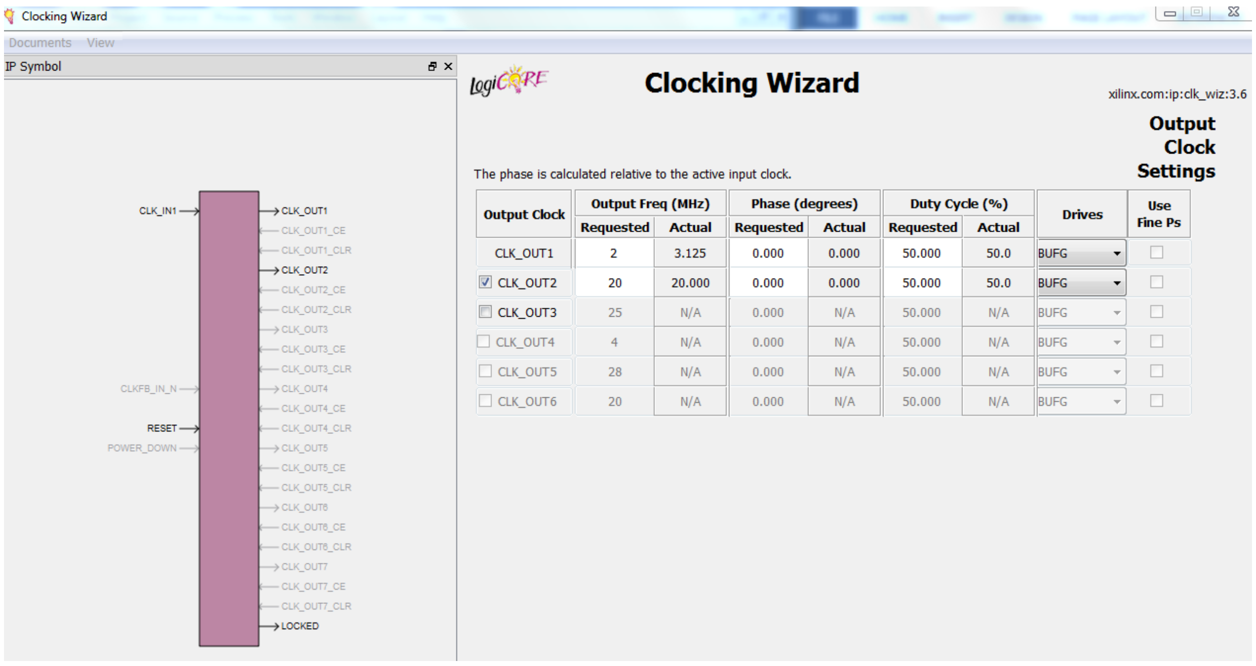


Figure 8 - SPI Modes - Waveform Comparison

The camera is clocked at a rate of 20 MHz to enable correct synchronization with the board, hence our state machine serial clock rate is also 20MHz. This clock output was generated using the ISE Clocking Wizard Intellectual Property (IP) core. This is shown below:



Clocking Wizard

xilinx.com:ip:clk_wiz:3.6

Output Clock Settings

The phase is calculated relative to the active input clock.

Output Clock	Output Freq (MHz)		Phase (degrees)		Duty Cycle (%)		Drives	Use Fine Ps
	Requested	Actual	Requested	Actual	Requested	Actual		
CLK_OUT1	2	3.125	0.000	0.000	50.000	50.0	BUFG	☐
☑ CLK_OUT2	20	20.000	0.000	0.000	50.000	50.0	BUFG	☐
☐ CLK_OUT3	25	N/A	0.000	N/A	50.000	N/A	BUFG	☐
☐ CLK_OUT4	4	N/A	0.000	N/A	50.000	N/A	BUFG	☐
☐ CLK_OUT5	28	N/A	0.000	N/A	50.000	N/A	BUFG	☐
☐ CLK_OUT6	20	N/A	0.000	N/A	50.000	N/A	BUFG	☐

Figure 9 – ISE Clocking Wizard IP core

3.2.2 Read Packets State

The camera transmits video information as packets of data in Big Endian (Most Significant Bytes first). There are two kinds of packets sent by the camera: video packets and discard packets. Each packet is 1312 bits in length so a 1312-bit shift register is declared to capture each packet. This also means it took 1312 clock cycles to read each packet. The SLCK and CS signals are initially set to 1 which kept the camera in Idle mode. A delay state of 30 clock cycles is added to enable the camera to boot up properly. After the delay, the SLCK signal is toggled to 0 and the camera loads data on the MISO pin. On the next clock edge which is the sampling edge, the SLCK signal is reset to 1 (which produced a positive edge) and the CS signal is toggled to 0. Hence the data at the MISO pin is read into the shift register. The SLCK signal is then toggled to 0 and the same process repeated for 1312 cycles.

3.2.3 Check Packets State

In the check packets state, the line numbers are verified. The line number for each packet is its first two Most Significant Bytes (MSB). Occasionally the camera sends a discard packet for camera synchronization. In this case, line number is sent as **xFxx** (values are in Hex as line number is 16 bits). The **x** bits are "don't cares" so discard packets are tested for by applying a bitmask of 0x**0F00** to the first byte and testing that it is equal to 0x0F. When the camera sends a video packet, the line number is sent as **xxNN** where NN is the packet ID. The packet ID indexes each packet in a video frame. The camera has a video resolution of 80 by 60. So 60 packets have to be sent for each frame. Hence for correct frame synchronization, the packets have to be transmitted by the camera in ascending packet ID; 0, 1, 2, 3, 4 till 59. This is why a 20 MHz clock rate (maximum rate for the camera) was used as the serial clock frequency. Lower clock frequencies could not keep up with the camera transmission rate and packets are transmitted out of order; 0, 1, 2, 3, 4, 0, 20, 10. With this understanding, for each packet line number, the first byte is checked to not be equal to 0x0F (discard packet check) and the second byte, is checked to be less than 59 (Packet ID check).

If a packet satisfies these checks, it is flagged as being valid and its respective pixels are sent to RAM. If a packet is not valid, the state machine branches to the delay state, then moves to the read state to get another packet.

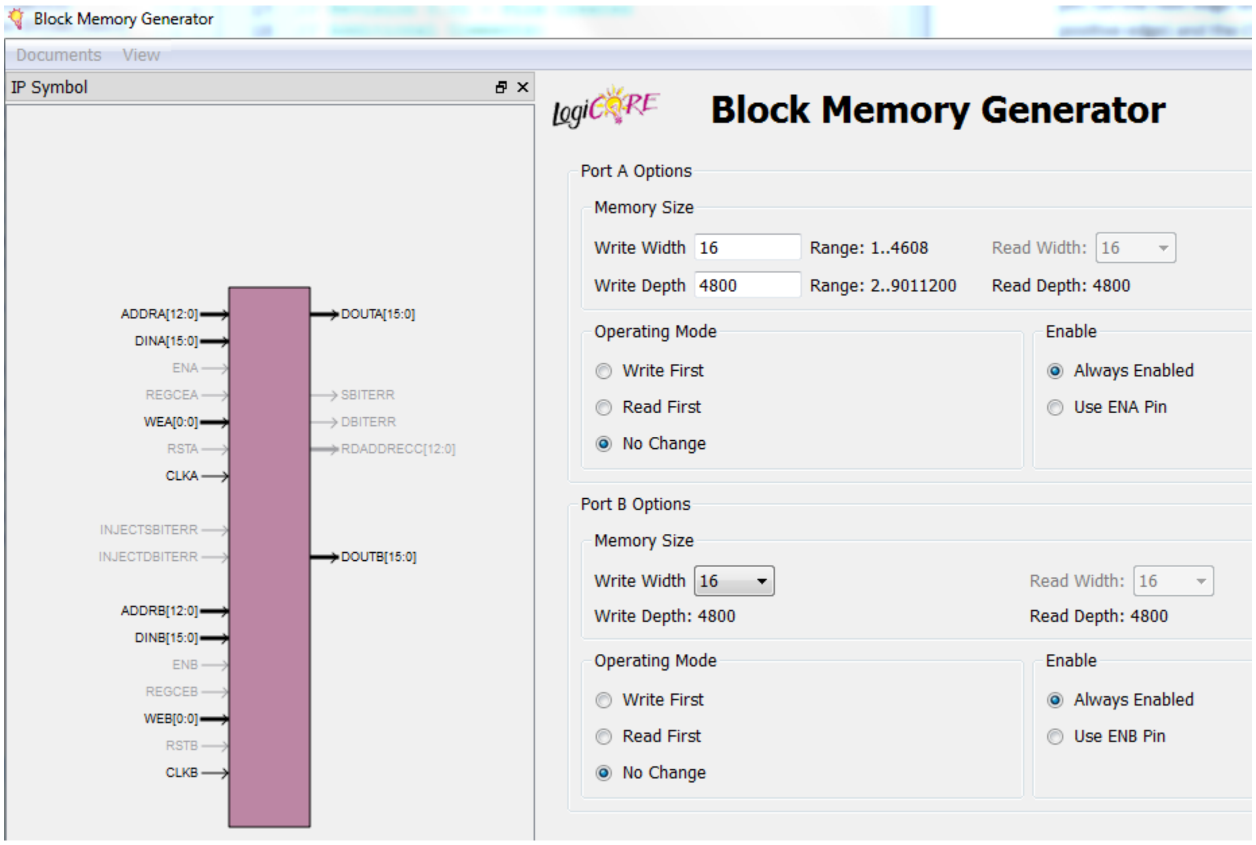
The next 160 bytes of each packet are its payload. The payload is the set of all of the pixels for each packet. Since each pixel is 16 bits in length, a payload of 160 bytes corresponds to 80 pixels per packet and this satisfies the 80 by 60 video resolution of the camera. For each packet, the payload is sampled into 80 pixels and the packet ID is used to generate the RAM write address for each respective pixel. This process is shown in the pseudo code below.

```
reg  [1279:0] Payload ;  // 1280 bits correspond to 160 bytes which are 80 pixels

reg [6:0] P_count = 0 ; //  Counter is initialized to zero, 7 bit register can handle
80 counts.

wire [7:0] Packet_ID // Packet ID number is a byte ( 8 bit) in length

   always@ (posedge clk)     //
     if(signal)      //   Signal is active for 80 clock cycles
          begin
            Pixel <= Payload[ (16*P_count) +: 16 ];  // Each pixel sampled from
payload
              Pixel_addr <= 80*(Packet_ID) + P_count ; // Each pixel RAM write
address   P_count <= P_count + 1'b1 ;  // Counter  increments from 0 to 79
             Write_enable <= 1 ;          // RAM write enable is always 1
            end
```

The pixels are sent to RAM and aligned at their respective addresses. This is the last stage of our SPI state machine.

3.2.4 True Dual Port RAM

A true dual port RAM of write depth 4800 and width of 16 bits is instantiated using the ISE IP core to buffer the pixels sent from the state machine. This instantiation is shown in Figure 10.



Block Memory Generator

Port A Options

Memory Size

Write Width 16 Range: 1..4608 Read Width: 16

Write Depth 4800 Range: 2..9011200 Read Depth: 4800

Operating Mode: Write First / Read First / [✓ No Change]

Enable: [✓ Always Enabled] / Use ENA Pin

Port B Options

Memory Size

Write Width 16 Read Width: 16

Write Depth: 4800 Read Depth: 4800

Operating Mode: Write First / Read First / [✓ No Change]

Enable: [✓ Always Enabled] / Use ENB Pin

Figure 10 - ISE IP core showing generation of the RAM block

The write depth is the actual memory size of the RAM and the write width is the data size of each element stored in the RAM. For our design, the RAM write depth is 4800 since we have to buffer 80 by 60 video frames from the camera, hence 80 * 60 = 4800 pixels. The write width is 16 because each pixel in a video packet is 16 bits in length.

3.2.5 VGA Module

While the pixels are being written to block RAM by the SPI state machine, the pixels are being read by the VGA Module for display. The VGA module consists of the VGA controller and the image processing algorithm (Image processing is depicted in a separate block in the Data flow Diagram in Figure 7 for convenience). The VGA controller generates the vertical and horizontal synchronization signal needed for the VGA monitor and outputs the pixel coordinates, pixel_x and pixel_y to the VGA address Generator module for pixel localization in RAM. A video controller diagram is depicted in Figure 11.

The video controller generates the synchronization signals and outputted data pixels serially through the VGA port of the FPGA board. The synchronization signals generator circuit (vga_sync) generates the timing and control signals. The **hsync** and **vsync** signals control the horizontal and vertical scans. The **pixel_x** and **pixel_y** signals specifies the current location of the pixel. The vga_sync circuit generates the **video_on** signal to indicate whether to enable or disable the display.

The pixels in each RAM position are read by the VGA module. The VGA horizontal and vertical signal counters are used to build an address for each VGA read from RAM.

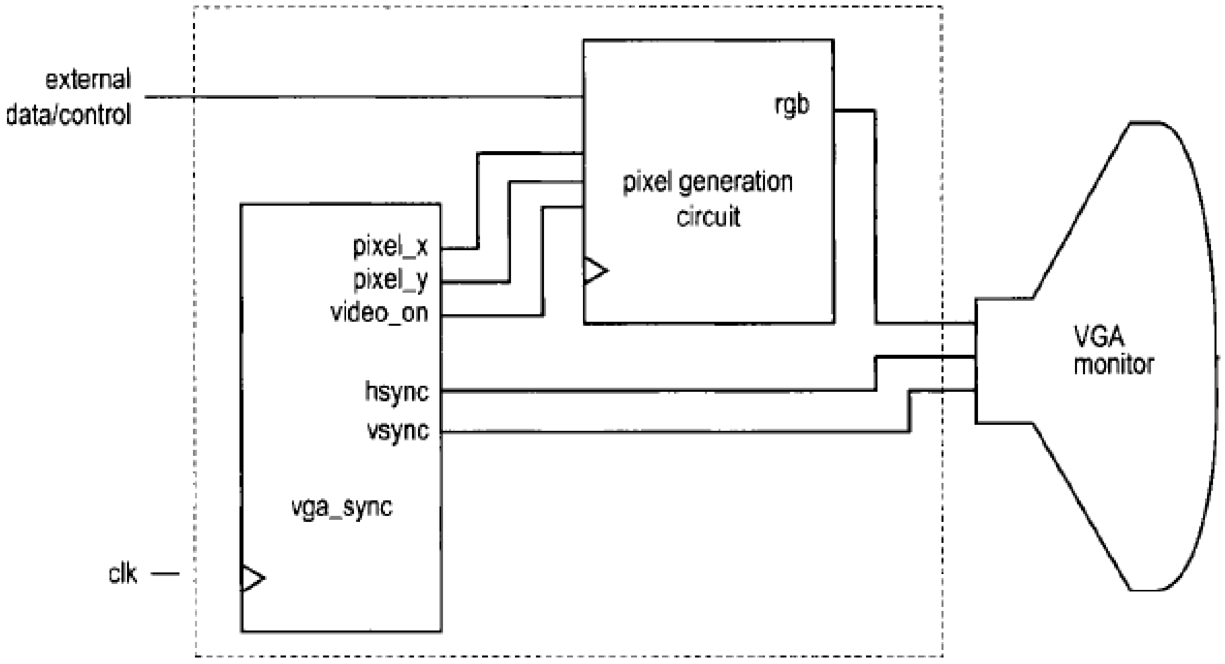


Figure 11 – Video Controller

The timing diagram of one horizontal scan is shown in Figure 12 and the timing diagram of one vertical scan is shown in Figure 13. The refresh rate for the VGA display is fixed at 60Hz. In order to achieve this refresh rate on a 640x480 pixel screen, the pixel rate calculation was done as follows:

Pixel Rate = (Total Horizontal Pixels * Total Vertical Lines *Number of screens / second)

= 800 * 525 * 60 = 25 MHz

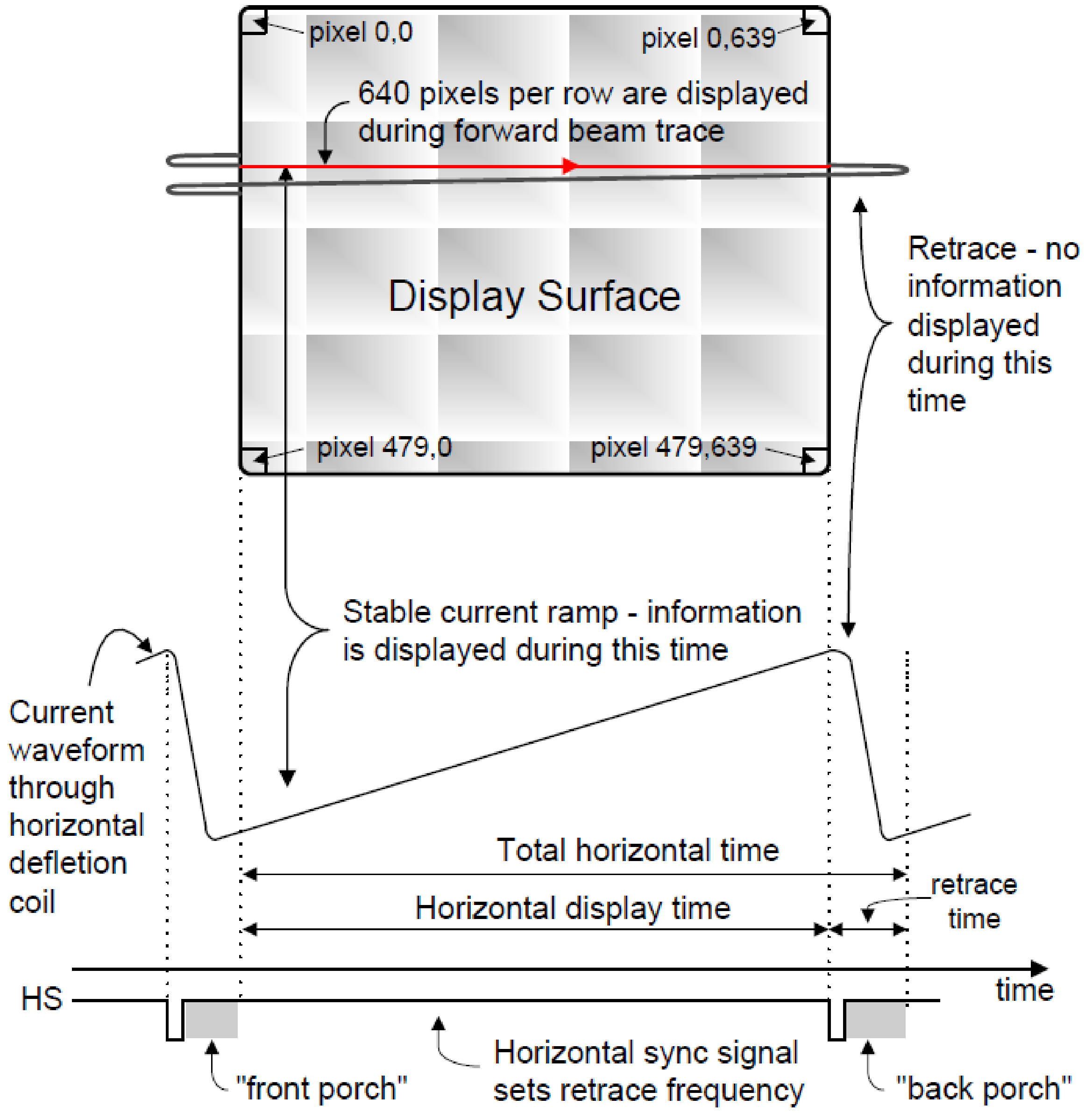

pixel 0,0
pixel 0,639
640 pixels per row are displayed during forward beam trace
Display Surface
pixel 479,0
pixel 479,639
Retrace - no information displayed during this time
Stable current ramp - information is displayed during this time
Current waveform through horizontal defletion coil
Total horizontal time
Horizontal display time
retrace time
HS
time
"front porch"
Horizontal sync signal sets retrace frequency
"back porch"

Figure 12 - Timing Diagram of the Horizontal Scan

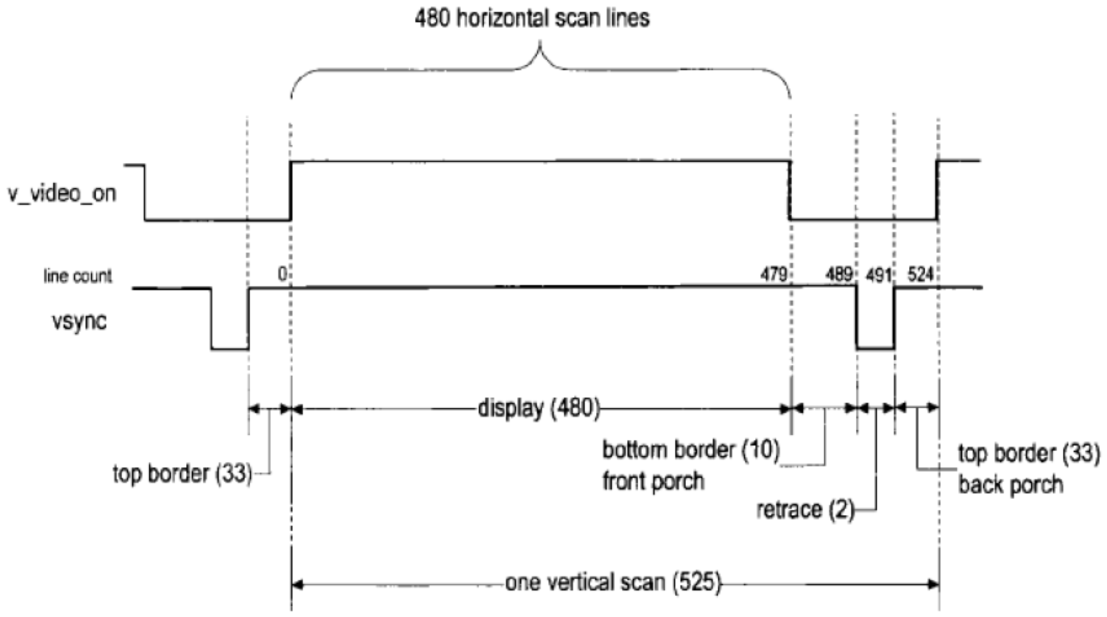


Figure 13- Timing Diagram of the Vertical Scan

3.2.6 Nearest Neighbor Interpolation

The image interpolation algorithm is actually implemented in the VGA module but for convenience, it is depicted as a separate module in the Data flow Diagram (Figure 7). Our VGA monitor has a resolution of 640 by 480 but our camera has a resolution of 80 by 60. This is a great visual limitation because only 1/64th of the monitor total area of display has to be used.

To solve this problem, the vertical and horizontal pixel coordinates are outputted 8 times by the VGA controller before being incremented. Hence each pixel in RAM is displayed 8 times horizontally and vertically. This technique enables us to scale the display from the native 80x60 given by the camera to a 640x480 VGA resolution without needing extra memory. Nevertheless, the quality of each frame is decreased and the display became blockier. Hence we trade image quality for larger frame display. Figures 14 and 15 show the comparison of the native 80x60 display when compared to the interpolated 640x480 display. Though there are noticeable blocky edges in the 640x480 display, these discrepancies do not affect the object detection algorithm.

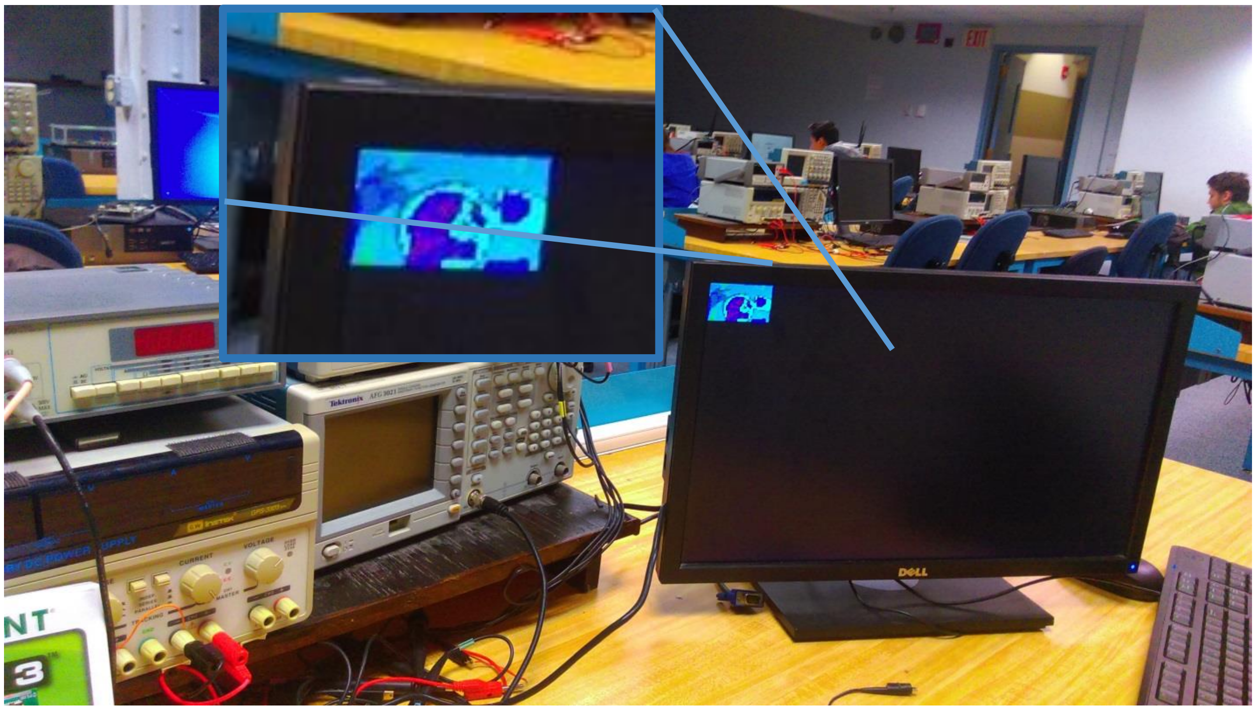


Figure 14 - Native 80 x 60 Display

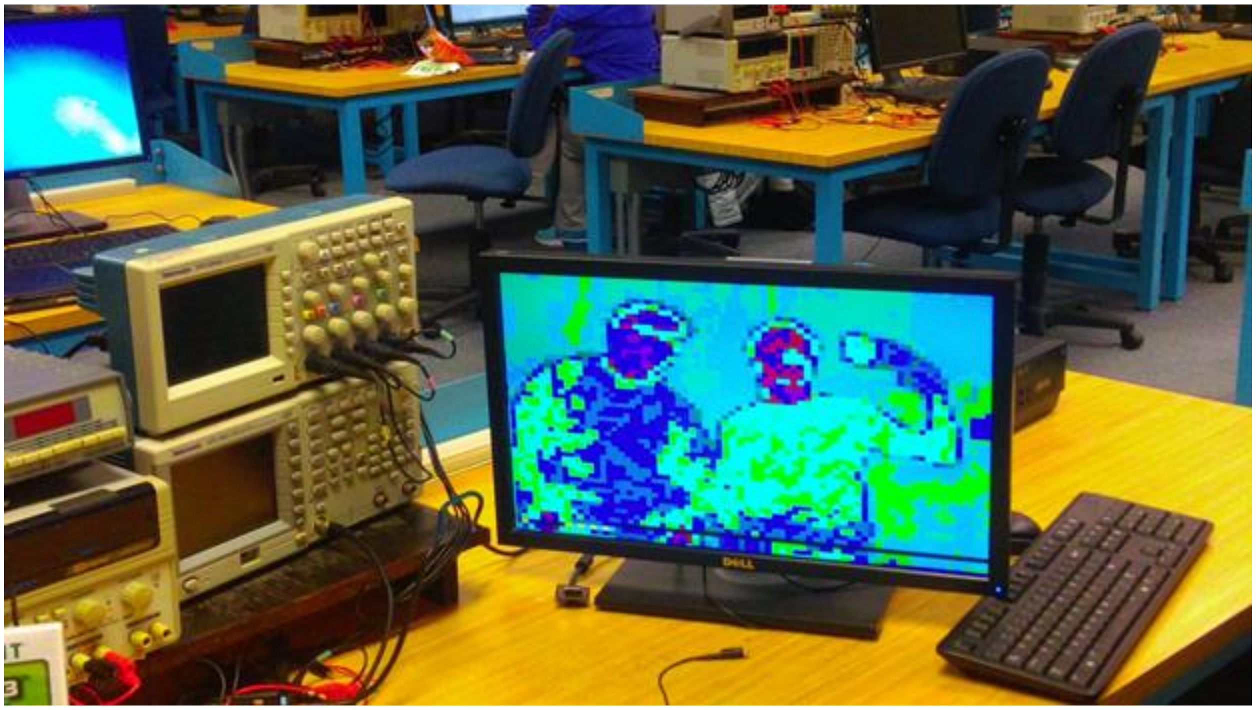


Figure 15 - Interpolated 640 x 480 Display

3.2.7 VGA Address Generator

The pixel coordinates from the VGA controller are sent to the address generator module where, the read address for each pixel is computed. The addresses are computed using the following pseudo code:

```
reg [6:0] hcnt,  // interpolated pixel coordinate from VGA controller
reg  [12:0] vcnt // interpolated pixel coordinate from VGA controller
reg  [12:0] hcount //  pixel coordinate from VGA controller
reg  [12:0] vcount //  pixel coordinate from VGA controller

reg [12:0] Addr // pixel Read address used by VGA

reg enable // read enable is always   set to 0

always@ (posedge clk)
begin

  if ( hcount >= 0 && hcount < 639  && vcount >=0 && vcount < 479  &&
enable==0 )
            begin
           if (Addr < 4800)
                Addr <= ((vcnt*80) + hcnt) ; // Address is used by VGA to read
pixel in RAM
           else
                 Addr <= 0;
```

Since the VGA controller generates the VGA signals and coordinates at 25 MHz, the VGA address generator module is also clocked at this rate. Hence the SPI state machine writes pixels to the RAM at 20 MHz using the Packet ID for write addressing while the VGA Module reads the pixels at 25 MHz using the VGA display coordinates for read addressing.

3.2.8 Alarm Output

In a real world application, having a visual display by itself would detract from our original goal of relieving some of the burden on the driver's visual attention. So, in addition to having a display of the filtered thermal camera with centroid features, we add an analog alarm circuit, as shown below. This circuit was driven by an output signal from the FPGA whenever a centroid was triggered. What followed was a short sequence of sound driven by a 555 timer to a buzzer.

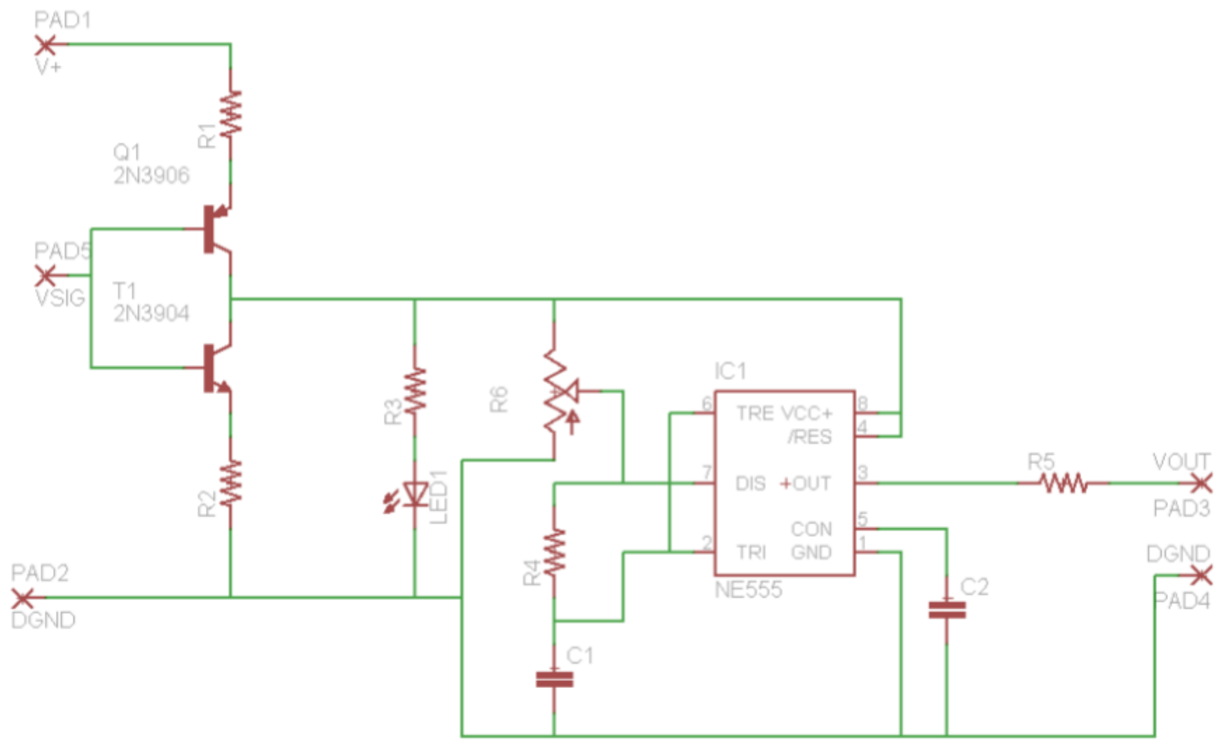


Figure 16 - Visual Representation of our Alarm Circuit

3.3 Image Processing

The Nexys 3 Spartan 6 board had an 8-bit VGA port. This means that the only 8-bit RGB (Red, Green, Blue) color signals could be outputted from board. The color order is 3 Red, 3 Green, 2 Blue. The Camera sends its video frame in Grayscale format but since we have to use only 8 bits out of the 16 bits available for each pixel, false color variations are obtained for display. Hence we have to convert each 16-bit pixel into 8 bit before processing and display. We make this conversion in Verilog using the following equation where scale is obtained as 1000 by experiments and simulations.

8bit_pixel = (16bit_pixel) * (scale)/ (32768);

Our Pedestrian Detection algorithm is based from the project "Real Time Face Detection" (see References for more information of this project) [13].

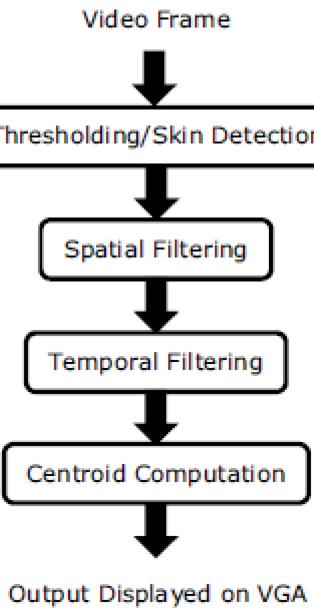


Figure 17 – Image Processing Filters

3.3.1 Thresholding

We convert each frame with 8-bit pixels into a binary image using a constant threshold value. Since we are using an infrared camera, we take advantage of the heat signature produced by the hot objects like the human skin and face. In a YUV color space, the U values (chroma values) of the skin value between a certain range. Hence we subtract the Red pixels from the Green pixels and obtain a threshold range after multiple experiments. The pseudo code for this filter is shown below.

```
wire [2:0] mgreen , mred, diff ;
    assign diff =  mred - mgreen ; // U value = Red pixels – Green pixels

  if ( diff <= 3'b010 && diff <= 3'b111 ) begin  // threshold range
             bwrgb[7:0]  <=  8’b00000001; //  color for detected pixels
                                                    end
else   begin

            bwrgb[7:0]  <=  8'b00000000;  // black color for invalid pixels
                                                  end
```

Our binary image is then passed to the spatial filter. Figures 18 and 19 show the results of the threshold filter.

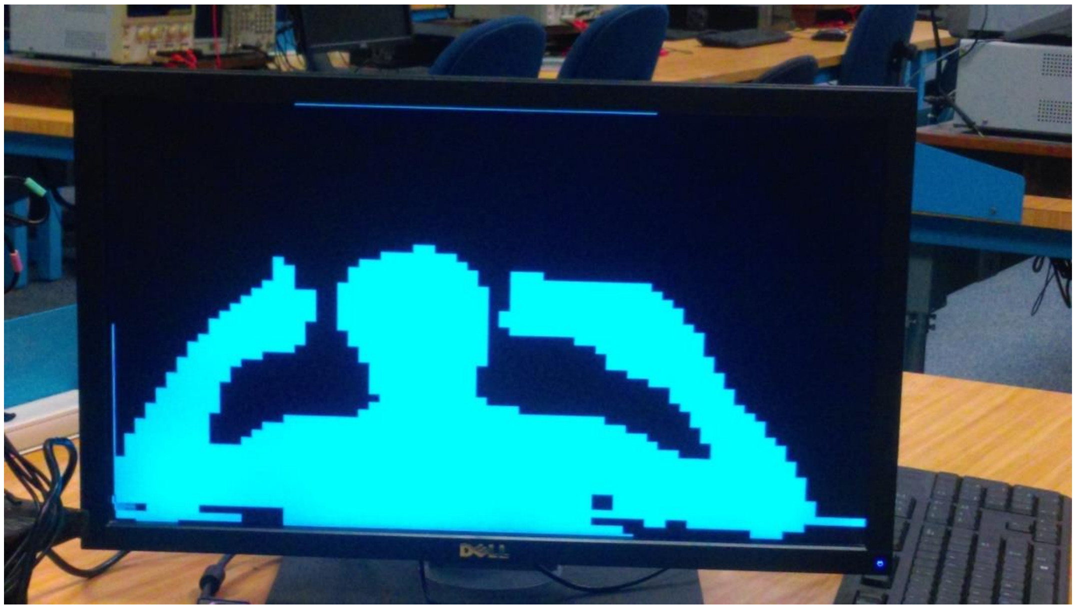


Figure 18 - Result of the Threshold filter with a single subject in the frame

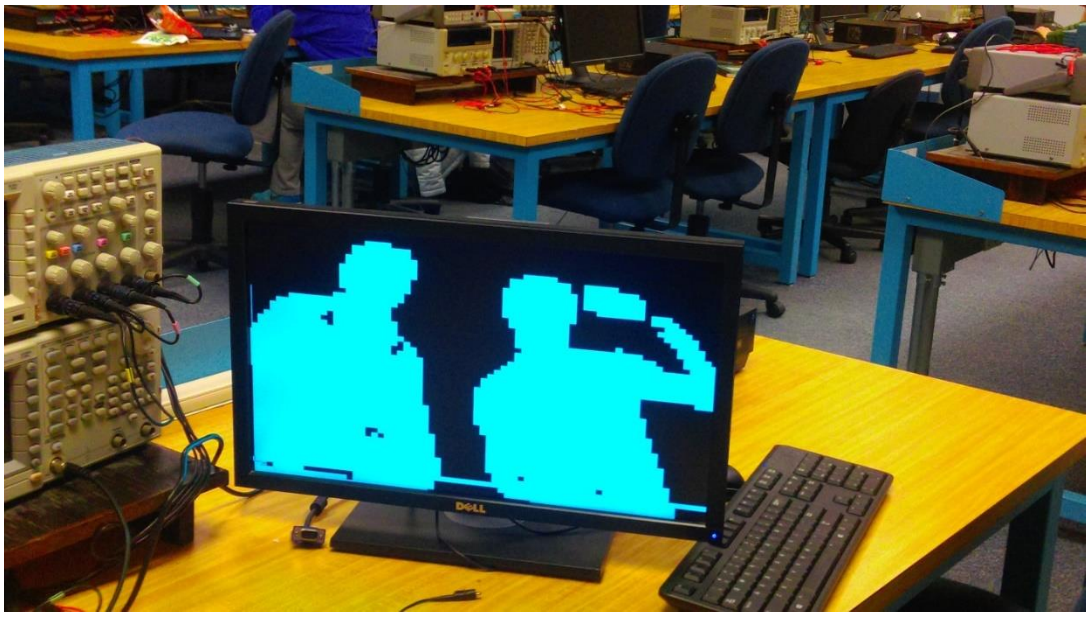


Figure 19 - Result of the Threshold Filter with two subjects in the frame

3.3.2 Spatial Filtering for Noise Removal

The spatial filter is a low pass Gaussian filter that blurs an image by attenuating the high frequency components at its edges. In order to implement this filter, for every pixel p, its neighboring pixels in a 9x4 neighborhood are checked. If more than 75% of its neighbors are valid binary pixels (binary value 1), p is also a valid pixel. Otherwise p is a non-skin pixel. This allows most background noise to be removed because usually noise scatters randomly through space.

To examine the neighbors around a pixel, their values need to be stored. Therefore, 10 shift registers are created to buffer the values of 10 consecutive rows in each frame. As shown in Figure 20, each register is 640-bit long to hold the binary values of 640 pixels in a row. Each bit in data_reg1 is updated according to the X coordinate. For instance, when the X coordinate is 2, data_reg1 [2] is updated according to the result of thresholding from the previous stage. Thus, data_reg1 is updated every clock cycle. After all the bits of data_reg1 are updated, its entire value is shifted to data_reg2. Thus, other registers (from data_reg2 to data_reg10) are only updated when the X coordinate was 0. Values of data_reg2 to data_reg10 are used to examine a pixel's neighborhood.

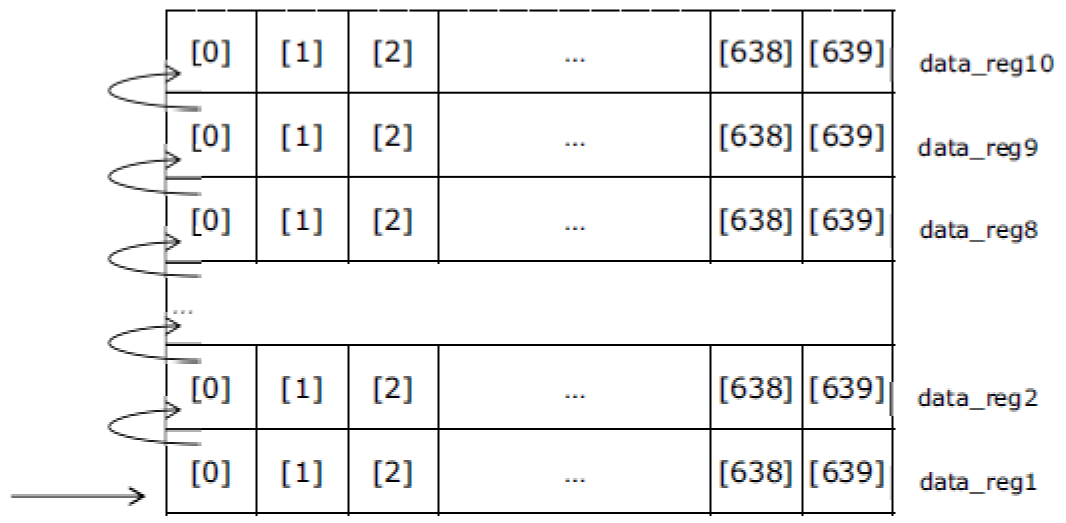


Figure 20 - Shift Register Data Flow

3.3.3 Temporal Filtering

A temporal filter is a high pass Gaussian filter applied on a sequence of images, this will blur the sequence evolution, smoothing out the temporal variation. The temporal filter is based on the following equation:

$$avg\_out = (3/4)\ avg\_in + (1/4)\ data$$

where data is the filtered result obtained from the previous stage of a pixel, namely p, in current frame, avg_in is the average value of p from previous frame, and avg_out is the average value of p in current frame

This is approximately equal to averaging four consecutive frames over time. To ease the computational effort, the equation above can be re-written as:

$$avg\_out = avg\_in - (1/4)\ avg\_in + (1/4)\ data$$

$$avg\_out = avg\_in - avg\_in >> 2 + data >> 2$$

3.3.4 Centroid Calculations

After applying the spatial and temporal filters to each pixel, we compute the centroid of each frame. We do not implement a blob labelling algorithm so we could not identify and label different objects in a single frame. Instead, we calculate the centroid of all the detected pixels in a frame. This works fine when only one object was in the camera frame, but if two or more objects are in the frame, the centroid and bounding box do not accurately describe each object. In order to ease this limitation, we extend our centroid calculations to the detection of two objects in a frame. The two objects have to be placed side by side relative to the camera. We compute the centroid for as shown in the pseudo code below.

```
always@(posedge iclk_25M) begin

        if  ( Beginning of Frame) begin     // At the start of frame,  where both
pixel coordinates are 0
                                sumX <= 30'b0;
                          sumY <= 30'b0;
                          Area <= 12'b0;
                      end

      else  if (( Valid Display range)) begin   // While the VGA signals are in the
display range
                    if( Pixel_valid) begin       // For every valid pixel ->
                            sumX <= sumX + hcount;  // 1) its horizontal pixel
coordinates is added
                          sumY <= sumY + vcount; //  2) its vertical pixel
coordinates is added
                                Area <= Area + 19'b1;    //   3) the pixel count is
incremented
                                  end
                 end
           else         begin
                      centroid_x <= (sumX)/Area;         // centroid x position is
computed
                      centroid_y <= (sumY)/Area;         // centroid y postion is
computed
```

In order to calculate the centroid for two objects the video frame is split into two according to where the centroid was, as represented in the figures below.

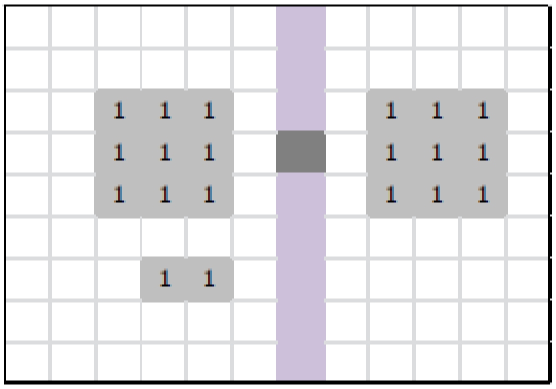


Figure 21 - Centroid (Whole Frame)

As shown in Figure 21, the frame is split into two according to the centroid position of the whole frame.

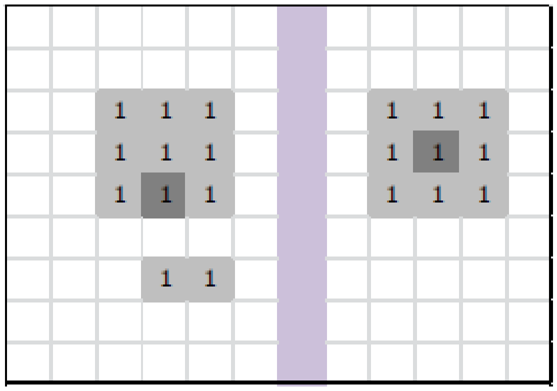


Figure 22 - Centroid (Half Frame)

As seen in Figure 22, the centroids of the objects in each half of the frame are computed independently.

Obtaining the centroid of each face region allows us to locate the face of each person present in a two-person video frame. Figure 23 shows the result of the centroid detection of a single person in a frame.

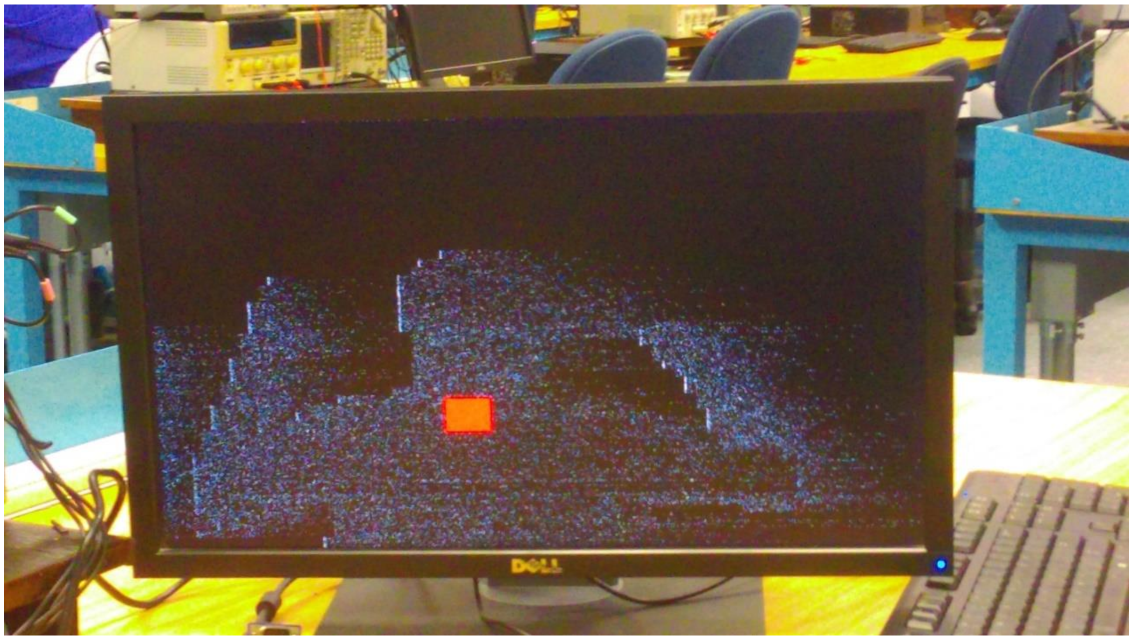


Figure 23 - Result of the centroid detection

Chapter 4 – Conclusions and Plans for Future Work

4.1 Conclusions

In conclusion, through the use of the FLIR Lepton Camera module and the Nexys 3 FPGA Board, our project team was able to successfully create a pedestrian detection module capable of detecting two pedestrians simultaneously and positively identify their position in the frame. Our current system had a detection range of approximately 15 feet, limited by the resolution of the thermal camera.

Though many camera-based pedestrian detection systems are available on the commercial market, they often use visible light cameras which are susceptible to interference from external conditions such as snow, rain or lack of lighting. An example of such a product is Subaru Eyesight, which places 3 separate cameras near the top of the user's windshield. Not only is this bulky, but it does not deal with pedestrian detection or use anything other than visible cameras. Its various features, like other products, include things such as Adaptive Cruise Control, Lane Detection, and others just to deal with other cars [13]. Our solution is a streamlined way to detect living things and warn the driver about hitting them. Through the use of an infrared camera, we were able to eliminate lighting and weather as potential interference.

The infrared camera-based pedestrian detection systems that are commercially available such as the FLIR PathfindIR are often very expensive for the common driver. The FLIR PathfindIR after-market kit for example cost in excess of $3000. This cost breaks down to $2500 for the camera module and around $750 for the Installation kit (which contains the LCD Display and the processing unit) [13]. The pedestrian detection module created during this project cost just under $500 total ($39 for the breakout board, $175 for the Lepton Camera module and $270 for the Nexys 3 board). As such this product can be seen as a cheaper alternative to the products already available.

4.2 System Constraints

Though our system is able to detect two pedestrians, there are constraints on our device when put in a real world scenario. The pedestrian detection algorithm which was implemented was only able to differentiate two people who are separately located on two sides of the image. This can be seen as a major constraint because objects will not always be positioned ideally in this setup.

Our algorithm also identified inorganic objects such as monitors and light bulbs as pedestrians because they emit heat signatures similar to that of the human body. In a real world scenario, this would not be ideal as many false positives may be incorrectly identified.

4.3 Future Improvements

For future improvements to our pedestrian detection module, we plan to use a higher resolution camera. With the use of a higher resolution camera, we would be able to detect people from farther distances as we would ascertain a larger native pixel count from the camera. We also plan to upgrade from the Nexys 3 board to a more powerful FPGA board, possibly the Zynq-7000 FPGA, as to handle the increased amount of processing that would be necessary for the larger data input. With the use of the higher resolution camera, we would also switch from using a VGA port to HDMI as this is more appropriate for video signals with high resolution.

In order to solve the second constraint of our current product, we plan to superimpose the output from a visible light camera with the output from the infrared camera. By doing this, we could use the additional pixel validity offered by the regular camera, to reduce the number of false positives (monitors, light bulbs, smart phones) detected by the algorithm. This would solve the limitations of our current product as the visual light camera would be able to distinguish pixels representing skin from other non-organic pixels, hence reducing the occurrences of false positvies which have heat signatures similar to the skin.

We also plan to implement a connected labelled algorithm in order to label valid pixels, buffer their labels in blob tables, merge similar labels, and calculate object properties of each merged object (area, centroid) and draw bounding boxes around valid pixels on the VGA display. We could not implement this due to the time constraints of our project. The development time of the SPI driver and VGA modules for the camera took a considerable portion of our project time and this confirmed the difficulty of FPGA programming and image processing in hardware.

As for the output signal from the module, in addition to the visual display, we are currently using a simple alarm circuit whenever the centroid is triggered. For future iterations, we plan to format it in a way that current control systems of cars could interpret it and act appropriately. One such way is through the use of a CAN (Controller Area Network) which would be ideal since this is one of the standards commonly used by microcontrollers in vehicles to communicate with other on-board devices.

Glossary

ADAS - Advanced Driver Assistance Systems

ASIC – Application-Specific Integrated Circuit

BRAM – Block Random Access Memory

CAN – Controller Area Network

CLB – Configurable Logic Block

FPGA – Field Programmable Gate Array

IP – Intellectual Property

IR - Infrared

LUT – Lookup Table

MSB - Most Significant Bit

Bibliography

[1] Mobile Eye, "mobileye.com - Pedestrian Detection," [Online]. Available: http://www.mobileye.com/technology/applications/pedestrian-detection/. [Accessed 1 March 2016].

[2] G. H. S. Association, "Pedestrian Trafic Fatalities by State- Prelim Report," 2014.

[3] [Online]. Available: http://www.ncbi.nlm.nih.gov/pmc/articles/PMC3821298/ .

[4] Insurance Institute for Highway Safety - Highway Loss Data Institute, "Crash Avoidance Technologies," [Online]. Available: http://www.iihs.org/iihs/topics/t/crash-avoidance-technologies/topicoverview. [Accessed 1 March 2016].

[5] Caltech, "Caltech.edu - Caltech Pedestrian Detection Benchmark," [Online]. Available: http://www.vision.caltech.edu/Image_Datasets/CaltechPedestrians/. [Accessed 2 March 2016].

[6] National Instruments, "FPGA Fundamentals," 3 May 2012. [Online]. Available: http://www.ni.com/white-paper/6983/en/.

[7] Xilinx, Inc., "Xilinx: What is an FPGA?," [Online]. Available: http://www.xilinx.com/training/fpga/fpga-field-programmable-gate-array.htm.

[8] S. Mittal, S. Gupta and S. Dasgupta, "FPGA: An Efficient and Promising Platform for Real-Time Image Processing Applications," *Proceedings of the National Conference on Research and Development in Hardware & Systems,* 2008.

[9] Xilinx, Inc., "Design Security," [Online]. Available: http://www.xilinx.com/products/technology/design-security.html.

[10] Digilent, Inc., "Nexys 3 FPGA Board Reference Manual," 10 April 2013. [Online]. Available: https://reference.digilentinc.com/_media/nexys:nexys3:nexys3_rm.pdf.

[11] SparkFun Electronics, "FLiR Dev Kit," [Online]. Available: https://www.sparkfun.com/products/13233.

[12] FLIR Systems, Inc., "Resources for FLIR Systems Cores & Components," 15 October 2014. [Online]. Available: http://www.flir.com/cores/display/?id=51878.

[13] "Real Time Face Detection and Tracking," [Online]. Available: http://people.ece.cornell.edu/land/courses/eceprojectsland/STUDENTPROJ/2012to2013/tnn7/tnn7_report_201212141110.pdf.

[14] Subaru, "Subaru Eyesight," [Online]. Available: http://www.subaru.com/engineering/eyesight.html.

[15] "FlIR.com," [Online]. Available: http://www.flir-direct.com/products/search?search=pathfindir&gclid=CjwKEAiAmNW2BRDL4KqS3vmqgUESJABiiwDTQZB_pBAC3Sk-0I0Yhz96YJ4F7Ta8p-qBv90EFxYIVhoCLUnw_wcB. [Accessed 6 March 2016].

Appendix A – Verilog Code

Main Module

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company: Worcester Polytechnic Institute
// Engineer:      Adam Karcs
//                Arsene Tchatchoua
//                Damani Walder
//
// Create Date:    14:36:38 03/25/2016
// Design Name:
// Module Name:    Main
// Project Name:
// Target Devices: Nexys 3
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
////////////////////////////////////////////////////////////////////


module Main(
input idata,
input iCLK,
input ireset,
input iswitch,
input iswitch_2,
input Creset,
output odcm,
output CS,
output Cam_clk,
output oHS,
output oVS,
output [7:0] orgb,
output sclk,
output led1,
output led2,
output led3,
output led4

 );

   wire  clk_100M, clk_25M,  clk_20M ;

   wire [12:0] Addr ,iaddr ;
```

```
    wire [15:0] dina , doutb ;

    wire [12:0] hcount ;
    wire [12:0] vcount ;
    wire blank = 0;
   wire wr, wr2 ;

    wire [6:0] hcnt;
    wire [12:0] vcnt ;



// Clock generator
SPI_dcm_2MHz B1 (
    .CLK_IN1(iCLK),      // IN
    .CLK_OUT2(clk_100M),     // OUT
    .CLK_OUT3(clk_25M),     // OUT
    .CLK_OUT4(clk_20M), // OUT
    .RESET(Creset),// IN
    .LOCKED(odcm)      // OUT
     );


// Address generator module for VGA read addresses
Address_generator U2 (
        .clk(clk_25M),
        .vcount(vcount),
        .hcount(hcount),
         .enable(wr2),
         .Addr(Addr),
        .hcnt(hcnt),
        .vcnt(vcnt)
    );

 // SPI State Machine module

SPI_module U1 (
           .clk(clk_20M),
           .reset(ireset),
           .data(idata),
           .Cam_CS(CS),
           .Cam_clk(Cam_clk),
           .ipixel(dina),
           .write(wr),
           .addr(iaddr),
          .fail(led1),
          .success(led2),
          .led3(led3),
          .led4(led4)
    );

    // true dual port ram
  RAM_IP_block R1 (
           .clka(clk_20M), // input clka
           .wea(wr), // input [0 : 0] wea
           .addra(iaddr), // input [12 : 0] addra
           .dina(dina), // input [15 : 0] dina
           .clkb(clk_25M), // input clkb
           .web(wr2), // input [0 : 0] web
           .addrb(Addr), // input [12 : 0] addrb
           .doutb(doutb) // output [15 : 0] doutb
);


VGA_thermo U4 (
         .irgb_1(doutb),
         .iclk_25M(clk_25M),
         .switch_mode(iswitch),
         .switch_mode_2(iswitch_2),
         .ireset(ireset),
        .oHS(oHS),
         .oVS(oVS),
         .orgb(orgb),
         .hcount(hcount),
         .vcount(vcount),
         .hcnt(hcnt),
         .vcnt(vcnt)
    );

endmodule
```

Address Generator module

```
module Address_generator(
 input clk ,
 input [12:0] vcount,
 input  [12:0] hcount,
 input enable,
 output reg [12:0] Addr,
 output reg [12:0] Addr2,
 input  [6:0] hcnt,
 input  [12:0] vcnt
 );
always@ (posedge clk)
begin
   if ( hcount >= 0 && hcount < 640 && vcount >=0 && vcount < 480 &&
enable==0 )
 begin
      if (Addr < 4800)
          Addr <= ((vcnt*80) + hcnt) ;
     else
         Addr <= 0;
 end
end
endmodule
```

SPI Module

```
module   SPI_module(

input clk,

input reset,

input data,

output reg Cam_CS,

output reg Cam_clk,

output [15:0] ipixel,

output write,

output [12:0] addr,

output fail,        // lights up the led U16 when a packet is a
// discard or line number is not correct.

output success    // lights up the led V16. So when V16 is lighting
//more, it means there are more success and the
//display will be good and vice versa.


);

// The clk was changed from 3.1Mhz to 2Mhz


// Variables for State machine sensitivity list


parameter S0 = 3'b000;

parameter S1 = 3'b001;

parameter S2 = 3'b010;

parameter S3 = 3'b011;

parameter S4 = 3'b100;

parameter S5 = 3'b101;

parameter S6 = 3'b110;

parameter S7 = 3'b111;
```

```
reg [2:0] next, current ;

 // variables for Camera_shift
wire Run, Go, Full ;
wire [1311:0] packet;
wire [7:0] ID, line_num ;
 wire [1279:0] pay_L;

Camera_shift M2 (
    .clk(clk),
    .reset(reset),
   .data(data),
    .Run(Run),
     .ID_bits(ID),
    .Line_num(line_num),
    .payload(pay_L)
    );


// variables for Check_packet
 wire check ;
Check_packet M3 (
    .clk(clk),
    .check(check),
    .ID_bits(ID),
    .line_num(line_num),
    .fail(fail),
    .success(success)
    );
// variables for Frame_buffers
```

```
wire signal, line_complete;


Frame_buffers M4 (
     .clk(clk),
    .image(pay_L),
    .signal(signal),
    .ipixel(ipixel),     //  image pixel for RAM 1
    .write(write),        // write enable for RAM 1
    .addr(addr),         //  pixel address for RAM 1
    .complete(line_complete),
     .line(line_num)
      );

 reg [7:0] cs_delay ;
reg [10:0] counter= 0;



// Next_state Logic Implementation

   always@(posedge clk)
begin
 case(next)

 S0 :    begin
     Cam_clk = 1 ;
Cam_CS = 1 ;
next = S1;
end
 S1 :    begin
```

```
   Cam_clk = 1;
Cam_CS = 0 ;

next = S2;
    end

 S2 :    begin
   Cam_clk = 0;
Cam_CS = 0 ;



    if (cs_delay == 30)
    begin
               next = S3 ;
               cs_delay = 0;
              end
  else
     begin
                cs_delay = cs_delay + 1'b1 ;
                next = S2 ;
                end
end

S3 :    begin
              Cam_clk = 1;
              Cam_CS = 0 ;
                if( counter == 1311) begin
                     counter = 0;
                    next = S5;
                                                   end
                else      begin
                       counter = counter + 1'b1 ;
                       next = S4;
                             end
         end


 S4 :    begin
                    Cam_clk = 0;
                   Cam_CS = 0 ;
                   next = S3 ;
            end


S5 :  begin
// at S4 if the packet is a discard packet or the line number is
//wrong, it fails the check so the state goes back to S0 to read //the
next packet.
        Cam_clk = 1;
        Cam_CS =  1 ;
            if(fail)
                    next = S7;
// if fail go to the fix packet state
            else if ( success)
// if the packet is an image packet, then the state goes to S5.At,S5,
//the packet will be sent to the frame buffer. Then the State goes
//back to S0.
                     next = S6;
            else
                     next = S7;    // condition for bad packet
// i.e  CRC Error
      end


 S6 :   begin
```

```
         Cam_clk = 1;
         Cam_CS =  1 ;
              if(line_complete)
                 next = S0;
              else
                next = S6 ;
        end

S7 :    begin
            Cam_clk = 1;
            Cam_CS =  1 ;
                 if (cs_delay == 30)
              begin
                  next = S0;
                  cs_delay = 0;
              end

              else
            begin
                    cs_delay = cs_delay + 1'b1;
                    next = S7;
            end
       end

endcase
end

 // Output Logic Implementation
assign Run = (next == S3);
assign check = (next == S5);
assign signal = (next == S6);

endmodule
```

Module Camera_shift

```
module Camera_shift(
input clk,
input reset,
input data,
input Run,
output[7:0] ID_bits,
output[7:0] Line_num,
output [1279:0] payload
    );

///////////////
// Camera data are read and fed into the shift register
reg [10:0] counter= 0;
reg [1311:0] shift_register = 0;

wire [15:0] CRC_bits;
// the first byte of each packet is its ID number, the second byte is
its line number

always@(posedge clk)
if(Run)
        begin
  shift_register <= {shift_register[1310:0], data};
// we are right shifting since we are reading the MSB first // and
shift_register is of form [1311:0]
```

```
          end


assign CRC_bits = shift_register[1295:1280]; // not used only
identified

assign ID_bits = shift_register[1311:1304];

       // assign ID_bits to ID of packet

assign Line_num = shift_register[1303:1296];

       // assign  Linenum to line number of packet

assign payload = shift_register[1279:0];


endmodule
```

Module Check_packet

```
module Check_packet(

input clk,

input check,

input [7:0] ID_bits,

input  [7:0] line_num,

output reg fail,

output reg success,

input [11 :0] l_num

);


reg  i = 0 ;


/// Since each packet has a 2 byte ID, ID_bits represents the ID of
the packet and line_num represents the line number of the packet.

always@(posedge clk)


if ( check )   // if the state is S5, check will be high

 begin
```

```
     if(ID_bits == 8'h0F)

 // if the ID is a discard (i.e == 8'h0F) or the line number is not
//equal to the expected line value, then signal a fail and do not
//change the value of the expected line value. line value goes from
//0 to 59.

       begin

          fail <= 1;

          success <= 0;

        end

     else if (ID_bits == 8'h00 && line_num < 8'd60  )

// if ID is not discard and the line number is corresponding (e.g at
//the first packet should have line number 0, so i starts at 0

        begin

          fail <= 0 ;

          success <= 1;

        end

     else

         i <= 0 ;

   end

endmodule
```

Frame Buffer Module

```
module Frame_buffers(

input clk,

input [1279:0] image,

input signal,

input [7:0] line,

output [15:0] ipixel,

output write,
output [12:0] addr,
output complete
  );

reg  [6:0]  p_count ;
reg  [15:0] p_val ;
reg wr ;
reg [12:0] p_addr = 0;

always@ (posedge clk)
     if(signal)
 begin
      p_val <= image[ (16*p_count) +: 16 ];
      p_addr <= 80*line + p_count ;
      p_count <= p_count + 1'b1 ;
      wr <= 1 ;
 end
     else
      p_count <= 0;

assign complete = (p_count == 79);
assign ipixel = p_val ;   // read the pixel output from the p_val
// register
assign write = wr ;
assign addr = p_addr;



endmodule
```

VGA Thermo Module

```
module VGA_thermo(
input [15:0] irgb_1,
input iclk_25M,
input switch_mode,
input switch_mode_2,
input ireset,
output oHS,
output oVS,
output reg [7:0] orgb,
output reg [12:0] hcount,
output reg [12:0] vcount,
output reg [6:0] hcnt,
output reg [12:0] vcnt
  );

reg [10:0] lobound = 11'd125; //0
reg [10:0] upbound = 11'd425; //16
reg blank = 0;


parameter red    = 8'b00000111; //red
parameter blue   = 8'b11000000; //blue
parameter green = 8'b00111000; //green
parameter yellow = 8'b00111111; //yellow
parameter cyan =   8'b11111000; //cyan


wire [15:0] irgb ;


assign irgb = irgb_1  ;
```

```
vga_controller_640_60 U2 (
.rst(ireset),
.pixel_clk(iclk_25M),
.HS(oHS),
.VS(oVS),
.hcount(hcount),
.vcount(vcount),
.hcnt(hcnt),
 .vcnt(vcnt),
.blank(blank)
);

reg  [6:0] h_cnt;
reg  [6:0] v_cnt ;
reg [7:0]  bwrgb , fltr3 ;
```

// STEP 1 : BACKGROUND SUBTRACTION BY USING THRESHOLD RANGE

```
wire [2:0] mgreen , mred, diff ;
assign mred =   new[7:5] ;//  {irgb[15],irgb[14],irgb[13]} ;
assign mgreen =   new[4:2]; //{irgb[12],irgb[11],irgb[10]} ;
assign diff =  mred - mgreen ;

always@(posedge iclk_25M)
    begin
if ( diff <= 3'b010 && diff <= 3'b111 ) begin //( diff <= 3'b010 &&
diff <= 3'b111 ) begin
      bwrgb[7:0]  <= cyan;
      data_reg1[hcount] <= 1'b1;;
```

```
                     end
else   begin
     bwrgb[7:0]  <=  8'b00000000;
     data_reg1[hcount] <= 1'b0;
       end
    end
```

//// STEP 2 : SPATIAL FILTERING

```
reg [639:0] data_reg1, data_reg2, data_reg3;
reg [7:0]  sprgb ;


always@( posedge iclk_25M)
     begin
 if ( hcount == 12'b0) begin
  data_reg2 [639:0] <= data_reg1 [639:0];
  data_reg3 [639:0] <= data_reg2 [639:0];
                        End
// We only used three registers to build the spatial filter because of
flip flop limitations on  the Nexys 3 FPGA. With a more powerful
board, more registers can be used, ideally ten registers.
else if
((data_reg2[hcount'd4]+data_reg2[hcount'd3]+data_reg2[hcount'd2]
+data_reg2[hcount'd1]+data_reg2[hcount]+data_reg2[hcount+'d1]
+data_reg2[hcount+'d2]+data_reg2[hcount+'d3]+data_reg2[hcount+'d4]
+data_reg3[hcount'd4]+data_reg3[hcount'd3]+data_reg3[hcount'd2]
+data_reg3[hcount'd1]+data_reg3[hcount]+data_reg3[hcount+'d1]
+data_reg3[hcount+'d2]+data_reg3[hcount+'d3]+data_reg3[hcount+'d4]
) > 7'd15) begin
sprgb[7:0] <= cyan ;


else   begin
sprgb [7:0] <= 8'b0 ;
       end
```

```
end


//// Step 3: TEMPORAL FILTERING
reg [7:0] previous_avg ,current_avg  ;
wire[7:0]  temp_in, temp_out ,temp_DQ , hold ;
wire [9:0] temp2 ;


   assign temp_DQ = hcount[0] ? temp_out : temp_in;
   assign  temp_in =  temp_DQ; // 10 bits
   assign temp_out =  temp_in - (temp_in >> 2) + (sprgb >> 2);
   assign temp2 = temp_out << 4;



 always@(posedge iclk_25M)
    begin
      if (temp2 > 10'b1110111111 ) begin
  fltr3 <= 8'hff;
// Draw centroid
    if (Area > 19'd500) begin
if ((hcount < avg_centroid_x + 19'd20) && (hcount > avg_centroid_x -
19'd20) && (vcount < avg_centroid_y + 19'd20)&&
(vcount> avg_centroid_y - 19'd20))
                                begin
             fltr3 <= red;
                                end
                        end
                                  end
     else     begin
   fltr3 <= 8'h0;
     if (Area > 19'd500)   begin
if ((hcount < avg_centroid_x + 19'd20) && (hcount > avg_centroid_x
-  19'd20) && (vcount < avg_centroid_y + 19'd20) &&(vcount >
avg_centroid_y - 19'd20)) begin

            fltr3 <= red;

                                      end

                             end

              end

   end


reg [29:0] sumX = 0 ;

reg [29:0] sumY = 0;

reg  [18:0] Area = 0 ;

reg [29:0]  centroid_x, centroid_y ;

reg [29:0] avg_centroid_x, avg_centroid_y ;



always@(posedge iclk_25M) begin

if ((hcount >= 20 && hcount < 620 && vcount >= 20 && vcount < 460
))  begin

    if (fltr3 == 8'hff) begin

       sumX <= sumX + hcount;

       sumY <= sumY + vcount;

       Area <= Area + 19'b1;

                          end

                end

else if ((hcount == 2) && (vcount == 478))

       begin

        centroid_x <= (sumX)/Area;

        centroid_y <= (sumY)/Area;

        avg_centroid_x <= avg_centroid_x - (avg_centroid_x >> 2) +
                          (centroid_x >> 2);
        avg_centroid_y <= avg_centroid_y - (avg_centroid_y >> 2) +
(centroid_y >> 2);
        sumX <= 0;
        sumY <= 0;
        Area <= 0;
      end
                       end



wire [7:0] new ;
assign new = (irgb)*(1000)/(32768);


always @ (posedge iclk_25M)
       begin
if ( hcount >= 0 && hcount < 640 && vcount >= 0 && vcount < 480
&&  !blank )
          begin
        if( switch_mode_2 )
           orgb[7:0] <= fltr3[7:0];
        else if ( switch_mode )
           orgb[7:0] <=  bwrgb[7:0] ;
        else if ( ~switch_mode && ~switch_mode_2)
           orgb[7:0] <= {irgb[2],irgb[4],irgb[6],irgb[8],
                         irgb[10],irgb[11],irgb[12],irgb[14]};
          end
else if (!blank)
           orgb <= 8'b00000000;
end


endmodule
```

VGA Controller Module

```
/*This controller includes the code for the nearest neighbor

interpolation. Each pixel is repeated 8 times vertically and

horizontally and that gives us a 640 by 480 display out of  our native

80 by 60 display without  the need for extra memory. */



library IEEE;

use IEEE.STD_LOGIC_1164.ALL;

use IEEE.STD_LOGIC_ARITH.ALL;

use IEEE.STD_LOGIC_UNSIGNED.ALL;



-- simulation library

library UNISIM;

use UNISIM.VComponents.all;



-- the vga_controller_640_60 entity declaration

-- read above for behavioral description and port definitions.

entity vga_controller_640_60 is

port(

   rst         : in std_logic;

   pixel_clk   : in std_logic;

   HS          : out std_logic;

   VS          : out std_logic;

   hcount      : out std_logic_vector(12 downto 0);
   vcount      : out std_logic_vector(12 downto 0);
   hcnt         : out std_logic_vector(6 downto 0);
   vcnt         : out std_logic_vector(12 downto 0);
   blank       : out std_logic
);
end vga_controller_640_60;


architecture Behavioral of vga_controller_640_60 is


------------------------------------------------------------------------
--
-- CONSTANTS
------------------------------------------------------------------------
--



-- maximum value for the horizontal pixel counter
constant HMAX  : std_logic_vector(10 downto 0) := "01100100000"; --
800
-- maximum value for the vertical pixel counter
constant VMAX  : std_logic_vector(10 downto 0) := "01000001101"; --
525
-- total number of visible columns
constant HLINES: std_logic_vector(10 downto 0) := "01010000000"; --
640
-- value for the horizontal counter where front porch ends
constant HFP   : std_logic_vector(10 downto 0) := "01010001000"; --
648
-- value for the horizontal counter where the synch pulse ends
constant HSP   : std_logic_vector(10 downto 0) := "01011101000"; --
744
```

```
-- total number of visible lines

constant VLINES: std_logic_vector(10 downto 0) := "00111100000"; --
480

-- value for the vertical counter where the front porch ends

constant VFP   : std_logic_vector(10 downto 0) := "00111100010"; --
482

-- value for the vertical counter where the synch pulse ends

constant VSP   : std_logic_vector(10 downto 0) := "00111100100"; --
484

-- polarity of the horizontal and vertical synch pulse

-- only one polarity used, because for this resolution they coincide.

constant SPP   : std_logic := '0';




----------------------------------------------------------------------
--

-- SIGNALS

----------------------------------------------------------------------
--



-- horizontal and vertical counters

signal hcounter : std_logic_vector(12 downto 0) := -"0000000000000";
--125// 0 (others => '0');-

signal vcounter : std_logic_vector(12 downto 0) :=  "0000000000000";
-- 240 /0




signal i : std_logic_vector(3 downto 0) :=  "0000";   -- 240 /0

signal k : std_logic_vector(6 downto 0) :=  "0000000";   -- 240 /0
signal j : std_logic_vector(3 downto 0) :=  "0000";   -- 240 /0

signal m : std_logic_vector(12 downto 0) :=  "0000000000000";   -- 240
/0


-- active when inside visible screen area.

signal video_enable: std_logic;


begin


   -- output horizontal and vertical counters

   hcount <= hcounter;

   vcount <= vcounter;

   --cnt <= counter ;

   -- blank is active when outside screen visible area

   -- color output should be blacked (put on 0) when blank in active

   -- blank is delayed one pixel clock period from the video_enable

   -- signal to account for the pixel pipeline delay.

   blank <= not video_enable when rising_edge(pixel_clk);


   -- increment horizontal counter at pixel_clk rate

   -- until HMAX is reached, then reset and keep counting

   h_count: process(pixel_clk)

   begin

      if(rising_edge(pixel_clk)) then

         if(rst = '1') then

            hcounter <= (others => '0');
         elsif(hcounter = HMAX) then
            hcounter <= (others => '0');
i <= (others => '0');
k <= (others => '0');
         elsif ( hcounter /= HMAX) then
 if( i /= 8) then
            hcounter <= hcounter + 1;
hcnt <= k ;
i <= i + 1 ;
else
  hcounter <= hcounter + 1;
i <= (others => '0');
k <= k + 1 ;
hcnt <= k + 1;
 end if;

         end if;
      end if;
   end process h_count;


   -- increment vertical counter when one line is finished
   -- (horizontal counter reached HMAX)
   -- until VMAX is reached, then reset and keep counting
   v_count: process(pixel_clk)
   begin
      if(rising_edge(pixel_clk)) then
         if(rst = '1') then
            vcounter <= (others => '0');
         elsif(hcounter = HMAX) then
```

```
          if(vcounter = VMAX) then
             vcounter <= (others => '0');
         j <= (others => '0');
         m <= (others => '0');
       elsif ( vcounter /= VMAX) then
             if( j /= 8) then
             vcounter <= vcounter + 1;
        vcnt <= m ;
        j <= j + 1 ;
 else
vcounter <= vcounter + 1;
       j <= (others => '0');
       m <= m + 1 ;
       vcnt <= m + 1;
   end if;
          end if;
       end if;
    end if;
 end process v_count;


 -- generate horizontal synch pulse
 -- when horizontal counter is between where the
 -- front porch ends and the synch pulse ends.
 -- The HS is active (with polarity SPP) for a total of 96 pixels.
 do_hs: process(pixel_clk)
 begin
    if(rising_edge(pixel_clk)) then
       if(hcounter >= HFP and hcounter < HSP) then
          HS <= SPP;
         else
            HS <= not SPP;
         end if;
      end if;
   end process do_hs;


   -- generate vertical synch pulse
   -- when vertical counter is between where the
   -- front porch ends and the synch pulse ends.
   -- The VS is active (with polarity SPP) for a total of 2 video
lines
   -- = 2*HMAX = 1600 pixels.
   do_vs: process(pixel_clk)
   begin
      if(rising_edge(pixel_clk)) then
         if(vcounter >= VFP and vcounter < VSP) then
            VS <= SPP;
         else
            VS <= not SPP;
         end if;
      end if;
   end process do_vs;


   -- enable video output when pixel is in visible area
   video_enable <= '1' when (hcounter < HLINES and vcounter < VLINES)
else '0';



end Behavioral;
```

User Constraints File

```
// fpga 100 Mhz clock
NET "iclk" LOC = V10;
NET "Cam_clk" LOC = E11;
NET "CS"   LOC = G11;


NET "idata"      LOC = F11;
NET "ireset"     LOC = T10;
NET "iswitch"    LOC = T9;
NET "iswitch_2"  LOC = V9;


NET "orgb(0)" LOC = U7; //RED0
NET "orgb[1]" LOC = V7; //RED1
NET "orgb[2]" LOC = N7; //RED2
NET "orgb[3]" LOC = P8; //GRN0
NET "orgb[4]" LOC = T6; //GRN1
NET "orgb[5]" LOC = V6; //GRN2
NET "orgb[6]" LOC = R7; //BLU1
NET "orgb[7]" LOC = T7; //BLU2



NET "oHS" LOC = N6;
NET "oVS" LOC = P7;
NET "led1" LOC = U16;
NET "led2" LOC = V16;

NET "led3" LOC = T11;
NET "led4" LOC = R11;
```