Final Report: Advanced Video Security System

ECE532: Group 7 (Liza Zoubakina, Ishraq Quayyum, Moeen Ahmed)

Index

1.0 OVERVIEW

1.1 Motivation

Computer vision has transformed security surveillance by using video cameras to detect motion, identify patterns, and report suspicious activity. In recent years, a large number of startups have capitalized on this concept to provide advanced video surveillance. Startup companies such as Umbo and Deep Sentinel have built cloud-based video security systems for businesses and apartments [1] [2]. Visual One is a home security startup which offers a smaller-scale solution at a cost of $7/month for homeowners. However, it relies on off-the-shelf cameras and loads them with its own software rack [3].

Our analysis of the video surveillance market propelled our group to engineer a small-scale home security system that would be integrated with hardware. Unlike other companies, we propose to create a small-scale security system that utilizes two FPGA boards and a camera in order to accelerate the process of computer vision while limiting the cost and resources of our solution. Although building a security system with computer vision is challenging in one semester, our group is enticed in learning about how this system would be implemented in hardware.

1.2 Goals

Our goals for this project include the following:

- Build a video surveillance system which takes in a live video feed and performs motion detection on accelerator hardware.
- Once motion is detected, our next goal is to:
  - Livestream video to the homeowner (in our case, a display monitor).
  - Send an email to the homeowner through the FPGA network.

We believe this approach to video surveillance systems will prove to be advantageous over other small-scale startups because we implement hardware accelerators in our solution which will speed up our image processing and provide more accurate results to the homeowner.

1.3 Block Diagram

The block diagram of our system can be seen in Figure 1.

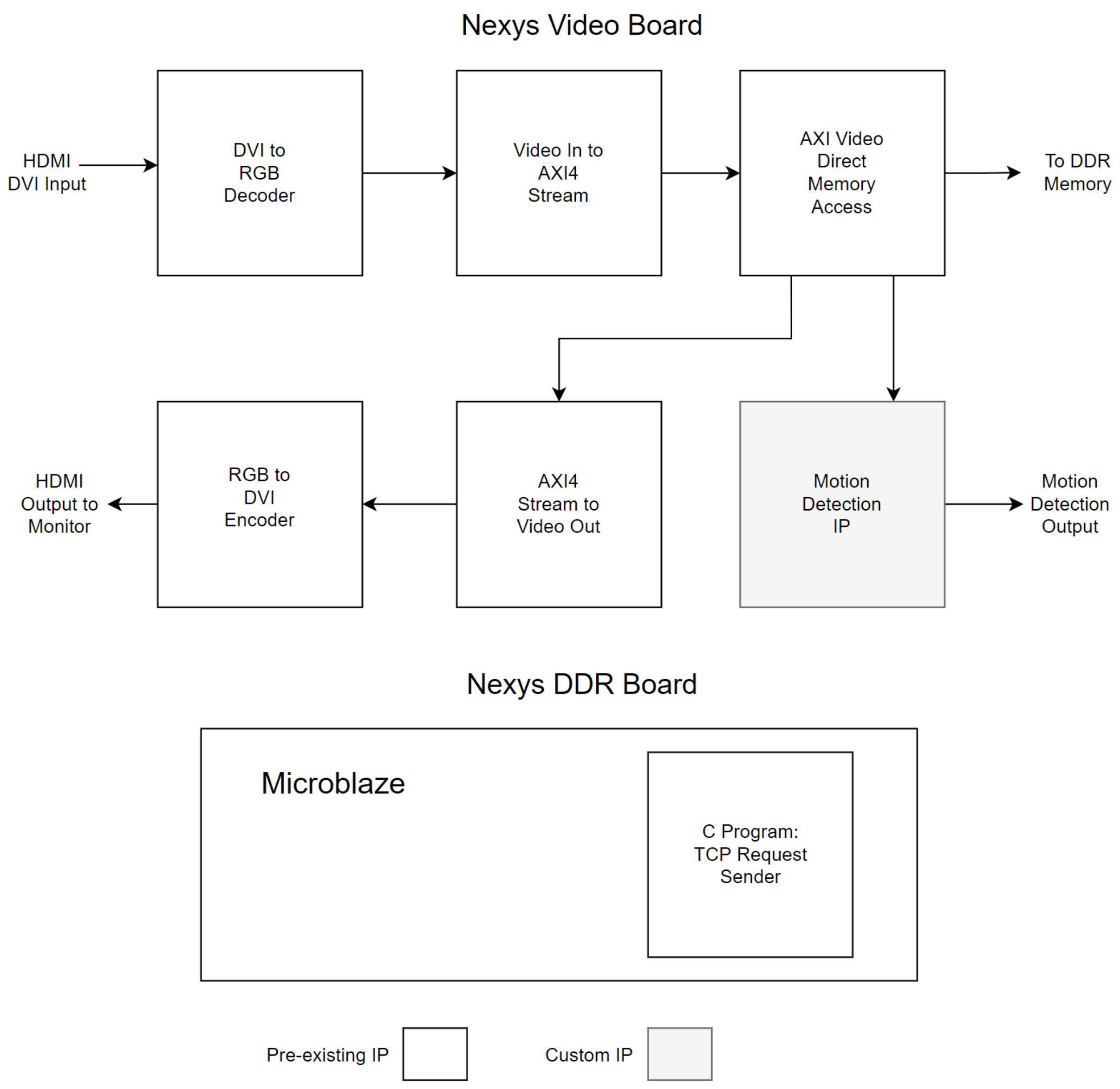

Figure 1: System overview of the video security system.

1.4 Brief Description of IPs

Table 1: Description of IPs and their functionality

IP / Block	Description	Type of IP
HDMI input	1080p HDMI Sony camera	Camera, not an IP
DVI to RGB Decoder	Deserializes input HDMI signals to 24 bit rgb format.	Existing IP

Video In to AXI4 Stream	Converts the RGBs signals into AXI4 Stream signals for the system to process.	Existing IP
Motion Detection IP	A custom IP which implements a simple motion detection algorithm which compares the color difference between two frames.	Custom IP
AXI4 VDMA (Video Direct Memory Access)	Stores the input AXI4 Stream signals composing of video frames into DDR3 memory. Provides high-bandwidth direct memory access between memory and AXI4-Stream.	Existing IP
AXI Stream to Video Out	Converts AXI4-stream signals to standard video output interface signals.	Existing IP
RGB to DVI Encoder	A video encoder which translates the 24-bit RGB format to HDMI signal.	Existing IP
Email Sender	A python script which triggers the running of a python script (to send an email) upon the arrival of a TCP packet from the Nexys Video to the Nexys DDR board.	Custom program

2.0 OUTCOME

2.1 Results

The final project met several criteria in our original plan with several modifications. Unfortunately, we were not able to integrate two major components (motion detection and the email-sending block) within the project. Overall, the project was not successful as we learned overtime that our goals were over-ambitious, and we were unable to scale down our project quickly enough to integrate the final product within our expected timelines. The details of our achievements, modifications, and failures can be found in Table 2 below.

Table 2: List of functional requirements, acceptance criteria, and the status of completeness

Functional Requirement / Feature	**Acceptance Criteria**	**Status**
Ability to obtain video input through HDMI	Correct video input should be detected verifiable using ILA	Completed
A neural-net block to perform object detection. Modified to using a simple frame-to-frame algorithm to detect motion.	Converts the RGBs signals into AXI4 Stream signals for the system to process.	Completed, but modified.
Motion Detection IP to detect movement in a video.	Output should be positive in case of moving objects in the input video	Not Completed, Tested only in

		simulation
Output video through HDMI to the display monitor.	Video exported to the display monitor must be playable	Completed
Email sending functionality	Emails should be receivable	Completed
Audio Output in case of motion detection	A speaker connected to the FPGA audio output should sound the alarm in case of motion detection	Failed (feature was taken out)
Face recognition (optional))	Input video frames should produce correct labels	Failed (feature was taken out)

2.1.1 HDMI video streaming from the HDMI-in port to the HDMI-out port

We successfully streamed an HDMI video from an input video camera to the output of the Nexys Video board using the HDMI demo project (Vivado 2016.4) created by Digilent [4]. The output stream was demoed on a display monitor, simulating the screen of a homeowner. This project has seen many revisions over the years and has multiple versions. Initially we chose to build the version corresponding to VIvado 2018.2. However, the HDMI pass through operation for this version was not working, even after attempts to debug the build of the HDMI demo. We found through several online help forums that other people were having similar issues with the 2018.2 version. After several attempts to get the project up and running, we decided to try an older version of the demo, corresponding to Vivado 2016.4, which turned out to be a success.

2.1.2 Motion Detection IP

For the motion detection IP, we were able to achieve the functionality we intended for in simulation. This custom IP takes in an AXI4-stream through a Slave AXI-stream interface and outputs the same stream through a Master AXI-stream interface. A detailed description of the motion detection algorithm can be read in Section 4.2. Due to several delays along the project, we were only able to test the IP in simulation. After being fed a few frames of pixel data in a test bench, the IP outputs a 1 signal when a significant number of pixels drastically change their colors and 0 when pixels do not change their colors.

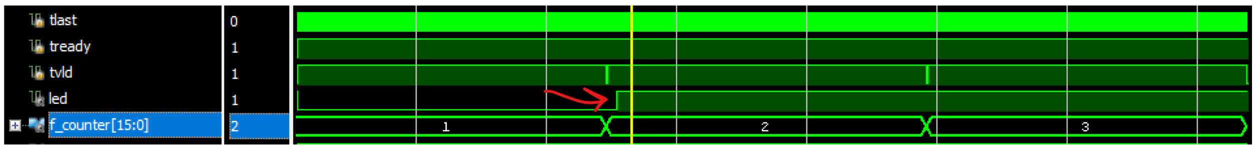

Figure 2: Simulation of Motion Detection IP. The output signal indicating detected motion (led) goes up when the second frame is being received after detecting a 1000 pixels which cross the threshold.

2.1.3 Email Sending Block

The email sending block was found to be working by the end. The functionality of the block is very simple and therefore relatively easy to test. The email was successfully received upon receiving the relevant signal from Microblaze. One issue with the current implementation is that it is not very secure. It uses the username and password stored in the form of plaintext. Furthermore, starting May 30 of this year, Google is going to be prohibiting use of third-party apps or devices which ask you to sign in to your Google Account using only your username and password. Therefore, a different more secure protocol may be more desirable to use in the future.

2.2 Possible Further Improvements

In hindsight if we could have started working using the 2016.4 version of the HDMI demo project, we could have made significantly more progress in the project.

For the Motion detection IP, we were not able to make it functional as we ran out of time. Running the IP on hard did not create a positive signal when the video camera was moved to induce motion in the video. We suspect it could have been a result of the motion threshold (45000) being too high. With minor debugging, the motion detection IP could be made functional on hardware.

Additionally we initially intended for a C program on Microblaze to accept output from the Motion Detection IP and send a TCP request to a DESL machine, which would in turn send an email to a user by running a python script. We were able to write a C program that sends a TCP request to a server running on the DESL machine. However we were not able to integrate our custom IP with this C program such that it would accept output from the motion detection IP and send a TCP request as a result.

3.0 PROJECT SCHEDULE

3.1 Initial Milestones and Actual Progress

Table 3: Comparison of our planned milestones and what was actually completed.

Milestone	Initial plan	What actually happened
Milestone 1	● Make progress in implementing YOLO CNN in Python and C.	● Researched YOLO neural net architecture, implementation. ● Created a google collab and began implementing a YOLO neural net in python (link).

	• Research camera peripheral and HDMI input block	• Researched, pretrained YOLO CNNs and transferred learning with YOLO CNN. • Researched on how to implement a base design to get HDMI video streaming • Completed extra tutorials to get a sense of other IP features within Vivado (VGA, AXI Timer).
Milestone 2	• Complete HDMI input block • Complete YOLO/CNN python/C implementation • Begin implementing YOLO/CNN neural net block	• Researched AXI interfaces in general. • Researching MIG and its input/output signals. • Completed the HDMI demo build (ported the build from 2018.2 to 2018.3) and successfully generated a bitstream after fixing upgrades and bugs. • Worked on the Audio demo as found on the Diligent website
Milestone 3	• Complete YOLO/CNN block • Complete Implementation of Image/overwrite and HDMI output block. • Begin Implementation of GUI on DESL	• Researched simple motion detection algorithms, settled on motion detection based on background subtraction (needed at least two frames of video to work). • Debugged the HDMI demo to get video stream passthrough to work between the input and output of the Nexys Video board. • Researched feasibility of different network options
Milestone 4	• Create a FIFO that allows both the AXI-Stream Slave and the Axi-Stream Master to access tdata • Get the hdmi demo working • Get the downsampling ip working • Write a script which can successfully send an email given some type of signal based on the receiving FPGA	• Learned where to place image processing IP in the HDMI demo. • Built a custom IP with AXI-Stream Slave and AXI-Stream Master interface • Debugged the HDMI demo to try to get it working by trying different configurations, 2018.3 and 2018.2 versions. Also working on trying with Vivado version 2016.2.

Milestone 5	• Integrate motion_detect IP with the HDMI demo. • Make some progress on downsampling IP • Completely working networking component	• Created a small testbench for the motion_detect IP that uses the ILA and Vivado simulation. • Ran into issues with integrating the IP with the rest of the HDMI demo as bitstream generation did not run to completion. • Researched different methods for connecting DESL and FPGA
Milestone 6	• Test working detect IP on HDMI demo. • Complete motion detection and downsampling IP • System integration for final demo	• Debugged motion detection by sharing code to be tried on the HDMI demo. • Used ILAs to probe incoming HDMI streams in an empty axi IP.

3.2 Evaluating the differences

Obviously, throughout the course of the project there have been major differences between what was originally planned to be completed for the week and what was actually completed. Near the beginning, we spent far too much time researching and evaluating the different options available to us whereas we should have picked an easy option to implement and left the rest as stretch goals. Moreover, as the term started to get busy we did not get enough done as we might have hoped and we kept putting off work to future weeks while making little progress. This left us with a great deal of work to do in the last few weeks and, with one of our team members ill, we were obviously not able to meet some of the core project requirements.

4.0 DESCRIPTION OF THE BLOCKS

4.1 Existing IPs

The HDMI demo project developed by Digilent includes pre-existing IPs which accept HDMI signals, convert them to RGB values, translate those signals into an AXI4-stream, and store them into DDR memory. This project also includes a Microblaze processor that contains a pre-existing C program which controls the HDMI output and resolution to the display monitor corresponding to the user input on the SDK terminal. We built our custom Motion Detection IP to interface with the existing IPs inside this HDMI demo. The original HDMI demo block diagram can be seen in Figure 3.

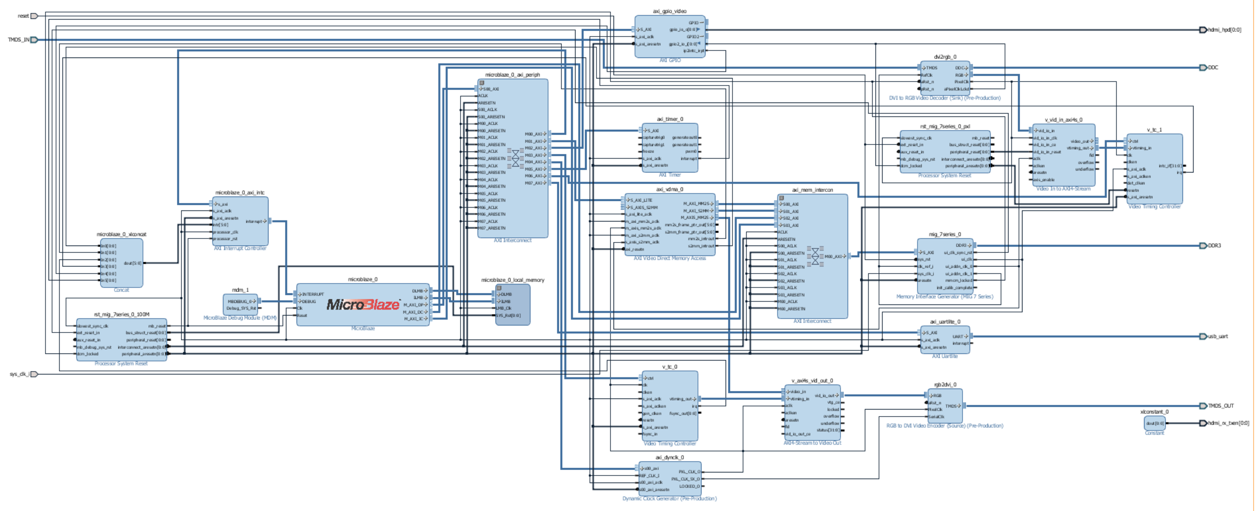

Figure 3: Diagram of the original HDMI demo, version Vivado 2016.4

The following is a list of most significant existing IPs in our design.

4.1.1 DVI to RGB Video Decoder and RGB to DVI Video Encoder

These are two pre-existing IPs which have been developed by Digilent. The ‘DVI to RGB Decoder’ IP deserializes input HDMI signals received at the HDMI input port of Nexys video board and converts them to 24-bit RGB format. Similarly, The ‘RGB to DVI Encoder’ IP serializes and converts the RGB signals into the DVI/HDMI format.

4.1.2 Video-in to AXI4-Stream and AXI4-Stream to Video-out

The IPs are also created by Digilent. Their purpose is conversion of video frame data to AXI4 stream format and vice versa. ‘Video-in to AXI4 stream’ IP converts RGB format signals into AXI4 stream format. The system uses AXI4 Stream protocol to process video signals. ‘AXI4 stream to video out’ IP converts AXI4 stream signals back into regular RGB formatted output.

4.1.3 AXI Video Direct Memory Access

The ‘AXI Video Direct Memory Access’ IP is created by Xilinx. This IP provides access to the memory though AXI4 Stream interface. AXI4 Stream video frames output by ‘Video-in to AXI4 Stream’ IP are outputted to this IP which stores these frames into DDR3 memory on the Nexys Video board. In our project, this IP only functions as a pass through IP for AXI4 Stream video frame signals which are passed onto our custom IP.

4.2 Custom IPs and Blocks

4.2.1 Motion Detection IP

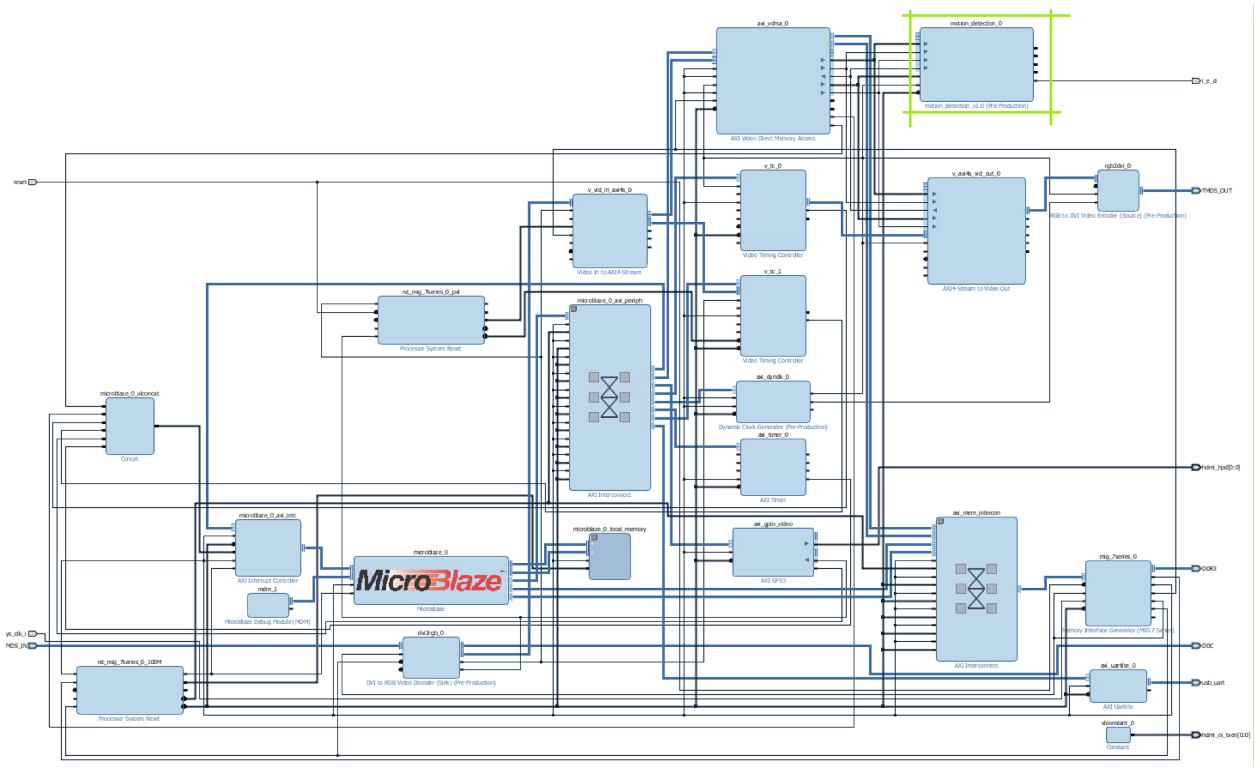

Figure 4: Block diagram of the HDMI Demo with motion detection custom IP highlighted in green.

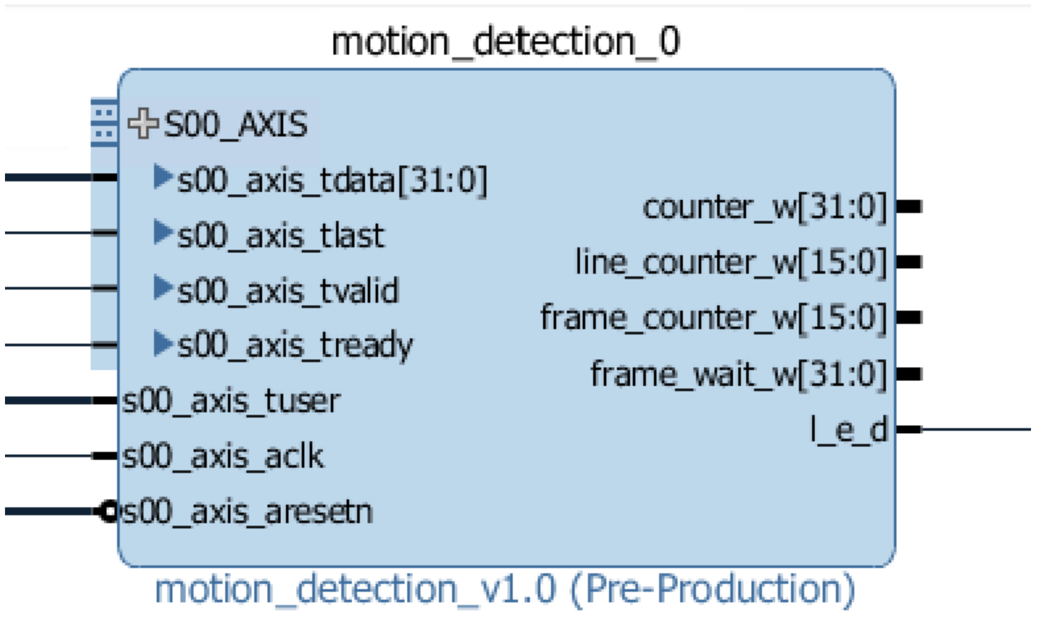

Figure 5: Close up of the motion detection IP showing input and output ports.

In this custom IP titled as "motion_detection_v1.0 ", we accept AXI4 Stream video frame data. This AXI4 stream arrives in individual video frames with a resolution of 640x480 pixels (indicated by the assertion of the `tuser` signal when a new frame arrives). Each frame can be further broken down into horizontal video lines, of which there are 480 lines. The end of a video line is indicated by the assertion of the `tlast` signal. Each video line comprises 640 pixels, which are 24-bit values that contain 3 x 8 bits to represent an RGB value. As each video frame arrives, it is stored in a Block Ram using a BRAM module created in ECE532: Assignment 2.

Table 4: Description of input and output signals of the motion detection IP.

	Ports	Direction	Description
1	s00_axis_tdata[31:0]	Input	This bus carries 24-bit pixel data. 8 bits for the red, green and blue pixels respectively. The last 8 bits are unused.
2	s00_axis_tlast	Input	This signal is asserted for 1 clock cycle whenever the last pixel of a row in a frame is being sent.
3	s00_axis_tvalid	Input	This signal is asserted whenever the AXI stream master interface has valid pixels on the tdata line.
4	s00_axis_tready	Input	This is asserted by AXI stream slave interface whenever it is ready to accept more pixels.
5	s00_axis_tuser	Input	Tuser is asserted by the master for 1 clock cycle whenever a new frame is starting.
6	s00_axis_aclk	Input	Asynchronous clock signal corresponding to the pixel clock
7	s00_axis_aresetn	Input	Reset signal
8	counter_w [31:0]	Output	A counter counts the number of clock cycles..
9	line_counter_w [15:0]	Output	A counter that counts the line number in a frame.
10	frame_counter_w [15:0]	Output	A counter that counts the number of frames completely received.
11	frame_wait_w [31:0]	Output	A counter that counts the number of clock cycles in between the ending of the first frame and arrival of the next.
12	l_e_d	Output	The signal which goes 1 when a frame is detected to have motion.

Whenever a new frame arrives, we use a pixel of the new frame and the stored pixels of the old frame to calculate the Euclidean color distance of the two pixels at the same x-y location. For this we employ the following formula:

$$\sqrt{(new\,red\ -\ old\,red)^{2} + (new\,green\ -\ old\,green)^{2} + (new\,blue\ -\ old\,blue)^{2}}$$

For this computation, we employ a total of 12 DSPs. In order to ensure the DSPs produced the correct output for the equation listed above, we created a test bench feeding different inputs to the DSPs and monitoring their output. The pipelining delay induced by the DSPs is equal to 12 clock cycles. If the output value of DSPs is greater than 45000, then we consider motion detection in the pixel and increment a ‘motion pixel counter’. After all pixels of the new frame have been processed in this way, if the counter value exceeds 1000, we consider motion detected for that frame and the motion detection IP outputs a 1 to signal motion detected.

4.2.2 Email sending block

The email sending functionality is provided by a Python script which interfaces with the C program running on the Microblaze via a TCP connection. The functionality of the network connection is derived from the network tutorial 5 as well as the network demo which were conducted earlier in the course. The Python script makes use of a Python library that is called the SMTP library which logs in with a provided username and password and then sends the email. (This poses obvious security concerns which were discussed in the ‘results’ section)

5.0 DESCRIPTION OF DESIGN TREE

The GitHub repository for this project is available at the following link: https://github.com/liza-zoubakina/ECE532-video-security-system. The following information is also available in the README.md file found at the top level of the repository.

The structure of the GitHub repository is as follows:

docs - Project documentation folder containing the mid-project presentation and the final report.

- **ECE532 Group 7 Report.pdf** - This is the final report (this document).

src - Source files for the project.

- **Nexys-Video-HDMI** - Main project built from the HDMI demo
  - **/proj/HDMI.xpr** - This is the main Vivado project file required to launch the final project that contains the modified HDMI demo.

- **ip_repo** - This is the source folder containing the IP files.

    - **motion_detection_1.0 -** This is the folder containing the component.xml file of the motion detection IP.

- **managed_ip_project** - This folder contains managed_ip_project.xpr where we can edit and repackage the motion detection IP.

- **networking/demo** - This is the source folder containing the networking components

    - **email_sending_functionality.py -** Within the folder, this is the Python script for the emailing functionality

6.0 TIPS AND TRICKS

- Try your best to integrate what you learned from the Assignments into your project, sometimes it is better to build off previously known concepts instead of starting off in a field that is completely new to you
- Learning how to divide and delegate work can be as difficult as doing the work yourself.
- Test benches and ILA can be used to debug custom IPs.
- Viewing HDMI signals using an ILA really helps to get an understanding of how the hdmi protocol works.

7.0 REFERENCES

1. *Umbo Computer Vision*. [Online]. Available: https://umbocv.ai/. [Accessed: 12-Apr-2022].

2. Anthony, Jonathan, and E. Nine, *Deep Sentinel*, 27-Oct-2021. [Online]. Available: https://www.deepsentinel.com/. [Accessed: 12-Apr-2022].

3. Coldewey, Devin, *Visual One smartens up home security cameras with object and action recognition,* 11-Mar-2020. [Online]. Available: https://techcrunch.com/2020/03/11/visualone-smartens-up-home-security-cameras-with-object-and-action-recognition/?guccounter=1&guce_referrer=aHR0cHM6Ly93d3cuZ29vZ2xlLmNvbS8&guce_referrer_sig=AQAAAJEx3QdU0IKfc9vNUw0ZISC40T_8krwl1-T1ayJr-sgidC2at3A1iUrt88v-pRp__7oSE1zeNsye4X0vzeG9zIDMyjGjjBu9BPs8iM0GLYiaPW4H8xEcZpGtguVBdEOPU06IdYB-iYf2t0NNhv8FpM2VVe67eJ8haMwt4JOApETB. [Accessed: 12-Apr-2022].

4. K, S. (2016). Nexys Video Hdmi Demo. Nexys Video HDMI Demo - Digilent Reference. Retrieved April 14, 2022, from https://digilent.com/reference/learn/programmable-logic/tutorials/nexys-video-hdmi-demo/start