ECE532: Remote Photo-Presence App

Group 7
Jackson Banbury
Kenneth Li
Eric Manget
Mahbod Mehramiz

Table of Contents

1.0 Overview

1.1 Description

The objective of this project was to explore the feasibility of augmented reality applications on an FPGA. With the growing prevalence of mobile augmented reality applications as well as the recent implementations of FPGA devices on mobile phones, the motivation for such an experiment is clear. The overall functional purpose of the project was to create an augmented reality remote ‘photo-presence’ platform that allows a user to remotely overlay a photo of their face onto a video stream, offering inclusion into a group image without the need for physical presence. The remote user takes a photo of themselves on their phone, which is transmitted wirelessly to a server connected to the FPGA, where it is then overlaid over a green indicator held by a member of the group photo. The high-level goals for the project were to obtain this functionality with a smooth framerate, and appropriate image scaling. The resulting project consists of a combination of custom-made hardware modules, a large number of modules provided by Xilinx, and software written in both C and Python. Custom hardware modules of interest include a frame buffer interface, a module to locate a green indicator in an image, a PMOD image transfer interface, and an image scaling module (which was omitted from the final design).

1.2 High-level pipeline diagram

Below is a high-level pipeline diagram of the project. The data inputs of the system are a group image captured by the HDMI camera and a ‘selfie’ image captured via an android phone. The output of the system is a merged HDMI stream of the group along with the overlaid selfie image. Descriptions of the individual modules are explain in section 4 below. Note that more low-level elements such as UART, camera-detect GPIO, etc. are omitted to offer a high-level diagram.

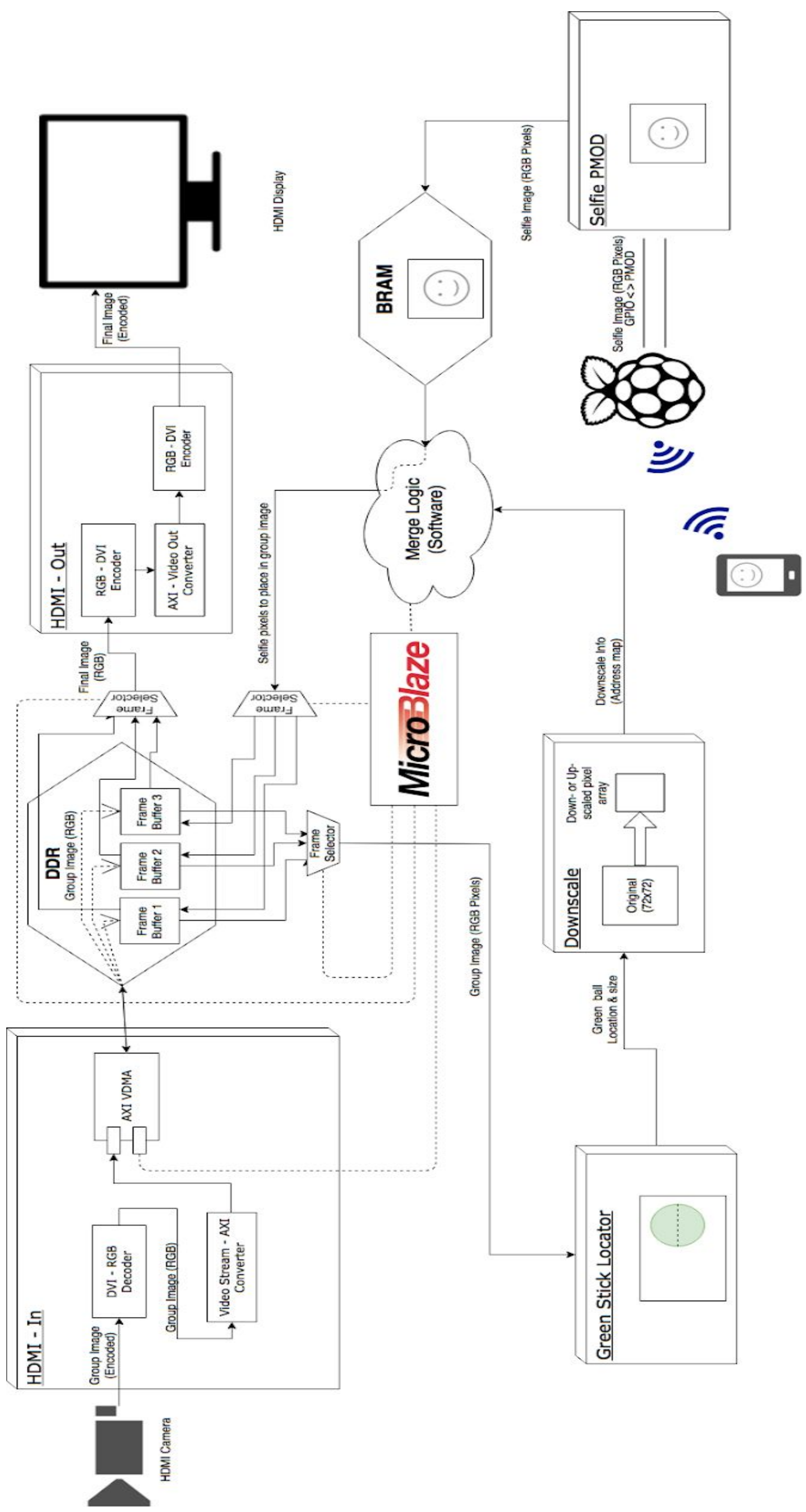


Figure 1: High-level Pipeline Diagram

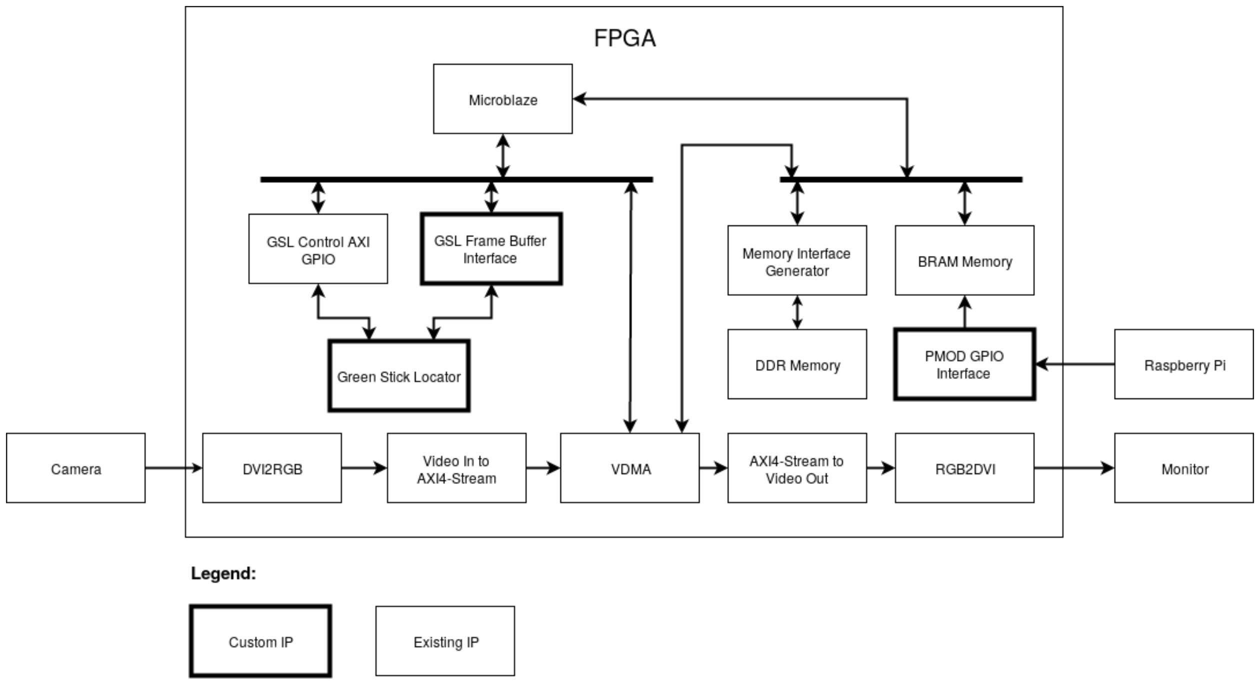


Figure 2: Hardware Module Diagram

2.0 Outcome

2.1 Results

The design team successfully implemented the majority of features and goals initially proposed. The following table provides the final status of the project.

Proposed Features and Goals	Description	Status
Transfer Selfie Image From Android Phone to Nexys Video FPGA	The design successfully sends a selfie image from an Android phone application to FPGA memory. The final design abandoned the initially proposed Ethernet connection to the FPGA in favor of a custom made PMOD interface to simplify integration.	Complete
Stream Group Video Data Into FPGA Memory	The design successfully takes advantage of an existing HDMI video camera project to stream frames into DDR3 memory. The final design abandoned the initially proposed PMOD video camera in favor of a higher quality HDMI video camera.	Complete

Locate Green Stick Indicator In Hardware Module	The design successfully incorporates a hardware module that finds a green target in an HDMI frame.	Complete
Superimpose Selfie Image Onto Group Video Frame	The design successfully superimposes selfie images onto the targeted positions in outgoing HDMI video frames.	Complete
VGA Resolution	The design successfully outputs the processed frames at 640x480 resolution to an HDMI monitor.	Complete
Accommodate Five People In Group Video	The design functions correctly when five people are present in the group video.	Complete
Prevent Green Stick Indicator Leakage	The design prevents the vast majority of the original green stick indicator pixels from being visible, except when it is held close to the HDMI camera.	Partially Complete
Downscale Selfie Image	Feature tested successfully with an algorithm in software and a tested working hardware module modelled after the algorithm was created. Hardware module was never fully integrated into final design.	Partially Complete
Frame Rate of 30 fps	The design fails to meet the initial goal of 30 fps.	Incomplete
Ensure No Noticeable Latency Between Group Video Feed and Processed Output Feed	The design fails to prevent noticeable latency between the incoming group video feed and the outgoing, processed, video feed.	Incomplete

Table 1: Final Design Status

2.2 Design Improvements

The most obvious and simple to implement improvement would be the integration of the existing downscaling hardware module. This would require fine tuning the module's interfaces with the green stick locator module and Microblaze software. Beyond this, improvements that seek to reduce the latency can be made to improve the frame rate. Primarily, this can be done by accessing DDR3 memory via a direct memory access hardware module rather than through the Microblaze software as the existing system consumes a lot of time executing instructions related to communicating with hardware modules via a custom interrupt protocol. In addition, alongside the previous improvement, the merge module can be implemented in hardware, reducing software reliance and the corresponding latency.

Another upgrade to the existing system would be incorporating functionality for streaming selfie frames, without significant latency, from an Android phone resulting in video on video HDMI output. In order for this to be accomplished the Android application needs to be modified to continuously send frames to the Raspberry Pi via wifi. In addition, the selfie image transfer rate from the Raspberry Pi to the FPGA can most immediately be improved by taking advantage of the unused PMOD ports in the JB and JC Nexys Video PMOD headers.

3.0 Project Schedule

Milestone 1:
Original : PMOD Camera, SD card , video stream app , ethernet
Actual : PMOD Camera, video stream app , ethernet(in progress)
Description : Here we found that ethernet was harder than anticipated thus it was still in progress. We dropped the SD card as well to focus on more vital parts.

Milestone 2:
Original : Hdmi output + encoder , client code (microblaze), server code + header stripper
Actual : Hdmi passthrough, ethernet(in progress), server code
Description : At the start of this milestone we dropped the PMOD Camera for the HDMI Camera due to the camera resolution. We also could not start client code as ethernet was still in progress.

Milestone 3:
Original : Pmod buffer, server buffer , green stick locator (get bounding box)
Actual : frame buffer (in progress), ethernet(in progress), green stick locator
Description : In this milestone we found that the frame buffer was harder than anticipated thus it was still in progress.

Milestone 4:
Original : Merge algo IP (+ buffer) , assembly of all subcomponents and testing
Actual : frame buffer(in progress), ethernet(in progress), Merge algo IP, downscale (software prototype)
Description : We could not start assembly of all subcomponents as the frame buffer was still in progress.

Milestone 5:
Original : Mid-project demo

Actual : Mid-project demo
Description : We needed to replace some modules with software for the Mid-project demo.

Milestone 6:
Original : Tweaks & polish
Actual : frame buffer(in progress), downscale(in progress) , ethernet(in progress)
Description : We needed to finish in progress modules.

Milestone 7:
Original : Final demo
Actual : Final demo, frame buffer, downscale , PMOD/GPIO image server, Merge algo(software)
Description : We finish in progress modules. We also replace ethernet with the PMOD/GPIO image server module

4.0 Description of the Blocks/Components

4.1 DVI2RGB

Input: HDMI Camera Data
Output: HDMI Camera Data, converted to RGB format
Description: This module is the dvi2rgb module provided by Digilent. This module takes HDMI differential signals and converts them into the RGB format with corresponding control signals [1].

4.2 Video In to AXI4-Stream

Input: HDMI Camera Data, converted to RGB format
Outputs: HDMI Camera Data, converted to AXI-Video stream format; corresponding timing information
Description: This module is the Video In to AXI4-Stream module provided by Xilinx. This module converts the RGB data into the AXI4-Stream Video protocol [2].

4.3 Memory Interface Generator (MIG)

Input: AXI Memory Requests
Output: Requested Memory Data
Description: This module converts AXI requests into DDR request to allow access of the DDR memory from AXI clients [3].

4.4 Video Direct Memory Access (VDMA)

Input: AXI Slave Interface from Microblaze, AXI4-Stream Video In
Output: AXI Master Interface to MIG, AXI4-Stream Video Out

Description: This module directly access DDR memory for quick access to frame data, allowing the camera to directly write to DDR and the monitor to read directly from DDR [4].

4.5 RGB2DVI

Input: Video Output Data, converted to RGB format
Output: Video Output Data in HDMI format
Description: This module is the dvi2rgb module provided by Digilent. This module takes the RGB formatted data and converts it into HDMI differential signals format with corresponding control signals [5].

4.6 AXI4-Stream to Video Out

Input: Video Output Data in AXI4-Video Stream format, corresponding timing information
Output: Video Output Data, converted to RGB format
Description: This module is the Video In to AXI4-Stream module provided by Xilinx. This module converts the AXI4-Stream Video data into RGB format [6].

4.7 AXI GPIO Video

Input: Lock signal from Pixel Clock
Output: Interrupt Signal to Microblaze
Description: This module is the glue logic for the Microblaze to detect when the input HDMI pixel clock has locked, allowing the software to handle the situation accordingly [7].

4.8 Input Video Timing Controller

Input: Control signals from Microblaze, Video Timing Information from converted HDMI data
Output: Interrupt signal to Microblaze
Description: This module enables detection of the timing signals from the converted HDMI in and relays this information to the Microblaze [8].

4.9 Output Video Timing Controller

Input: Control signals from Microblaze
Output: Interrupt signal to Microblaze, Video Timing Information from converted HDMI data
Description: This module enables generation of the timing signals to the Video Out data and relays this information to the Microblaze [8].

4.10 Microblaze

Description: Main processor unit of the project, allowing the execution of the software component of the project [9].

4.11 AXI Interrupt Controller

Input: Hardware interrupt signals
Output: Interrupt request to Microblaze

Description: This module handles multiple interrupt requests, allowing the Microblaze to handle them [10].

4.12 AXI Uartlite

Input: UART Bus
Output: AXI Interface to Microblaze
Description: This module converts UART requests into the AXI4 interface, allowing the Microblaze to handle them [11].

4.13 Frame Buffer Interface

Input: Memory request
Output: Corresponding response
Description: These modules allow access for our hardware modules to DDR through software. When the hardware sends an address for a pixel, the frame buffer interface interprets it and writes the address to an AXI GPIO input, which will trigger the AXI GPIO module to send an interrupt signal to the Microblaze. When the Microblaze receives the request, the software services the request and writes the requested data into the AXI GPIO output. The frame buffer interface reads the AXI GPIO output, and sends that information to the hardware module.

4.14 Green Stick Locator

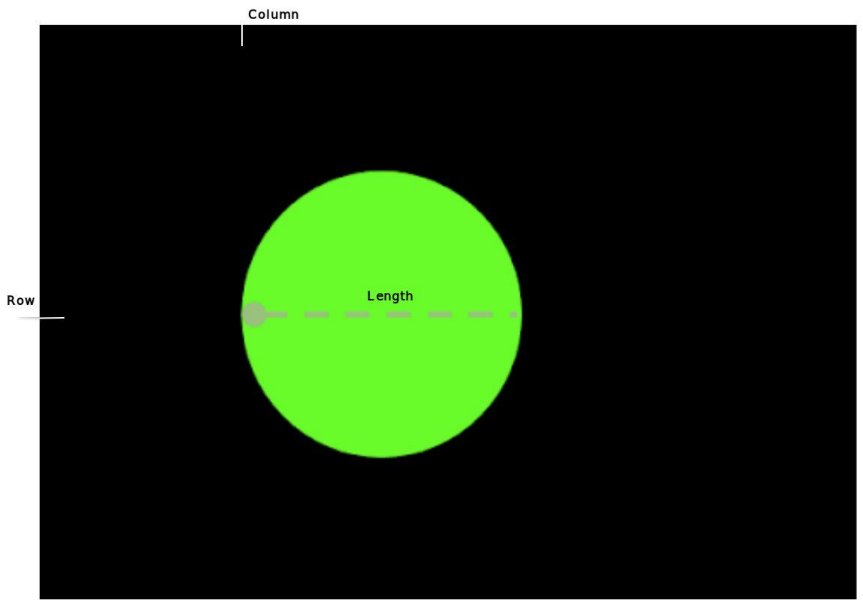


Figure 3: Green Stick Locator Parameters

Inputs: Group image
Outputs: Three parameters used to construct scaled bounding box
Description: The Green Stick Locator (GSL) module is used to identify the position and size of the green indicator in the group image so that it can be later processed for overlaying the selfie image. In order to generate the bounding box, the GSL obtains and provides three parameters

(shown above in figure 3): the length of the longest sequence of consecutive green pixels, and the starting row and column of said sequence. Once it receives a ‘begin’ signal from the MicroBlaze, the GSL requests the group image pixel-by-pixel, counting the longest sequence of consecutive green pixels in a row by using a pre-calibrated RGB threshold. If a pixel’s green value is higher than it’s red and blue value by a certain threshold, it is classified as green and a counter is incremented. With each increment, the current longest sequence of green pixels for that row is maintained, until at the end of the row that sequence is compared to the current overall longest sequence - if the new one is longer, it replaces the overall longest sequence, as well as the corresponding row and column of the beginning of that longest sequence. After parsing through all of the pixels in the image, the longest sequence and its respective starting row and column are sent to the Downscale module.

4.15 Downscale

Inputs: Cropped 72x72 selfie , Green sequence width, Green sequence position/address
Outputs : Downscaled/upscaled pixel address, done signal
Description : The downscale module takes the green sequence width and start address (the start address given is the start of the middle row of the replacement area) from the GSL to find the group photo pixel addresses to be replaced with the computed selfie image pixel addresses. This is done by calculating a ratio between the two image coordinate systems, which is multiplied by the desired downscale coordinates and rounded to get the spatially closest pixel coordinate from the original 72x72 selfie image. To overcome the need for division and the importance of floating-point precision, a .mem file was pre-created in software that offers all of the possible ratios based on the 72x72 original image, which was shifted by 8 for precision. The resulting map of group pixel addresses to be replaced by selfie image addresses are sent procedurally on a pixel-by-pixel basis. This scaling method has two significant advantages for a hardware implementation: there is only one original photo pixel address required for each pixel in the scaled image, and, more importantly, the algorithm does not actually need to see/handle any of the actual pixel data - it simply makes a map of the addresses to be used by the merge block for substitution.

4.16 Phone App

Inputs: Face
Outputs: Selfie image to Pi via WiFi
Description: The app used for taking and sending the selfie image is called IP Webcam, created by Pavel Khlebovich and available on Google Play. Given that it has all of the desired functionality for this application, using this premade app allowed for more time to be spent on hardware modules. The app is actually a live stream of the front camera that is self-hosted on a webserver on the phone, on which a single frame can be fetched from any other device on the same WiFi network via HTTP [12].

4.17 Pi Server

Inputs: Selfie from phone app

Outputs: Cropped 72x72 selfie in byte form to PMOD
Description: The Pi Server component is written in Python and run on a Raspberry Pi 3. Its purpose is to relay the full-size selfie hosted via WiFi on the phone app to the PMOD interface of the FPGA. It does this by first grabbing a single frame from the aforementioned webserver on the phone (with phone's IP address to be predefined), and converting and cropping it into a 72x72 pixel 8-bit RGB array. After raising the predefined reset GPIO pin as well as several cycles of the clock pin to signal the start of the transmission to the PMOD, the image array is then parsed across every color of every pixel, with the 8 bits of a certain color corresponding to 8 of the pins on the Pi's GPIO. For example, if a certain color was 149 out of 255, or 10010101 in binary, the 8 defined GPIO pins would be high, low, low, high, low, high, low, high, respectively. After this, the clock pin is momentarily set to high to signal the PMOD to read the 8 pins and store it as the next color. This process continues for all 15,552 colors of the 72x72 image. For a visual explanation of this pipeline, see Appendix 1.

4.18 PMOD-BRAM Interface

Inputs: Selfie image from raspberry Pi
Outputs: Selfie image to BRAM
Description: This hardware module converts byte data sent via two PMOD headers on the FPGA, JA and JB in particular, to 32 bit words containing pixel color data and writes them to a BRAM memory location that also shares access with the Microblaze. PMOD headers JA and JB, while built with different purposes in mind, can both be used as GPIO ports, connecting to the Raspberry Pi's GPIO pins, at 3.3V and frequencies less than 10 MHz [13]. The JA header reserved 8 ports for byte data, as well as two for VDD and GND. In addition, two ports on the JB header were used for an active high reset signal and a signal that acts as a clock, used for synchronization, alongside more VDD and GND connections. The Raspberry Pi first sends a reset signal for initializing FSM states in the hardware interface. The Pi then sends byte data at 3.3V and at a maximum frequency of 10 KHz, which is then saved on the positive edge of the JB port's clock signal. Once three bytes are received, and at the positive edge of a 100 MHz clock sent to BRAM, the data is placed on a 32 bit output wire, with the leading 8 bits set low, the BRAM's data address is appropriately incremented by 4, and the write enable and all four byte enable signals are set high. If another image is sent then the Raspberry Pi reasserts the reset signal allowing the hardware module's FSM to reset the address and restart the write procedure. The BRAM's interface is created with the Xilinx Block Memory Generator IP which the module directly connects [14]. The Block Memory Generator is set to dual port allowing an AXI BRAM Controller to provide the Microblaze with access to BRAM [15]. A simulation waveform can be found in Appendix 2.

4.19 Merge Software

Inputs: Fetched frame from HDMI Camera, Green Stick Locator Coordinates
Outputs: Modified frame to be sent to HDMI Monitor
Description: This software modifies the fetched group video frame with the selfie image based on the Green Stick Locator coordinates received. There are three frame buffers needed to

perform this operation, one for writing the camera frame into, one for performing the merge operation, and one for displaying to the monitor. When the hardware is ready to operate on another frame, the merge software sets the camera frame buffer to the next frame buffer and the operating frame to the original camera frame. Then the signal to perform the green stick location and downscaling is sent to hardware. During this time, the downscaling IP will send requests with two addresses, the group picture pixel address to replace and the selfie picture pixel address to replace it with. The software will continue to wait for the hardware requests until it sends a done signal, indicating that the pictures have been fully merged. Finally, the software marks the operating frame to be the display frame, and the process repeats indefinitely.
Due to the complications in using the downscale hardware, the downscaling process is not performed. So during the hardware process, when the green stick locator is finished finding its coordinates, it is sent to the software. The software determines the center point of the green stick locator, and pastes the image into the group photo, centered on that point. Everything else works as described above.

5.0 Design Tree

The most significant files are separated into categories and explained below.

Hardware Modules

- ❏ GreenLocator.v
    - ❏ Module to locate the green target in the group video frame
- ❏ Frame_buffer_interface.v
    - ❏ Module that provides the interface between Microblaze and the Green Locator module
- ❏ Pmod_gpio_interface.v
    - ❏ Module to send selfie image pixel data received via PMOD headers to BRAM
- ❏ Downscale_gpio_interface.v
    - ❏ Working but unused selfie image downscaling module
- ❏ NexysVideo_Master.xdc
    - ❏ Nexys Video FPGA constraints file
- ❏ GreenLocator_tb.v
    - ❏ Test bench for the GreenLocator.v module
- ❏ Frame_buffer_interface_tb.v
    - ❏ Test bench for the frame_buffer_interface.v module
- ❏ Pmod_gpio_interface_tb.v
    - ❏ Test bench for the pmod_gpio_interface.v module

Microblaze Software

- ❏ Video_demo.h
- ❏ Video_demo.c
    - ❏ Microblaze software needed to run the working demonstration

Raspberry Pi Software

- ❏ Selfieserver_pil2.py
    - ❏ Software which enables the Raspberry Pi to receive a selfie image via wifi from an Android phone and then send it to the FPGA

Test Modules - miscellaneous software and hardware tests for varying hardware configurations

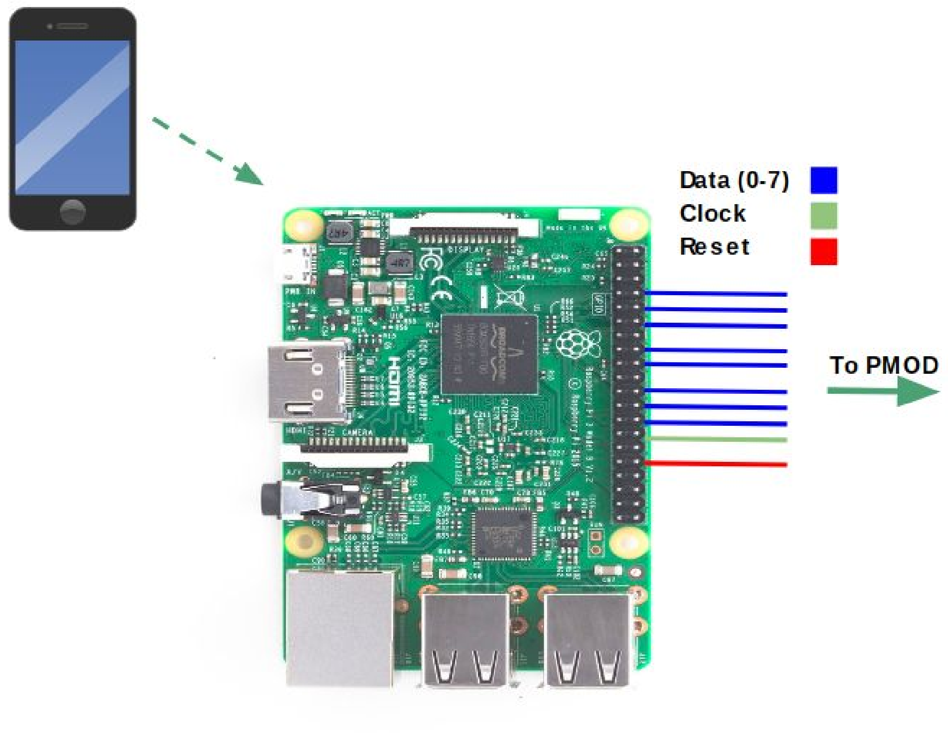
- ❏ button_debouncer.v
- ❏ frame_buffer_trigger.v
- ❏ frame_buffer_trigger_tb.v
- ❏ green_locator_trigger.v
- ❏ green_locator_trigger_tb.v
- ❏ interrupt_test.v
- ❏ interrupt_test_tb.v

Tips and Tricks

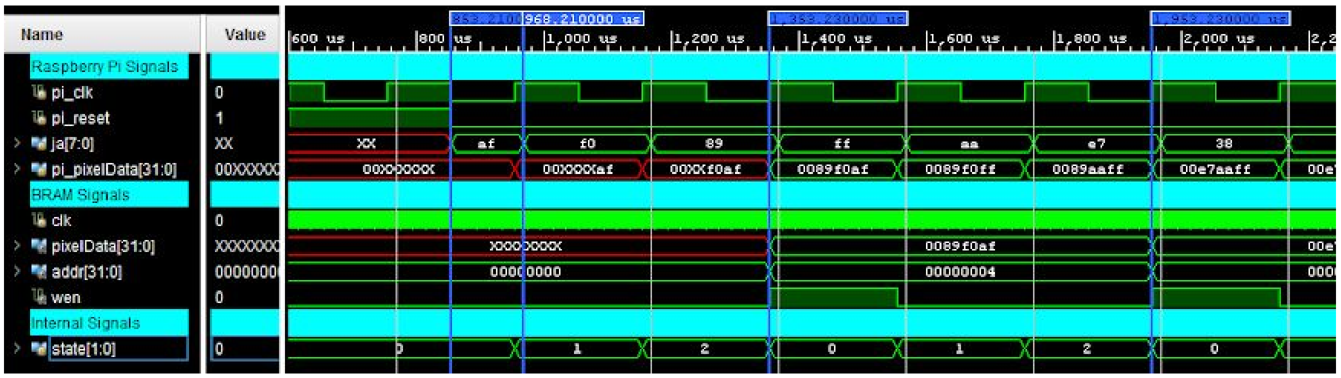
Our group learned a lot over the course of the project, and would have saved a lot of time throughout the semester if we knew at the beginning what we know now. One of the most important lessons was that trying to read and reference other projects online can actually be more difficult and take more time than simply thinking about the problem and creating one from scratch. Spending time understanding someone else’s code and trying to integrate it into our own project can take a lot more time and effort than it is worth. Another lesson was to integrate modules into the design after completion. This would have enabled us to verify the functionality in respect to the design as a whole and would have let us catch integration problems earlier, such as the lack of resources in integrating Ethernet and the difficulties in accessing memory contents through hardware.

Appendix

Appendix 1: Selfie to GPIO to PMOD on the Pi Server



Appendix 2: Testbench waveform of the PMOD interface



References

[1]"Digilent/vivado-library", *GitHub*, 2018. [Online]. Available: https://github.com/Digilent/vivado-library/blob/master/ip/dvi2rgb/docs/dvi2rgb.pdf

[2]"Xilinx Video AXI4", *Xilinx.com*, 2018. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/v_vid_in_axi4s/v4_0/pg043_v_vid_in_axi4s.pdf

[3]"Memory Interface Generator", *Xilinx.com*, 2010. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ug086.pdf

[4]"AXI VDMA", *Xilinx.com*, 2016. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_vdma/v6_2/pg020_axi_vdma.pdf

[5]"Digilent/vivado-library", *GitHub*, 2017. [Online]. Available: https://github.com/Digilent/vivado-library/blob/master/ip/rgb2dvi/docs/rgb2dvi.pdf

[6]"AXI4 Stream to Video Out", *Xilinx.com*, 2017. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/v_axi4s_vid_out/v4_0/pg044_v_axis_vid_out.pdf

[7]"AXI GPIO", *Xilinx.com*, 2016. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf

[8]"Video Timing Controller", *Xilinx.com*, 2017. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/v_tc/v6_1/pg016_v_tc.pdf

[9]"Microblaze Processor Reference Guide", *Xilinx.com*. [Online]. Available: https://www.xilinx.com/support/documentation/sw_manuals/mb_ref_guide.pdf

[10]"AXI Interrupt Controller", *Xilinx.com*, 2017. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_intc/v4_1/pg099-axi-intc.pdf

[11]"AXI UART Lite v2.0", *Xilinx.com*, 2017. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_uartlite/v2_0/pg142-axi-uartlite.pdf

[12]"IP Webcam", Pavel Khlebovich, *Play.Google.com*, 2017. [Online]. Available: https://play.google.com/store/apps/details?id=com.pas.webcam&hl=en

[13]"Nexys Video FPGA Board Reference Manual", *reference.digilentinc.com*, 2017. [Online]. Available:

https://reference.digilentinc.com/_media/reference/programmable-logic/nexys-video/nexysvideo_rm.pdf

[14]"Block Memory Generator v8.4", *Xilinx.com*, 2017. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v8_4/pg058-blk-mem-gen.pdf

[15]"AXI Block RAM (BRAM) Controller v4.1", *Xilinx.com*, 2017. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_bram_ctrl/v4_1/pg078-axi-bram-ctrl.pdf