The Edward S. Rogers Sr. Department of Electrical & Computer Engineering
UNIVERSITY OF TORONTO

ECE532: Final Design Report

MovieSync

Rakib Ahmed, Isidor Brkic, Daniel Campoverde, Luis Munoz

April 9th, 2020

Table of Contents

1. Overview

1.1. Background and Motivation

The Internet of Things (IoT) is set to change the way people interact with technology. Being home, one of the major areas where IoT redefines our activities. The MovieSync project aims to enhance the user's home video viewing experience by controlling the color and intensity of LED/lights in a room to match the video stream content. The goal is to create a more immersive and enjoyable viewing experience.
Currently, products such as the Phillips HDMI SyncBox perform similar functions, but it is only compatible with other Phillips products like their Hue lights. Our project has the advantage of versatility because it works with any set of smart lights with Wi-Fi connectivity and it accepts any HDMI input video sources.

1.2. Goal

The goal of this project was to design a system that synchronizes smart lights to the content of a video stream. The smart lights change based on the color and intensity of each video frame on the HDMI input. Whereas an HDMI output port allows the user to watch the original video stream and to apply a simple color filter to each frame in real-time. This system was implemented using a Digilient Nexys Video Artix-7 FPGA board, a Wi-Fi Pmod and a set of smart lights: Zigbee Controlled RGB LED strip and White and Color Ambiance Philips Hue bulbs.

1.3. Design Overview

The project has, at a high level, two parts: HDMI and Network based features. The HDMI part encapsulates inputting HDMI data, finding the average color and intensity, applying a filter and outputting the filtered HDMI data. The average color is readable by MicroBlaze and which filter to use is settable by MicroBlaze . The second part is the control of the LEDs/lights through a Network connection. Software running on the MicroBlaze processor can fetch the average color data and send commands that control the color and brightness of the lights via Wi-Fi that uses a TCP connection. Figure 1 shows a high level system overview of the MovieSync Project.

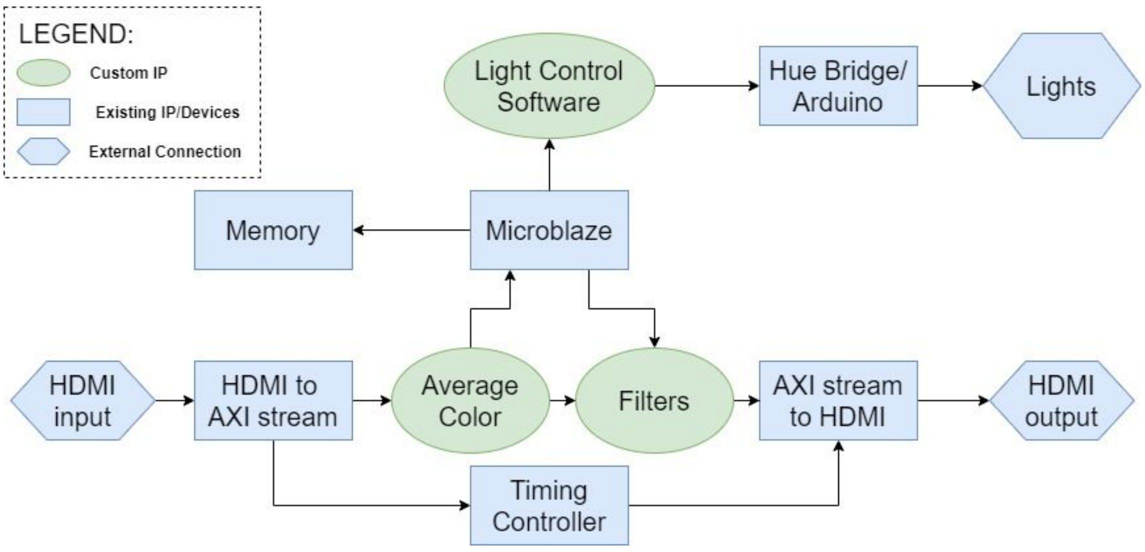

Figure 1: System block diagram.

HDMI-based Components

The HDMI part has two custom IP blocks. Both are memory mapped peripherals. The Averager IP computes the average color of a frame and stores it in a memory mapped register. The Filter IP can be written to by the microblaze to choose which filter it applies. The other blocks required to pass the HDMI stream, such as the HDMI/AXI stream converters and the Timing Controller, consist of IPs from both Xilinx and Digilent.

Network-Based Components

- **Wi-Fi LEDs/Lights Control**

The light control requires a MicroBlaze as well as other IPs from Xilinx and Digilent, especificifically, the MIG core IP and the Wi-Fi Pmod IP, respectively. The MIG IP core is used to increase the size of the instruction memory since block RAM in the FPGA is not sufficient for the size of the application. The custom part of this side is the software running on the MicroBlaze. To enable the wireless communication, this software sets up a TCP connection and sends HTTP requests to the Philips Hue bridge. The Hue bridge then updates the color and brightness of the lights wirelessly.

- **Ethernet LEDs Control**

The Ethernet components are similar to the Wi-Fi implementation because they also require the Microblaze and other IPs to achieve color control of the LEDs. For the network connectivity, this part of the design requires use of the RGMII port in an AXI Ethernet Gigabit Subsystem, and a DMA to interface with the microblaze and MIG. The software running in the Microblaze establishes a TCP connection to a python server and sends commands to control an RGB LEDs strip connected to an Arduino UNO board. The connection between the python server and the Arduino is achieved by using the Firmata communication protocol.

2. Outcome

The HDMI side features work as proposed. We implemented a custom IP block to get the average color (from this we can infer an average intensity too). We implemented filters both simple and more interesting. Finally the HDMI related features run as line rate for 1080p video. Overall, this part of the design successfully implemented and integrated all the HDMI related functionality.

The Network side of the project was also successfully implemented. We were able to create projects using each of Ethernet and Wi-Fi which were able to set the color and intensity of two lighting devices. The software loops updating a lighting strip, for both Network implementations, and then updating a light bulb, for Wi-Fi only. The latest can separately control these lighting elements as desired.

We were also able to successfully integrate the Wi-Fi project with the HDMI part. The application controls the color and brightness of an LED strip and a Phillips Hue smart-light bulb. The color is obtained from the Averager IP block and then it is sent as commands to the lightning devices at an average of 20Hz.

We had difficulties integrating the Ethernet and the HDMI parts. The HDMI project uses the MIG to generate the required 100MHz Microblaze clock and additional clocks for other components. Ethernet requires a 125MHz clock but the MIG can generate a 125.49 MHz clock. Instead, the required clock could be generated using the Clock Wizard. However, the presence of both the MIG additional clocks and the Clock Wizard causes a placement error in Vivado. It is possible that with more time we could have run the HDMI project using the Clock Wizard as the HDMI project should not require the storage of the MIG.

The issues with the Ethernet integration plus the advantages of Wi-Fi and existing Philips Hue APIs were the criteria to use the Wi-Fi implementation in our final demonstration. The final design set-up is shown in Figure 2.

Future work for the HDMI part could be more filters, and for the Network and light control part could be enhanced control over more lightning devices. As another feature, we could have chosen to compute luma (brightness) explicitly, rather than implicitly, extracting it from the average color. This would be a straightforward thing to do, one might find the intensity of each pixel and average those in the same way as we found the average color. The averaging functionality could be improved to compute local averages in addition to the global frame average by partitioning the entire frame into blocks. Coupled with LED strips and more smart-lights bulbs that allow for different sections of lights to be driven to different colors, this would account for an interesting next step for the project: driving different lightning devices with colors from different sections of the screen. For the light control we could also implement algorithms that match the filters to set specific ambiances.

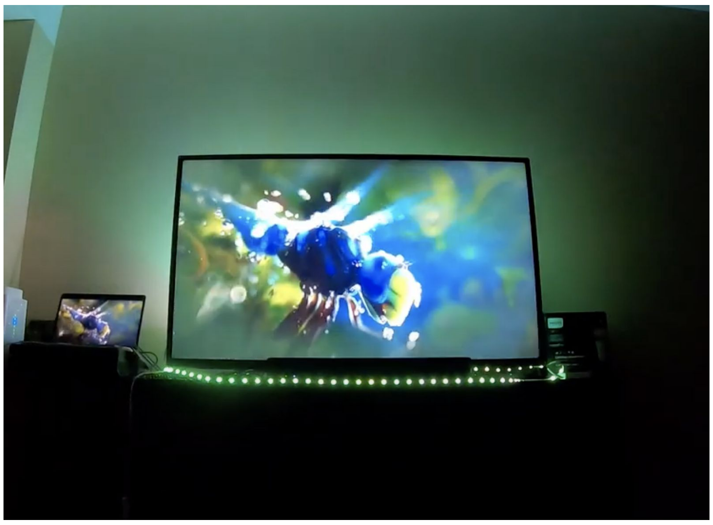

Figure 2: The entire project functioning with lights and a vignette filter.

3. Project Schedule

The following table shows how our actual progress compares to the milestone plan put forth in the proposal.

	Proposed goals	**Progress**
1	Block diagram. Background research.	Created a block diagram. Found a sample HDMI project.
2	Implement HDMI passthrough. Control lights using a python script	Modified sample project. Incorrectly thought that HDMI passthrough was working. Began work on Averager block. Researched ways to control lights (via Arduino board or Hue bridge). Found Philips Hue API specifications. Began researching how to use the Wi-Fi pmod.
3	Create Averager block and insert it into the HDMI stream.	Determined that HDMI passthrough was not working and why the sample project was running at 720p. Developed a system to send a start command over ethernet upon data input from switches and buttons. Developed a script to control brightness and color of lights through http commands. Began implementing a project to send http requests through the Wi-Fi pmod.
4	Create a simple filter and insert it into the HDMI stream.	Successfully implemented HDMI passthrough at 1080p. Implemented a greyscale filter. Began work on implementing more fine grained averaging (for sections of the screen). Controlled lights via ethernet from the board. Turned off/on lights over the Wi-Fi pmod.
5	Control the lights from the Microblaze via ethernet and a python script.	Implemented a vignette filter. Initially, changed the ethernet protocol from TCP to UDP, but went back to TCP because of issues with UDP connection to the network.

		FPGA board can control the color, saturation and temperature of the lights through Wi-Fi.
6	Implement Wi-Fi communication to control the lights.	Implemented vignetting to fade to either black or to the average color. Working on integrating Ethernet with HDMI project. Added more customization options to the python server. Integrated HDMI project with Wi-Fi project.
7	Implement more interesting filters. Complete integration.	Wi-Fi selected as best option for light control because of existing light control protocols and issues with Ethernet integration Added GPIO AXI interface to choose filters being applied to the video feed. Improved robustness of the connection handshake. Finished integration.

Table 1: Comparison of proposed milestones to group accomplishments over the project.

We found that we set forth a reasonable amount of work since we were kept busy for the whole term, but were also able to complete the project. In our planning we seem to have thought too linearly. As in only working on one piece of the project at once. We found over the weeks that it was difficult for too many people to work on the same piece of the project at once. What worked better was to start some of the later pieces earlier and in parallel. This meant that some of the earlier milestones took longer than expected, but work had already begun on later milestones allowing them to be completed quicker.

4. Description of the Blocks

4.1. HDMI

We implemented HDMI passthrough using the following IPs. Some of the more important parameters for the IPs are recorded here.

IP	VLNV	Parameters/Changes
DVI2RGB	digilentinc.com:ip: dvi2rgb:2.0	TDMS clock range: >=120MHz Prefered resolution: 1920x1080
Video In to AXI4-Stream	xilinx.com:ip: v_vid_in_axi4s:4.0	Pixels per clock: 1 Video format: RGB Native video input component width: 8 AXI4S video output component width: 8 FIFO depth: 1024 Clock mode: common
Video Timing Controller	xilinx.com:ip: v_tc:6.1	Max clocks per line: 4096 Max lines per frame: 2048 Enable generation: yes Enable detection: yes Auto generation mode: yes Video mode: 1080p
AXI4-Stream to Video Out	xilinx.com:ip: v_axi4s_vid_out:4.0	Clock mode: Common Timing mode: Slave FIFO depth: 1024
RGB2DVI	digilentinc.com:ip: rgb2dvi:1.4	Generate SerialClk internally from pixel clock: no
Constant	xilinx.com:ip: xlconstant:1.1	

Table 2: IPs required for HDMI passthrough.

The *NexysVideo_Master.xdc* constraints from the Digilent HDMI sample project for 2018.2 were used [1]. The only change was to the TMDS clock.

```
create_clock -period 8.333 -name tmds_clk_pin -waveform {0.000 4.166} -add [get_ports TMDS_IN_clk_p]
```

It was non-trivial, but important to connect *vtg_ce* on the *AXI4-Stream to Video Out gen_clken* on the *Video Timing Controller*.

4.2. Microblaze

We implemented a Microblaze to orchestrate the Ethernet/Wi-Fi connections, set the filter and to read the average color. Some of the more important parameters for the Microblaze and its associated IPs are recorded here.

IP	VLNV	Parameters/Changes
Microblaze	xilinx.com:ip: microblaze:10.0	Enable exceptions: no Enable local memory bus instruction interface: yes Enable local memory bus data interface: yes Enable peripheral AXI instruction interface: no Enable peripheral AXI data interface: yes
Memory Interface Generator	xilinx.com:ip: mig_7series:4.1	Generate a 200MHz output clock. Enable peripheral AXI Instruction Interface: yes M_AXI_IP port connected to memory slave
Microblaze Debug Module	xilinx.com:ip: mdm:3.2	N/A
Local Memory Bus	xilinx.com:ip: lmb_v10:3.0	N/A
LMB BRAM Controller	xilinx.com:ip: lmb_bram_if_cntlr: 4.0	N/A
Block Memory Generator	xilinx.com:ip: blk_mem_gen:8.4	N/A
Processor System Reset	xilinx.com:ip: proc_sys_reset:5.0	N/A
AXI SmartConnect	xilinx.com:ip: smartconnect:1.0	N/A
AXI Interconnect	xilinx.com:ip: axi_interconnect:2.1	N/A

Table 3: Microblaze and related IPs.

4.3. Averager

The Averager IP keeps a running sum for each of R, G and B. For example, the *red_sum* is the sum of the R components of every pixel in the frame seen so far. Upon the transition to the next frame the average color is updated to each sum scaled down to a maximum of 255. The average color is passed through a clock crossing FIFO so that it may be stored and read in the microblaze's clock domain. The Averager IP also outputs the average color in the pixel clock domain for use by the filtering block.

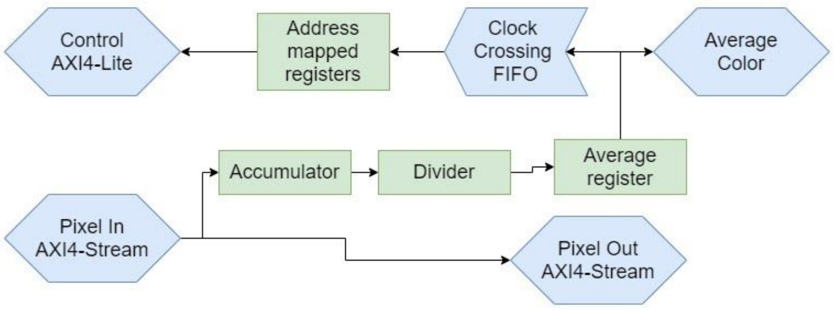

Figure 3: Block diagram of the Averager IP.

4.4. Filter

The filter IP takes in an AXI stream of pixels, applies a filter and outputs a stream of pixels. The control interface allows for a microblaze to write to a register at address *FILTER_BASE_ADDRESS*+0xC. The lower 4 bits of this register are input into a clock crossing FIFO at the AXI clock and read from the FIFO at the pixel clock rate (every cycle the FIFO is not empty). The selected filter is updated upon each new frame so as to not change filters in the middle of a frame.

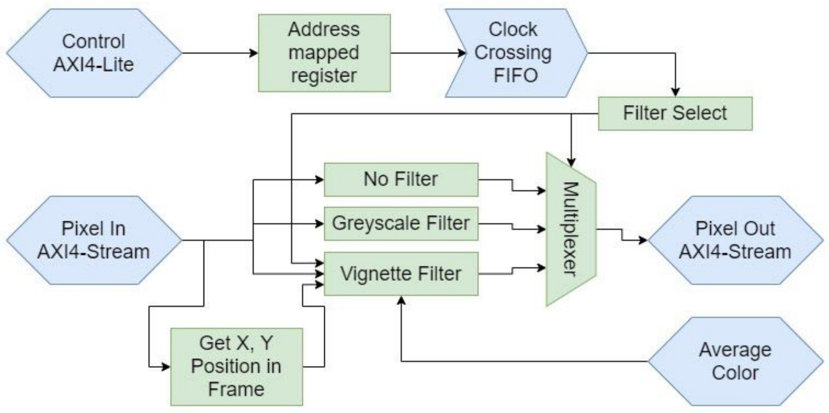

Figure 4: Block diagram of the Filter IP.

To simplify implementation, all filters take the same number of clock cycles inserting empty pipeline stages as required. All filters operate on each input pixel, and then one of their results is output.

Filter Select	**Filter**	**Variation**
0, 5, 8-15	No Filter	
1	Greyscale	
2, 3, 4, 6	Vignette	Slower dropoff (2,3), Faster dropoff (4,6) Fade to black (2,6), Fade to average color (3,4)
7	Test Pattern	

Table 4: Implemented filters.

The default filter upon startup is *No Filter* which does not modify the pixels.

The Greyscale filter operates pixel-wise. Each output pixel is a linear combination of the three input colors. Because we perceive color intensity unequally we use the following conversion.

$$Y = \frac{4899}{2^{14}}R + \frac{7471}{2^{16}}B + \frac{4809}{2^{13}}G$$

Where the input pixel is RBG and the output pixel is YYY.

The vignette filter also operates pixel-wise. It first computes R^2 as follows.

$$R^2 = \left(\frac{X-X_{mid}}{2}\right)^2 + (Y - Y_{mid})^2$$

It then uses the most significant bits of R^2 to address into a lookup table (either the slow dropoff or fast dropoff table). The lookup table returns a multiplier M from 0 to 0xFF. Each color C' of the output pixel is computed as a function of the corresponding input color C (C in RBG) and the corresponding average color (C_{avg} in $R_{avg}B_{avg}G_{avg}$ if fading to the average color - else $C_{avg} = 0$ if fading to black).

$$C' = \frac{MC}{2^8} + \frac{(255-M)C_{avg}}{2^8}$$

The creation of the dropoff tables is discussed in Appendix A.

4.5. Wi-Fi

The Wi-Fi connectivity is enabled by the Wi-Fi Pmod hardware connected to port *JA* of the Nexys Video board . Digilent provided the Wi-Fi Pmod IP core library as well as example applications using the IP drivers [2]. This IP core is connected to the MicroBlaze processor through an AXI Lite interface. WF_INTERRUPT and UART interrupt signals were connected to a concat v2.1 IP.

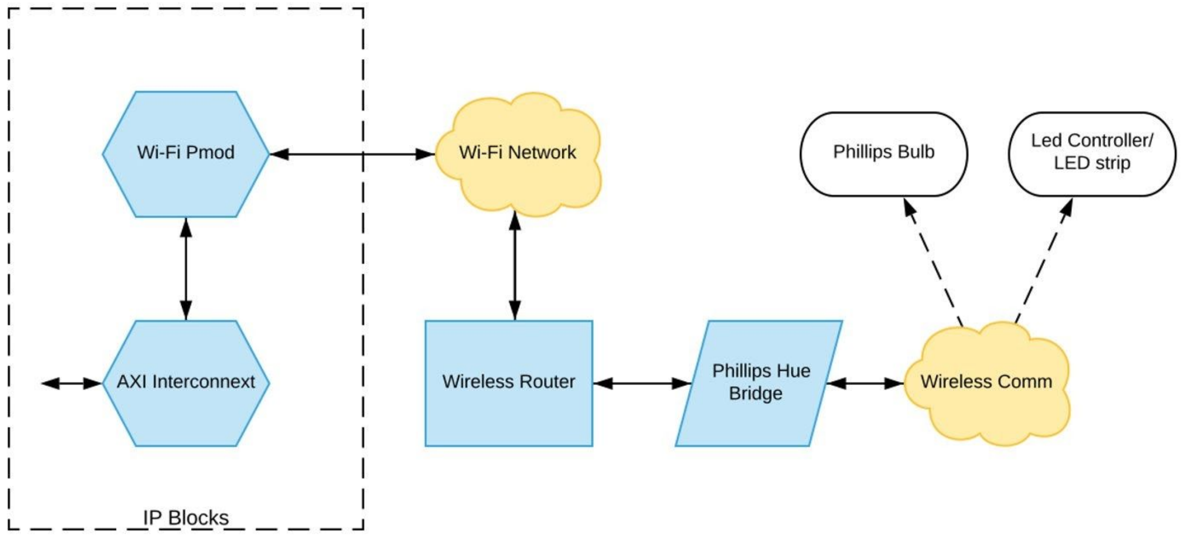

Figure 5: Block diagram of Wi-Fi Network Interface

There were occasions when MicroBlaze was able to connect to the WLAN but was not able to establish a TCP connection with the Hue bridge. To solve this issue, one important change was made on one of the files in the Pmod drivers [3]. Two lines of code were uncommented so that the Pmod Wi-Fi MAC address is properly set in the routing table.

4.6. Ethernet

We implemented Ethernet connectivity using the IPs listed in Table 6.

IP	Parameters/Changes
AXI 1G/2.5G Ethernet Subsystem	Ethernet: RGMII
AXI Direct Memory Access	N/A
AXI Interrupt Controller	Port Number: 6 (variable)

Table 6: Ethernet IP blocks

The following are other related IP blocks. Please refer to Table 3 for more information

- Memory Interface Generator
- Microblaze Debug Module
- Processor System Reset
- AXI SmartConnect
- AXI Interconnect

The Ethernet connectivity and light control works as follows. Once that the data of the average color is determined, the Ethernet IP blocks: AXI Ethernet Gigabit Subsystem and DMA read it from the Memory Interface Generator (MIG) through the Microblaze processor. Next, it connects to a python server and sends custom messages via a TCP connection. Then, the python server interprets the messages and translates the Red, Green, Blue colors' data. Finally, this data is sent as commands to an Arduino UNO which controls the RGB LED strip. The connection from the python server to the Arduino UNO is established via the Firmata communication protocol. The python server will send the last RGB data received with a delay of 50ms between commands. The general overview of this section can be observed in Figure 6.

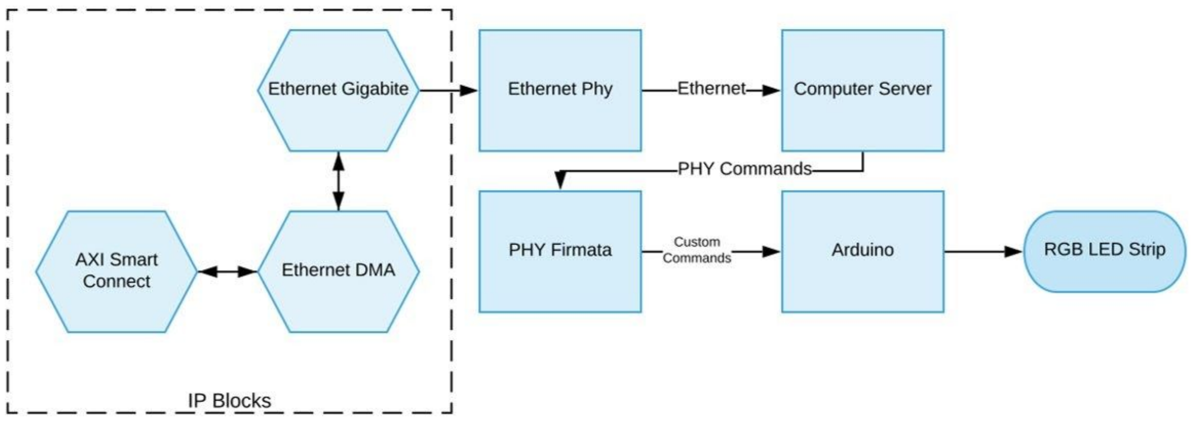

Figure 6: Block diagram of Ethernet interface.

5. Description of Your Design Tree

This section provides a summary of the project folder structure. All project files can be found in GitHub: **https://github.com/danielcampoverde/Movie-Sync**

- docs
  - documentation and resources such as the final report and presentation slides.
- src
  - MovieSync.sdk
    - SDK files to be run on the microblaze.
    - To be used with SDK 2017.4
    - Reads average color, sets which filter, programs lights using WiFi pmod.
  - MovieSync.srcs
    - To be used with Vivado 2018.1
    - constrs_1
      - constrs1.xdc - Important. Mostly same as from Digilent HDMI tutorial.
      - ila_debug.xdc - Set up ilas for debugging.
    - sources_1/bd/dsgn_1
      - Files relating to the top level block diagram of the project.
- ip_repo

    - The filtering and averaging IPs.
    - In filter_1_sub there are IPs for the FIFOs used to cross the clock domains.
- vignette
    - This is the script that was used to generate and test the vignette dropoff function.
    - There are a number of parameters to the vignette that can be varied to change the form of the dropoff.
    - The vignette_xform_*.mem file can be swapped into the filter module in filter4.sv

6. References

[1] "Digilent/Nexys-Video-HDMI," *Nexys Video HDMI*, 08-Mar-2017. [Online]. Available: https://github.com/Digilent/Nexys-Video-HDMI. [Accessed: 09-Apr-2020].

[2] "PmodWIFI V1.0," *Digilent*, 22-Aug-2019. [Online]. Available: https://github.com/Digilent/vivado-library/tree/master/ip/Pmods/PmodWIFI_v1_0. [Accessed: 09-Apr-2020].

[3] "Cannot use the pmod wifi echo client," *Digilent Forum*, 21-Feb-2020. [Online]. Available: https://forum.digilentinc.com/topic/19647-cannot-use-the-pmod-wifi-tcp-echo-client-example/. [Accessed: 09-Apr-2020].

Appendices

Appendix A

The dropoff function for the vignette was implemented using cosines and constants (as a function of *R*. It is stored in block RAM addressed by R^2. The first range of *R* from 0 to 300 is constant and passes the input unchanged. The next ranges from 300 to 338 and from 338 to 550 are both cosines. They have different amplitudes and frequencies, but are chosen so that the whole function (including transitions) is continuous and differentiable. The final section after 550 is constant.

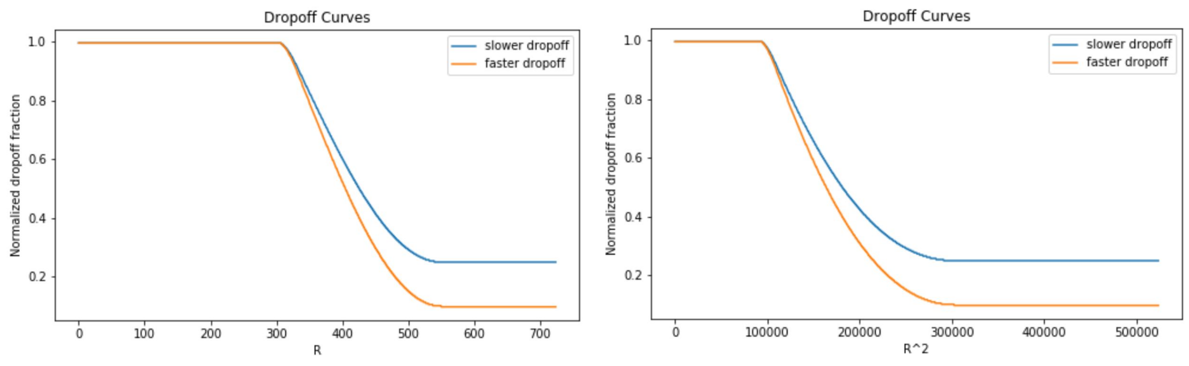

Figure 7: Dropoff functions normalized to one as a function of *R* and R^2.

Appendix B

The block design of the project is shown below. The project was created in Vivado v2018.1 and the software application was developed using Xilinx SDK v2017.4.

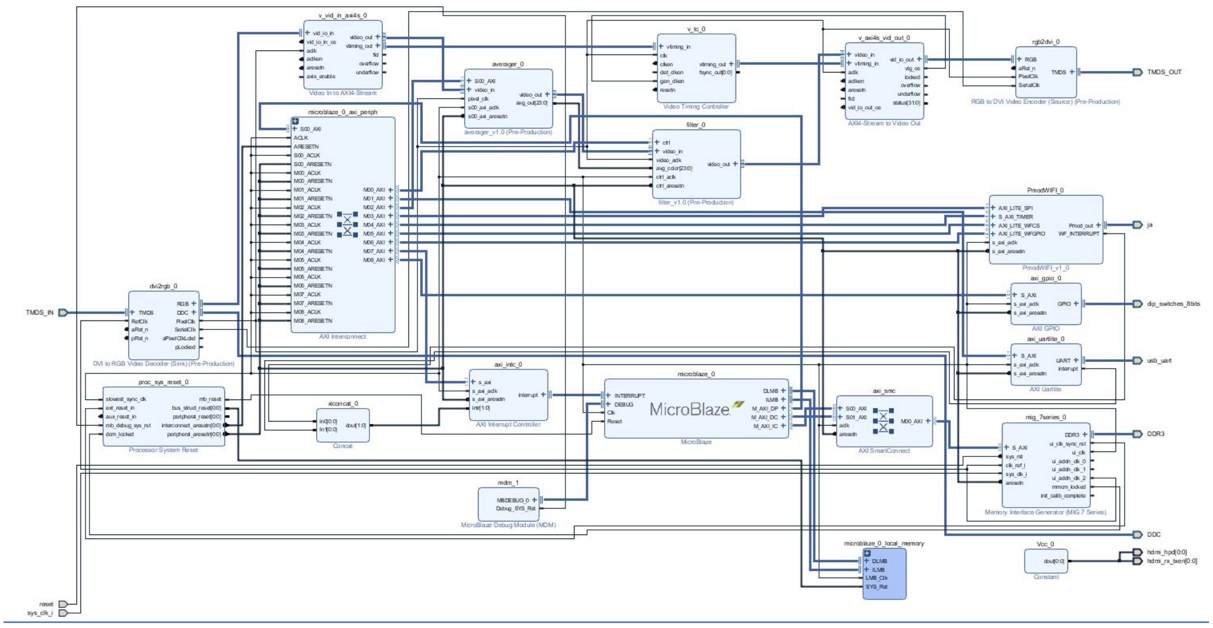

Figure 8: Block diagram of the entire design