University of Toronto

Faculty of Applied Science and Engineering

ECE532 Digital Systems Design

Final Report

Document Name	Final Report for ECE532
Project Title	Snapchat Filters
Group Number	2
Team Members	Shariq Khalil Ahmed Kazi Sudipto Arif Mingyue(Shirley) Yang
Submission Date	April 13, 2017

Table of Contents

1 Overview

1.1 Motivation

With the recent rise of social media as video streaming platforms, image processing has come to the forefront. In particular platforms like Instagram and Snapchat provide real time virtual images or filters that can superimpose an image such as a 2-D face or animal ears onto a person's face (see figure 1) [1]. This is currently mostly done by software which has its limitations compared to hardware when it comes to real time detection and drawing. In particular the low level access to the video streaming data in hardware offers potential for great speedups. With this in mind Group 2 decided it would be best to bring this concept of Snapchat filters to an FPGA and implement it in hardware.

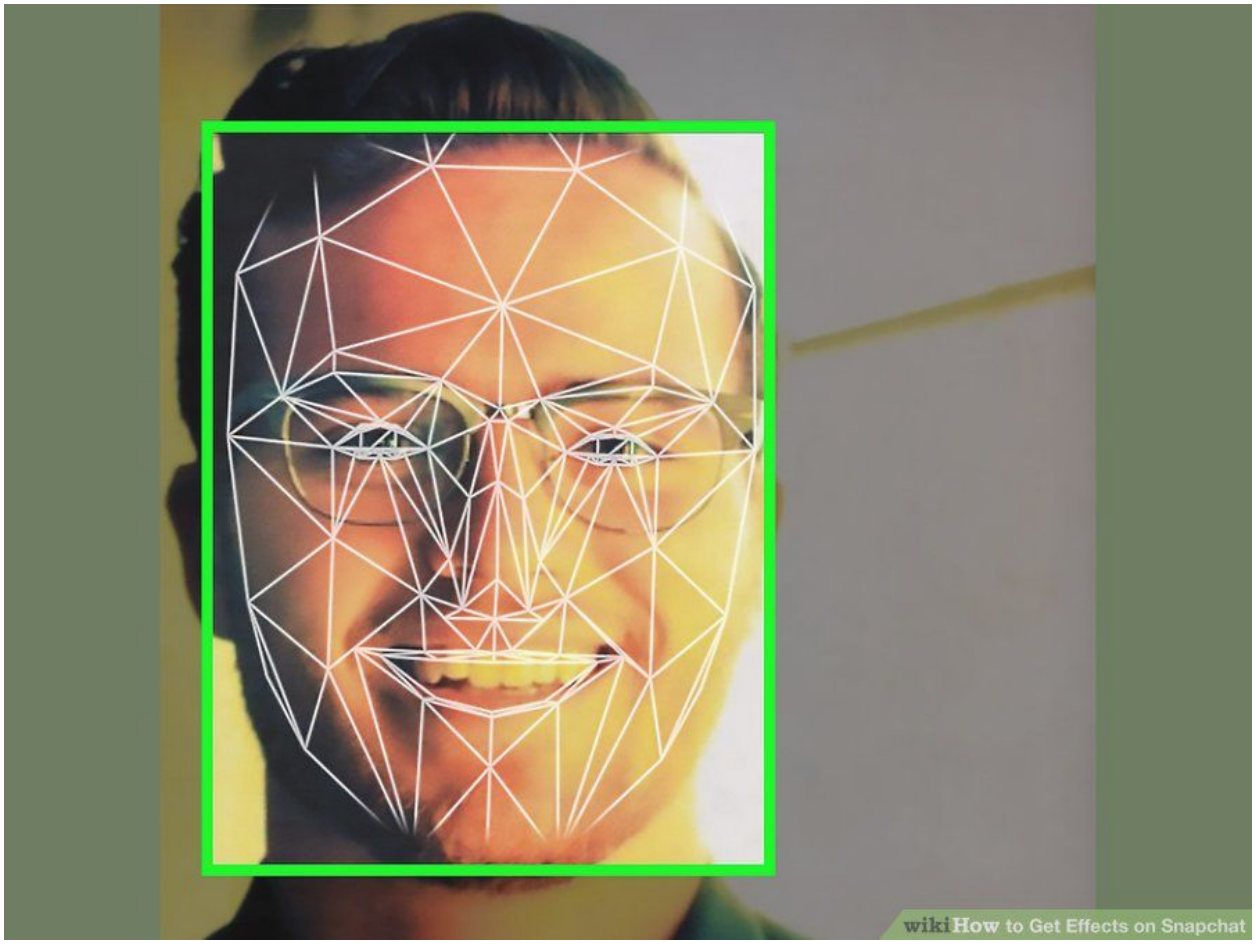

Figure 1. Example of face detection by snapchat filter [2].

1.2 Project Goals

The goal of this project is to implement some form of real time image drawing over a person's face. The intended final product should be able to detect the location of a person's face in real time from a form of video input. It will then calculate the size of overlay over the person's face and size a custom image or "filter" that would cover their face. There were two intended types of filters. The first was a simple image overlay from an arbitrary image such as a smiley face or an animal emoji. The type of overlay image would be controlled by some form of external input via bluetooth preferably a smartphone app as this removes the need to build a functioning external bluetooth input as the focus of this course is the FPGA. The bluetooth input would allow cycling through various filters and as these are cycled, the system would instantaneously change and overlay the correct image on the person's face. The second type of filter was face swap which would entail detecting two people's faces and swapping them. The intended final system will have all of these components fully integrated.

1.3 Block Diagram

Figure 2 shows the block diagram of the entire system with further details provided later in the report.

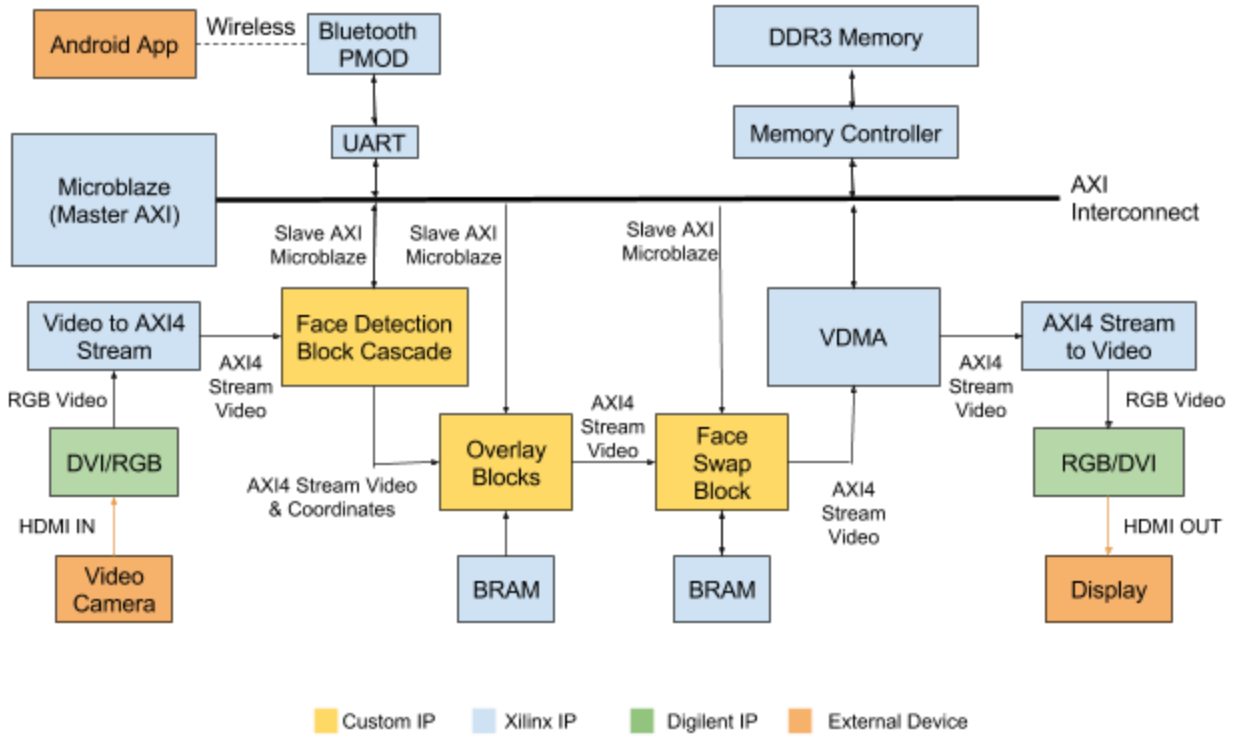

Figure 2. Block Diagram of Final Design

1.4 IP Overview

1.4.1 Processor Code and Reference Tutorial

This design utilizes Digilent's Nexys4 video HDMI demo tutorial [3]. The DVI-to-RGB block and RGB-to-DVI blocks were IPs provided by this tutorial. The C code for reading and writing to frame buffers was based upon the video demo project. Custom blocks such as face detection and overlay were integrated into this pre-existing project. Code for initialization of the custom blocks was added to the DemoInitialize() and DemoRun() functions in videodemo.c.

1.4.2 IP Blocks

This section classifies each of the IPs used in this project along with their function.

Table 1: Custom IP blocks

IP	**Function**	**Origin**
Colour Block Detection	Detect color block and infer boundaries of the face area Communicates with MicroBlaze through AXI slave interface	Custom
Overlay Image Drawing	Draw selected BRAM image on top of detected face Controlled by MicroBlaze through AXI slave interface	Custom
Face Swap	Swap faces by writing and reading video pixels to BRAM Controlled by MicroBlaze through AXI slave interface	Custom

Table 2: Pre-existing IP blocks for Video processing from Xilinx and Digilent

IP	**Function**	**Origin**
DVI to RGB Video Decoder	Decode input DVI video signal to RGB signal	Digilent [3]
Video In to AXI4-Stream	Convert RGB video signal to AXI4-Stream for other custom blocks to process	Xilinx

AXI4-Stream to Video Out	Convert AXI4-Stream to RGB signal	Xilinx
RGB to DVI Video Encoder	Encode RGB signal to DVI signal	Digilent [3]
AXI Video Direct Memory Access (VDMA)	Store and read AXI4-Stream video frame Communicates with memory interface generator through AXI bus to access memory	Xilinx
Memory Interface Generator (MIG 7)	Enable access to DDR SDRAM memory on board Used by VDMA	Xilinx
AXI GPIO	General purpose input/output for video Controlled by MicroBlaze through AXI bus interface	Xilinx
Video Timing Controller	Control timing for AXI to video out	Xilinx

Table 3: Complementary Blocks

IP	**Function**	**Origin**
UARTlite for Bluetooth PMOD	Communicates with the bluetooth PMOD Controlled by MicroBlaze through AXI slave interface	Xilinx
Block Memory Generator (BRAM)	Used by overlay and face swap blocks to store RGB pixels for overlay image and video stream	Xilinx

Table 4: Miscellaneous Xilinx blocks

IP	**Function**	**Origin**
MicroBlaze	32 bit soft processor; Control and configure various blocks in design	Xilinx
Processor System Reset	Generate reset signals for bus and peripherals	Xilinx
AXI Interconnect	Allows communication between MicroBlaze and other blocks used	Xilinx
AXI Interrupt Controller	Concentrates interrupt signal from peripherals to	Xilinx

	a single interrupt output to MicroBlaze	
AXI Timer	Provide two timers with interrupt and event capture capabilities. Can be used for software debug. Connected to AXI bus interface	Xilinx
AXI Uartlite	Enables user control through uart terminal. Connected to usb uart interface on board	Xilinx
MicroBlaze Local Memory	Containing: ***Local Memory Bus, LMB BRAM Controller, and Block Memory Generator*** Provides access to BRAM memory for the processor	Xilinx
Dynamic Clock generator	Generate clock signal to coordinate communication between MicroBlaze and peripherals on AXI bus.	Xilinx
MicroBlaze Debug Module	Support JTAG-based debug tools and the ILA for MicroBlaze	Xilinx
Integrated Logic Analyzer (ILA)	Debug signals for various blocks	Xilinx

2 Outcome

2.1 Results

In the end this project was partially working. There was great difficulty in integrating all of the separate components into one final working demo. Face detection, image overlay filter and face swap filter worked in separate parts but not in one integrated system. Bluetooth input was integrated into all of the parts. The original goal of using Viola-Jones was not met as the team determined that implementing all of it in hardware would be an undertaking that did not fit the project timeline. Instead a substitute algorithm was used that detected two blocks of color on the user's face to approximate their face. The original project specifications versus what was implemented in the end are shown in Table 5.

Table 5: Status of original project functional specifications

Original Specification	**Final Specification**	**Status**
Detects user faces using Viola-Jones algorithm	Detects user faces using 2 blocks of color on each side of their face to approximate the coordinates of the face	Partially Implemented. Facial detection in hardware is merely the sides of the user's face and software has to approximate the face dimensions from these two points.
Overlay custom image over coordinates of face	Overlays any one of 4 preset images stored in BRAM to coordinates of face	Implemented. Overlay image can be drawn over specified region in screen.
Swap faces of users	Swaps approximate faces detected by face detection	Implemented. Can swap region of two detected faces on screen in real time.
Bluetooth input able to change filters	Bluetooth input able to change filters	Implemented. Bluetooth input can cycle filters.
Integrated final system	System not fully integrated	Not Implemented. System only works in parts.

The required components for the original project were all included in the final system. Even though the overall end product was not integrated, each working piece had all of these components in them. These components are summarized in Table 6.

Table 6: Status of project component specifications

Functional Specification	**Status**
HDMI input and output	Implemented
Bluetooth input	Implemented
Overlay images stored in memory	Implemented
Custom Hardware IPs for each feature (Face Swap, Overlay, Face Detect)	Implemented

The validation criterion for the project are shown in Table 7. Most of the features of the project did not fully meet the requirements the group originally proposed. This was due to the overestimation of the capabilities of MicroBlaze as the original idea was conceived from a software point of view. It would have made more sense to research in full what was feasible for the Nexys Video board before committing to some of these criterion for success.

Table 7: Validation criterion for the project

Functional Specification	**Status**
Support a wide variety of filters	Partially Implemented. Can only support 4 image overlay filters due to BRAM memory limitations and face swap. DDR memory integration was not successful for supporting a wide variety of filters.
Accurately put filter on multiple (>2) faces	Partially Implemented. Current algorithm approximates face and overlay images are square pixel images, not face-fitting filters like actual Snapchat. Current solution is also limited to supporting only two people as only four blocks of color can be properly detected at any given time by the system and these colors are very sensitive to environmental lighting factors. This also meant that the requirements for the replacement algorithm were also not meant as only two dots were on each face compared to the planned algorithm having as many points on the face to capture as many parts of the face as possible.
Integrated final system	Not Implemented. Only discrete components of system work. Full system integration failed.
Support 1080p	Implemented. HDMI demo project used to base solution off of supports this

2.2 Reflection on Final Status of Project

The final project was ultimately not up to the standards of the initial proposal. The reason for this was due the overestimation of what could be accomplished for this project. The original proposal assumed that implementing facial detection and overlay would be straightforward enough

allowing for the project to have multiple custom IP blocks. Even the modified facial detection algorithm was non-trivial as the AXI streaming interface was not easy to fully understand which took considerable project time to get right. Furthermore the lack of a definite number of filters to support left that part of the project too open making it hard to properly validate. If this project could be attempted again it would be imperative to do better initial research about the design platform as well as base the solution with a hardware first approach as only thinking about it from a software perspective ended up obfuscating the difficulty of tasks mainly the facial detection algorithm. Also limiting the number of total features may help focus on getting a minimum viable product that is integrated and works.

2.3 Future of Project

Extensions to this project would first involve getting all the individual components integrated into a single design and working properly. The groundwork is already done to essentially add other filters as the image overlay cascade can be extended to include other filters. Another bluetooth input app can also be added to customize each of the two users using the system with separate filters as the current system makes any user detected by the system have the same filters applied on their face. Multiple bluetooth inputs would allow each user to pick their own filters. Adding DDR memory support to the project could also improve the number of filters and effects. Also eliminating the squareness of the images to make overlay look more authentic would help polish up the end product.

3 Project Schedule

The project underwent several changes over time. The summary of all the milestone reports summarizing the expectations and accomplishments and why there may be discrepancies between them for each week are located in Appendix A. The main highlights of the project are summarized in Table 8 showing pivotal points in the project.

Table 8: Highlights from each week of project

Week	**Highlight**
Week 1	● Added HDMI demo project from Digilient to serve as backbone of project. In particular relied on input and output conversion blocks as well as having most of the HDMI I/O in place.
Week 2	● After doing further research team decided to not implement Viola Jones in hardware after coming across the theory and implementing it in software. ● The team started researching a potential backup algorithm for detecting

	user faces and settled on emulating facial capture techniques by placing dots on face.
Week 3	● Successfully implemented testbench for color block detection ● Block detection not functional but testbench covers potential input test cases. ● ILA implemented for bluetooth to verify input from app was functional.
Week 4	● Ran into major trouble attempting to get basic color detection block working. ● Overlay block was delayed. ● Custom bluetooth app was created. ● Not much done due to other commitments (major deadlines in other courses). Week 5 was considered a double milestone to cover this one.
Week 5	● Block detection algorithm fixed but bugs still persisted. One thought was to add better image filtering to increase accuracy of algorithm. ● Integrated AXI Streaming interface into block detection algorithm.
Week 6	● Added image filter via Xilinx Image Enhancement IP. Results were not satisfactory as noise reduction was minimal. ● Most block detection bugs fixed ● Started working on overlay block
Week 7	● Block detection fixed ● Overlay blocks implemented for face swap and image filter overlay ● Tested and calibrated colour thresholds for color block detection IP ● Attempt to integrate everything failed

4 Block Descriptions

This section outlines a detailed description of each of the custom IPs written by the group as well as specifics about third-party IPs and their sources.

4.1 Colour Detection Block

The colour detection block is a custom block written by the team and is responsible for detecting a single block of a specific colour in the incoming video feed from the camera. In the team's design, it takes the camera feed in RGB format as input but the block can support any 24-bit RGB format video following the AXI4-Stream Video Protocol [4]. The block is not limited to any specific resolution and automatically scales to any input resolution using the end-of-line and start-of-frame signals in the AXI4-Stream Video. Since the block does not need to modify the feed, the same video stream is pushed as output. The block also has an AXI-lite slave port interface so it can be configured from a master such as MicroBlaze by writing to the registers of

the IP as listed in Table 9. The colour to be detected is specified through these registers using thresholds for the red, green and blue channel. The detected block location is outputted as four 32-bit signals specifying the minimum and maximum region of the block on the x and y axis of the video (xmin, xmax, ymin, ymax) as seen in Figure 3. These values are also exposed as read-only registers through the AXI-lite interface.

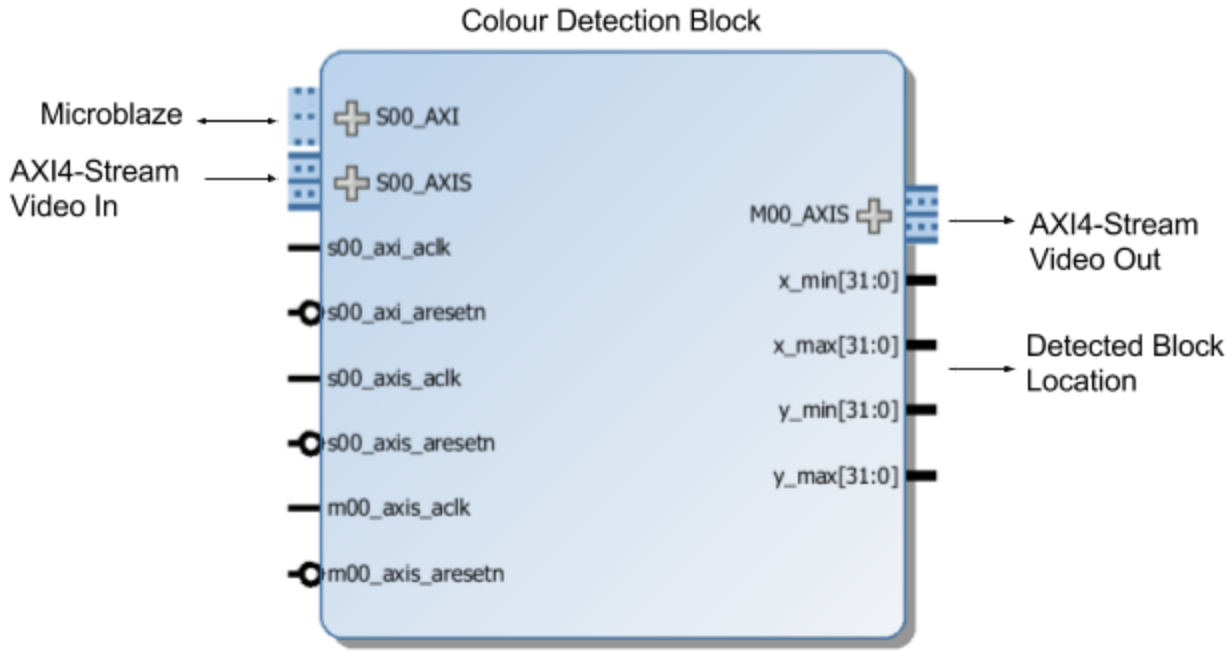

Figure 3. Colour Detection Block

Table 9. Colour Detection Block AXI-4 Slave Interface Mapped Registers

Address Offset	Register Name	Expected Type	Read/Write	Default Value
20 (Register 5)	x_min	32-bit unsigned int	Read-Only	0xffffffff
24 (Register 6)	x_max	32-bit unsigned int	Read-Only	0x0
28 (Register 7)	y_min	32-bit unsigned int	Read-Only	0xffffffff
32 (Register 8)	y_max	32-bit unsigned int	Read-Only	0x0
60 (Register 16)	r_min	8-bit unsigned int	Read/Write	0
64 (Register 17)	r_max	8-bit unsigned int	Read/Write	100
68 (Register 18)	g_min	8-bit unsigned int	Read/Write	100
72 (Register 19)	g_max	8-bit unsigned int	Read/Write	255

74 (Register 20)	b_min	8-bit unsigned int	Read/Write	0
80 (Register 21)	b_max	8-bit unsigned int	Read/Write	100

The Finite State Machine (FSM) that is modelling an edge-detection algorithm is summarized in Table 10. The input signals in the table for each state are ordered in order of precedence (i.e. the higher input state is checked before the lower one in the logic). The FSM takes input signals of start-of-frame, 1-cycle modified end-of-line and pixel_input_threshold_match to move between the states. The algorithm works by following input pixels as they are scanned in across the frame until it hits a pixel that satisfies the threshold conditions set in slave register 16-21.

The FSM shifts from NOTHING to DETECTING state and moves back to NOT_DONE or NOT_DONE_LINE states when input pixels do not match. When a line without any pixel match is reached, FSM transitions to DONE state waiting for the next frame (start-of-frame signal) to transition to NOTHING. Location of the detected colour block is tracked by keeping x (width) and y (height) counters (utilizing start-of-frame and end-of-line signals) that specify location of current pixel input on frame. x_min, x_max, y_min, y_max are continually updated in DETECTING state and their values are moved the slave register 5-8 in the DONE state. When a frame without any matching pixel is encountered, the slave register 5-8 revert to their default values specified in Table 9.

Table 10: FSM for edge detection algorithm

State	**Input Signals**	**Next State**	**State Action**
RESET	Current pixel in colour threshold `is_match = 1`	*DETECTING*	Set output boundary coordinates for the object to the coordinates detected in last frame. Reset boundary coordinates for object as screen refresh is assumed.
	Current pixel not in colour threshold `is_match = 0`	*NOTHING*	
NOTHING	Screen refresh `sof = 1`	*RESET*	This state assumes detection has yet to occur so per its name it does nothing.
	Current pixel in colour threshold `is_match = 1`	*DETECTING*	

	Current pixel not in colour threshold `is_match = 0`	*NOTHING*	
DETECTING	Screen refresh `sof = 1`	*RESET*	Updates the boundary coordinates of the detected object if necessary (i.e. if x of current pixel is smaller than x_min detected so far)
	End of current line `eol = 1`	*NOT_DONE*	
	Current pixel not in colour threshold `is_match = 0`	*DONE_LINE*	
	Current pixel in colour threshold `is_match = 1`	*DETECTING*	
DONE_LINE	Screen refresh `sof = 1`	*RESET*	State where no more detection occurs for the rest of the current line in the screen and thus nothing happens in this state.
	End of current line `eol = 1`	*NOT_DONE*	
	Any other input state	*DONE_LINE*	
NOT_DONE	Screen refresh `sof = 1`	*RESET*	This state means that object detection has started and is still waiting to see if the object ends at the current line or if it continues on the next line.
	End of current line `eol = 1`	*DONE*	
	Current pixel in colour threshold `is_match = 1`	*DETECTING*	
	Current pixel not in colour threshold `is_match = 0`	*NOT_DONE*	
DONE	Screen refresh `sof = 1`	*RESET*	State where the detected object boundaries are confirmed and output early to avoid having to wait for screen refresh to do so.
	Any other input state	*DONE*	

Due to the design of the algorithm, the colour detection block can only detect a single coloured block in the input video but this limitation did not affect this project as a series of these blocks formed a cascade with each configured to detect a different block of colour in the video.

4.2 Overlay Block (Image Drawing)

As shown in Figure 4, the overlay block takes HDMI input stream and writes output stream to the VDMA. It follows the AXI4-Stream Video Protocol using 24-bit RGB format [4] A single-port read-only block memory generator (BRAM) initialized with pre-generated COE image files are read and used to replace pixels in the overlay area of the AXI4-stream [5]. The block memory generator operates in standalone mode and has 24-bit port width.

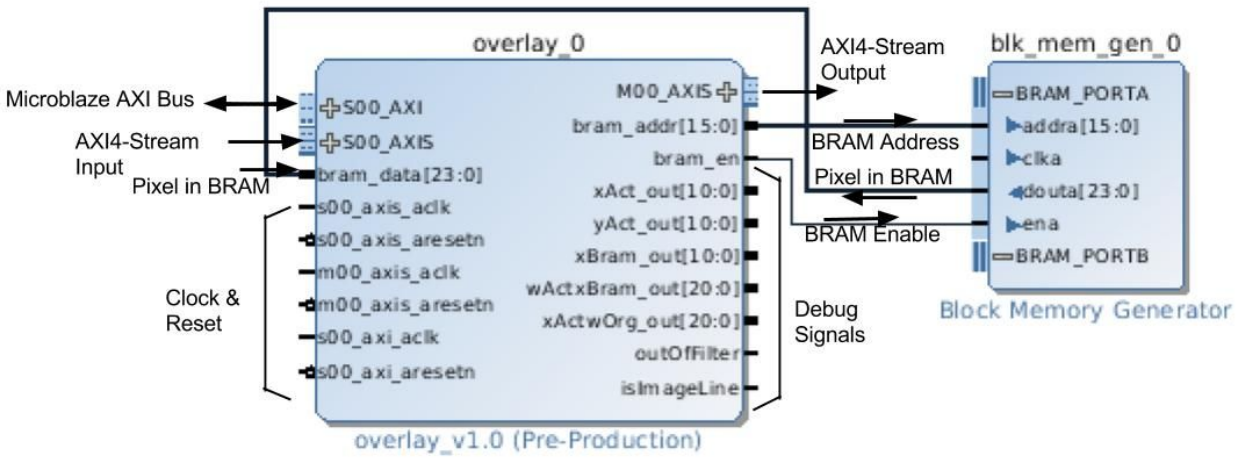

Figure 4. Overlay Block and BRAM

The overlay block contains registers(xLeft, xRight, yTop, yBott) for the boundary of the image filter. There is also a register that indicates which image in the BRAM should be displayed. These registers are writable through the AXI bus by the MicroBlaze, as shown in Table 11.

Table 11: Overlay Block AXI4 Slave Interface Mapped Registers

Address Offset	Register Name	Expected Type	Read/Write	Default Value	Usage
0 (Register 0)	filterType	32-bit unsigned int	Read-Write	0	0 - draw uniform color block; 2 - draw BRAM

					image
20 (Register 5)	xLeft	32-bit unsigned int	Read-Write	0	Specify the leftmost x boundary of the overlay area
24 (Register 6)	xRight	32-bit unsigned int	Read-Write	0	Specify the rightmost x boundary of the overlay area
28 (Register 7)	yTop	32-bit unsigned int	Read-Write	0	Specify the y value at top of the overlay area
32 (Register 8)	yBott	32-bit unsigned int	Read-Write	0	Specify the y value at bottom of the overlay area
36 (Register 9)	imgOffset	32-bit unsigned int	Read-Write	0	Starting address of overlay image in BRAM

Counters are used to keep track of the x & y position on the screen. The x-coordinate counter is incremented for every valid pixel and reset to zero when every end-of-line signal is encountered. The y-coordinate counter is incremented when an end-of-line signal is seen. Both counters are reset to zero at the start of every frame.

The overlay block scales the original image in BRAM to various sizes determined by the color detection block. The idea is that it should scale the image as the person's detected face region changes size possibly due to them moving away or towards the camera. As doing floating point calculation and division/multiplication is time consuming in hardware, accumulators using addition/subtraction are used to approximate the scaling effect. A detailed description is shown as the following.

Suppose Wactual and Woriginal represent the width of the actual image boundary determined by the block detector and width of the original image in BRAM respectively. Xactual and Xoriginal represents the x position of the pixel on the screen and in the original BRAM image respectively as Figure 5 shows. With x & y position on the actual screen found previously, our goal is to approximate Xoriginal, the x position of the pixel in BRAM.

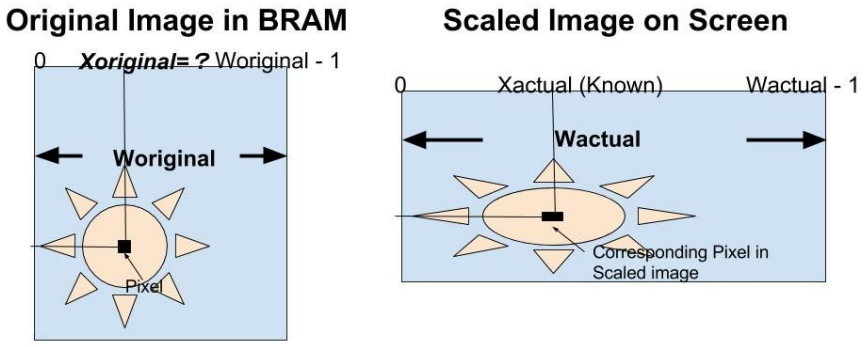

Figure 5. Pixel Correspondence in Scaled Image

The scaling formulas is shown as following:

$$\frac{Xactual}{Xoriginal} \approx \frac{Wactual}{Woriginal} \quad => \quad \text{Xactual Woriginal} \approx \text{Xoriginal Wactual}$$

As Xoriginal increments at least by 1 in every step, Xoriginal Wactual increments at least by Wactual. Therefore, error of approximation can be controlled within the range of Wactual:

$$\text{Xactual Woriginal} - \frac{Wactual}{2} < \text{Xoriginal Wactual} < \text{Xactual Woriginal} + \frac{Wactual}{2}$$

Therefore, two accumulators are used to keep track of "Xoriginal Wactual" and "Xactual Woriginal". Whenever Xactual Woriginal is out of the lower boundary, Xactual is incremented. The same applies in the y direction, while Yoriginal increments by multiples of pixels in one line of the original image every time, instead of one pixel at a time. The overlay block then adds up the approximated Xoriginal & Yoriginal position to calculate the address to use in BRAM.

4.3 Swap Block (Face Swap)

The swap block is a custom block written by the team that takes a video stream and swaps the contents of two specified blocks in the video frame. It does this by leveraging a dual-port Block RAM (BRAM) to continually store the contents of each of the blocks so they can be outputted in the output stream.

The block follows the AXI4-Stream Video Protocol [4] to input and output the video streams in a 24-bit RGB format. Instead of complicating the design with FIFO buffers for both the input and output streams, the design equates both the streams and only replaces pixel out data when the stream is scanning in the region of the specified blocks. The location of the blocks are specified

using the AXI slave port registers 0-7 by defining the two regions by x_min, x_max, y_min, y_max as seen in Table 12. These can be written to using any master such as MicroBlaze in a memory-mapped fashion. When the input stream is scanning in the defined region, the input pixels are stored by interfacing with the BRAM (port A) using the Xilinx Block Memory Generator IP [5] (found in Vivado as part of LogiCore IP library) as seen in Figure 6 (Note that the BRAM is clocked at the same clock as the video stream). Simultaneously, the pixels stored from the other blocks location (from the previous frame) are read from BRAM port B and sent out as pixel data for the output stream.

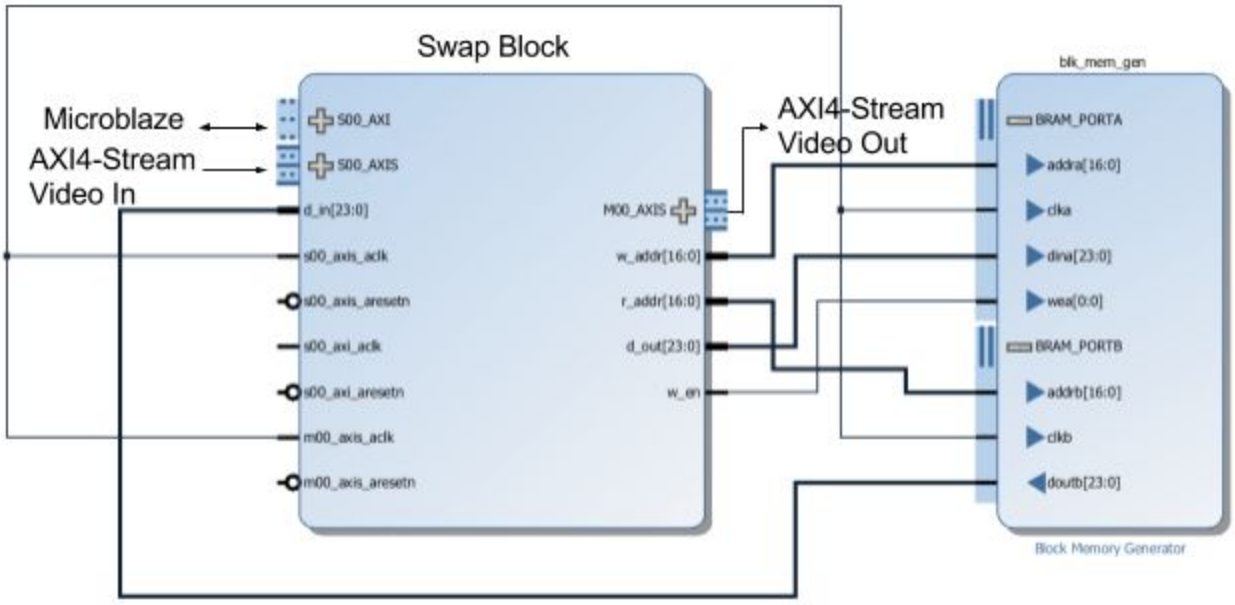

Figure 6. Swap Block and Interface with BRAM

Table 12. Swap Block AXI-4 Slave Interface Mapped Registers

Address Offset	Register Name	Expected Type	Read/Write	Default Value
0 (Register 0)	block1_x_min	32-bit unsigned int	Read/Write	0
4 (Register 1)	block1_x_max	32-bit unsigned int	Read/Write	0
8 (Register 2)	block1_y_min	32-bit unsigned int	Read/Write	0
12 (Register 3)	block1_y_max	32-bit unsigned int	Read/Write	0
16 (Register 4)	block2_x_min	32-bit unsigned int	Read/Write	0
20 (Register 5)	block2_x_max	32-bit unsigned int	Read/Write	0
24 (Register 6)	block2_y_min	32-bit unsigned int	Read/Write	0

28 (Register 7)	block2_y_max	32-bit unsigned int	Read/Write	0

Similar to the blocks detailed above, the swap block keeps track of location of current pixel in frame by using x and y counters controlled by start-of-frame and end-of-line (modified to be 1-cycle long) signals present in the AXI4-Stream Video input. This allows the design to scale to any resolution for the AXI4-Stream Video input instead of being limited to a specific resolution. However, the size of each of the two blocks specified through slave registers 0-7 is artificially limited to a maximum of 200x200 pixels after taking BRAM space limitations into consideration.

4.4 Video In/Out, VDMA and other Video IP Blocks

The video input and output are connected to HDMI ports. In order to figure out the colour for every pixel, modules are used to first convert DVI signal from HDMI In to RGB and then to AXI stream format. The AXI stream is stored into a VDMA frame buffer. For video output, the AXI stream stored in VDMA is converted to RGB and the to DVI to send as output via HDMI Out. The VDMA accesses the memory through the AXI bus. A memory interface generator (MIG 7) interacts with the VDMA to provide memory access. The MIG is connected to the DDR3 pin on the board. [3, 6] A diagram of the blocks used for video In/Out and their relations are shown in Figure 7.

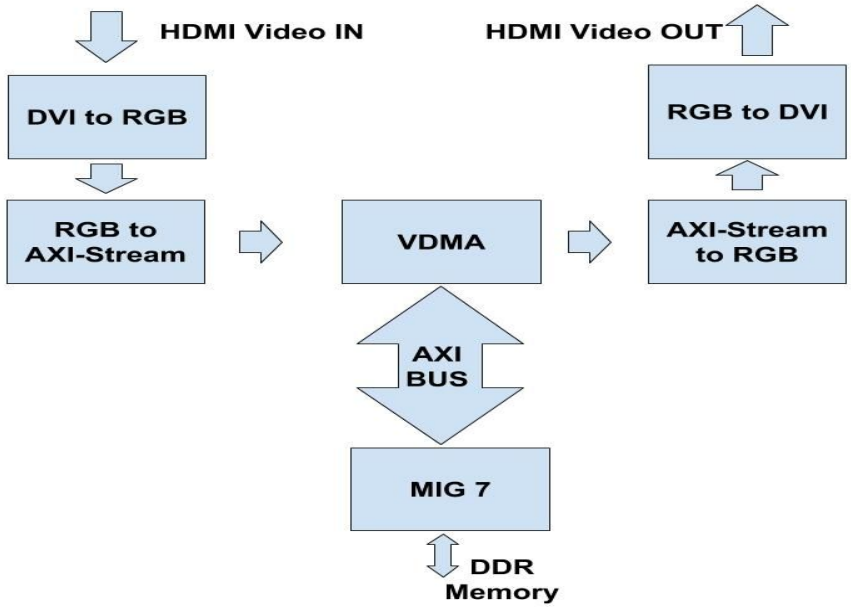

Figure 7. Blocks for HDMI Video In & Out

Since the team imported a pre-existing project with its own pipeline for the video stream, various other IP blocks were already present. These other IP blocks from Xilinx and Digilent and their configurations that make up the video input and output are listed as follows:

- ***DVI to RGB Video Decoder***: This block decodes a DVI signal to RGB format. Its input is connected from the TMDS_IN port for HDMI. This block has DDC ROM enabled. It has a TMDS clock range of <120MHz. [3]
- ***Video In to AXI4-Stream***: This block converts the RGB video in signal to AXI4 stream format so that the VDMA and custom blocks can interpret it. Its video output format is configured to RGB. Its native video input component width and AXI4 stream output component width are both set to 8. It has a FIFO depth of 32. [7]
- ***AXI4-Stream to Video Out***: This blocks converts AXI stream signals fed from the VDMA to RGB format. 1 pixel is accepted per clock cycle. Output is configured to RGB video format. AXI video input component width is set to 8. It has a FIFO buffer with a FIFO depth of 32. For timing, common clock mode and master timing mode are chosen. The hysteresis level is set to 12 and the native video output component width is set to 8. [8]
- ***RGB to DVI Video Encoder***: The block encodes RGB data format to DVI video format. It outputs signal through the HDMI port. The block has its TMDS clock range set to <120MHz. It uses an external serial clock signal from a dynamic clock generator. [3]
- ***AXI Video Direct Memory Access (VDMA)***: Xilinx LogiCore IP v6.2, This block provides memory access through the AXI bus interface. It stores and reads video frames. For video in and video out, both read channel and write channel are enabled. Both read and write have a stream data width of 24 bits, as each RGB pixel is 24 bit. Read burst size and write burst size are both configured as 32. Its GenLock mode is set to master. Software configuration is set through the AXI bus interface. [9]
- ***Memory Interface Generator (MIG 7)***: Memory interface generator enables access to memory on board. It is configured with 1 controller for DDR3 SDRAM and AXI bus interface is enabled. [10]
- ***Video Timing Controller***: This blocks controls the timing of the “AXI to video out” block. It has vertical/horizontal sync/blank generation enabled. This block also has active video generation. Detection is disabled. It includes an AXI4-Lite interface for software configuration. [11]
- ***AXI Video GPIO***: This GPIO is used for video input and output. Dual channel is used for video input and output. GPIO width is set to 1 for both channels. [12]

4.5 UARTlite for Bluetooth PMOD

Since the bluetooth stack runs on the RN-42 chip on the PMOD itself with a UART interface to the FPGA, there is no need to employ a custom IP or a complex third-party IP for establishing communication over bluetooth. The Xilinx UARTlite IP [13] was used in our design which can be found in Vivado as part of the LogiCore IP library. The RX and TX outputs of the IP are connected to the TX and RX pins of the PMOD through a PMOD header (set using .xdc constraints file) and the IP is configured to the default settings used by the BT2PMOD. These consist of:

- Baud Rate: 115200bps
- Data Bits: 8
- Parity: No Parity
- Stop Bits: 1

The MicroBlaze is connect to the IP as a master using the Slave AXI port interface. Data to and from the PMOD (and by extension bluetooth) can be exchanged using either the high-level or low-level drivers automatically generated for use on the MicroBlaze processor. The usage is identical to transferring data over USB UART and the drivers allow for multiple byte sending as well as receiving in both polling and blocking fashion.

4.6 Other Configured IPs & Tools

Other than the various IPs needed in the overall system for specific purposes, the general system IPs were:

- ***MicroBlaze***: Xilinx IP v9.6. The MicroBlaze IP initializes all IPs that require software configuration at the start of the program. It also continuously reads boundary detected by the colour detection block and writes the information to the overlay block. The MicroBlaze is configured for maximum performance. It has barrel shifter enabled, extended floating point unit support and MUL64 integer multiplier. Integer divider, additional machine status register instructions, pattern comparator, and reversed load/store and swap instructions are enabled. The MicroBlaze uses a cache size of 32KB. [14]
- ***Processor System Reset***: Xilinx Logicore IP, v5.0. This block generates reset signals for IPs and AXI bus used in this project. It uses external reset IP interface. Active high reset is used for 1 bus and 1 peripheral. Active low reset is used for 1 interconnect and 1 peripheral. [15]
- ***AXI Interconnect***: Xilinx Logicore IP, v2.1. This block acts as a bus to allow communication between the MicroBlaze and other IP blocks. Configured with 1 slave

interface for the MicroBlaze and 17 master interfaces to control blocks used in this project. [16]

- ***AXI Interrupt Controller***: Xilinx Logicore IP, v4.1. AXI interrupt controller aggregates interrupt signals from peripherals to a single interrupt signal to the processor. Uartlite and AXI Timer interrupt signals are intercepted by the controller in this project. It has registers that enable checking interrupt status through AXI interface. Default configuration is used with a clock frequency of 100 MHz and active high level interrupt. [17]
- ***AXI Timer***: Xilinx Logicore IP, v2.0. AXI timer provides two timers that can be used for interrupt and event capture during software debug. This block is configured to 32-bit mode and is connected to the AXI4 Lite interface. Trigger and generated out signals are both set to active high as default. [18]
- ***AXI Uartlite***: Xilinx Logicore IP, v2.0. AXI Uartlite allows the user to set resolution and other configurations through the terminal using MicroBlaze. This block is connected to the USB UART port board interface. It uses default configuration and has an AXI clock frequency of 100MHz, baud rate of 115200, 8 data bits and no parity. [13]
- ***MicroBlaze Debug Module***: Xilinx Logicore IP, v3.2. This module supports JTAG-based debug tools for MicroBlaze. Default configuration is used with 1 debug port and 32 bit external trace data width. [19]
- ***Integrated Logic Analyzer (ILA)***: Xilinx Logicore IP v6.0. Integrated logic analyzers are used to debug various IP blocks. Configuration is dependent on the signals included in the debug process. [20]

The tools used to assist with this project were**:**

- ***Android App for Bluetooth***: An Android app was developed to send bluetooth signals to the Bluetooth PMOD on the Nexys Video board.
- ***BMP to COE file converter***: In order to initialize BRAM with images, an online tool was used to convert BMP file to COE file format [21]. As this tool converted COE files containing 8-bit RGB format, a script was written to extend the 8-bit RGB to 24-bit RGB used in this project.

5 Design Tree

The entire Xilinx Vivado project along with associated files can be found on Github here: https://github.com/shariqkhalilahmed/G2_SnapchatFilters. The repository is comprised of the following structure in Table 13.

Table 13. Structure of Design Tree Repository

Directory	Description
docs	Documents pertaining to design and hurdles of project
src	All files related to project implementation
src/proj	Vivado project and associated files excluding IPs source but including SDK workspace (hdmi.sdk)
src/proj/hdmi.sdk	Contains videodemo Xilinx SDK project with code intended for MicroBlaze processor
src/g2_ip	Managed IP repository for custom IPs written by our team (G2) including block_detection, overlay etc.
src/repo	Contains custom IPs from Digilient including RGBtoDVI etc

6 Tips and Suggestions for Future Students

One important tip for a project of this magnitude is do sufficient research especially if the project is dealing with areas that the team might not necessarily be familiar with. It is easy to assume everything is straightforward before actual implementation as practical design considerations do not come up till implementation. Data transfer protocols also take some time to understand and time should be allotted to fully understand them before proceeding. In addition, it is more important to get a minimum viable product with all basic components integrated before developing complex features. If a project includes many components, the complexity for integration increases. The time left for integration is also taken up by the development of various features. Although building software implementations are good ways to verify the correctness of algorithms, it takes time and increases the amount of work for the project. Sometimes only thinking from a software perspective can hide the complexity needed for hardware designs.

7 Appendix

Appendix A

Table A.1: Milestone Summaries

Milestone	Status
Milestone 1	**Current Milestones** ● Acquire input from camera through HDMI IN to place in frame buffer (Shariq) ● Construct interface to allow for storage of frame buffer in memory (DDR) (Shirley) ● Output frame buffer to display through HDMI OUT (Kazi) ● The above should enable us to have continuous flow of frames from camera to display **Milestone Status** ● Analyzed and utilized Nexys 4 Video HDMI tutorial to accomplish the milestones ● Video feed from camera is captured through HDMI IN and is able to be displayed on display through HDMI OUT **Reasons for differences between goals and accomplishments** ● The HDMI demo setup most of the hardware for implementing HDMI IN to HDMI OUT and the team had to simply understand how the software worked to demo it making this milestone straightforward. **Project Modification** ● The current facial recognition algorithm (Viola-Jones) may be replaced if the team finds that another algorithm might be more suitable after development starts.
Milestone 2	**Current Milestones** ● Implement software version of Viola-Jones algorithm (Shirley) ● Researched alternative facial detection methods that satisfied goal of real-time image processing (Kazi & Shariq)

	Milestone Status ● Viola-Jones algorithm working in software ● Team came up with other option of using markers on face to detect face **Reasons for differences between goals and accomplishments** ● Team had trouble first understanding the theory behind Viola-Jones. Then the team realized that third-party libraries like OpenCV would not work on MicroBlaze as well as the lack of storage options for a training library. These considerations led to this milestone not being fully met as it was ultimately changed as an alternative was sought instead. **Project Modification** ● We originally scheduled to have the hardware version of the facial detection block working for Milestone 4. However due to various group member's schedules along with changing the facial detection method, this does not seem feasible. Most likely we would have this milestone ready for Milestone 5. ● We will probably not continue with Viola-Jones and opt for an easier face detection method which will be discussed with the TA in lab.
Milestone 3	**Current Milestones** ● Implement bluetooth block and use debugging code to test it (Shariq) ● Implement image detection and overlay in software (Shirley) ● Implement hardware image detection and create testbench for it (Kazi) **Milestone Status** ● Bluetooth block works with testing verified in ILA ● Software version of image detection and overlay work ● Image detection in hardware has been implemented and tested using testbench **Reasons for differences between goals and accomplishments** ● Team had a working testbench and ILA demo in time for the testbench demo. These were more or less what was promised.

	Project Modification ● Not using Viola-Jones anymore for image detection.
Milestone 4	**Current Milestones** ● Implement color block detection in streaming hardware(Kazi) ● Development of Android Application customized towards our project (snapchat filter selection) (Shariq) ● Aid team member (Kazi) in completing his goal of achieving hardware implementation of basic multiple-colored block detection (Shariq) ● Started working on overlay block (Shirley) **Milestone Status** ● Had to consider adding edge detection which changed the overall goal for this week. Therefore this milestone has been shifted by a week and added to the mid project demo. ● Completed development of a customized android application that has bluetooth device selection, buttons-based UI to change filters and text-based input to send other messages to the FPGA powered system ● Team members unable to complete hardware implementation of block detection not completed due to unforeseen changes to detection method (currently trying edge detection) and busy schedule for week but this was already expected as noted in his last milestone report. Expected to complete next week as predicted in last milestone report. **Reasons for differences between goals and accomplishments** ● Team had several major deadlines in other courses that week (i.e. multiple projects and midterms per each team member). No real work was done that week as a result. **Project Modification** ● No major modifications have been made since last week but slight tweaks have been to the block detection algorithm.
Milestone 5	**Current Milestones** ● Integrate bluetooth input from Android app with streaming HDMI project to control drawing (Shariq) ● Detect block of colour in hardware (Kazi) ● Integrated AXI Stream interface with our custom IP (Kazi & Shariq)

	● Created testbench for colour detection FSM in detection block and verified it via simulations (Kazi) ● Created custom block for drawing overlay image (Shirley) **Milestone Status** ● Currently able to detect block of colour and able to feed data from Android app to change central point of colour block to different colour. Still debugging the overlay image block. **Reasons for differences between goals and accomplishments** ● Android app was completed meeting the goal for the bluetooth block in the overall design. However considerable difficulty in implementing and understanding the AXI streaming interface for video purposes lead to the delay of the overlay block as it was not working in time. The colour detection was working but had some bugs. **Project Modification** ● None to report at this point in time.
Milestone 6	**Current Milestones** ● Implement detection of a single object of uniform colour (Shariq) ● Implement filtering of pixels to reduce noise (Kazi) ● Added BRAM to overlay block. Fixing bugs. (Shirley) **Milestone Status** ● Block detection was determined to be buggy and is still being debugged. Hopefully this can be corrected and extended in time for the final project demo. ● Noise filtering was added to the HDMI input but the effects seemed minimal as the Xilinx IP does not seem to really do much to improve image quality. **Reasons for differences between goals and accomplishments** ● Team ended up wasting a lot of time doing futile efforts such as adding and image filter and this lead to the colour block detection's completion being delayed.

	Project Modification ● Due to time constraints it is looking more and more likely that complex image drawing will not be possible. The team aims to have some form of overlay done.
Milestone 7	**Current Milestones** ● Implement working face swap block (Shariq) ● Implement and calibrate working colour detection block (Kazi and Shariq) ● Complete working overlay block (Shirley) ● Integrate entire system to include all working blocks once they are finished (Kazi) **Milestone Status** ● Overlay, face swap and block detection are all functional but only when run separately ● Overall system integration failed as multiple bugs existed by joining everything together **Reasons for differences between goals and accomplishments** ● Each of the custom IP blocks worked as planned more or less. The issue was that by the time the team got around to putting it all together there were a lot of new bugs introduced and simply not enough time to fix all of them for the demo. The team instead decided to demo each individual block to show that the project works in components. **Project Modification** ● None to report.

8 References

[1] “Surrey Social Media,” *Surrey Social Media*. [Online]. Available: http://blogs.surrey.ac.uk/socialmedia/new-features-galore-from-snapchat-and-its-copycats-november-social-media-roundup/. [Accessed: 12-Apr-2017].

[2] "How to Get Effects on Snapchat", *wikiHow*, 2017. [Online]. Available: http://www.wikihow.com/Get-Effects-on-Snapchat. [Accessed: 12-Apr- 2017].

[3] “Nexys Video HDMI Demo,” *Nexys Video HDMI Demo [Reference.Digilentinc]*. [Online]. Available: https://reference.digilentinc.com/learn/programmable-logic/tutorials/nexys-video-hdmi-demo/start. [Accessed: 12-Apr-2017].

[4] “AXI Reference Guide”. pg 96. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_ref_guide/latest/ug1037-vivado-axi-reference-guide.pdf [Accessed: 12-Apr-2017].

[5] “Block Memory Generator Datasheet”. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/blk_mem_gen/v8_3/pg058-blk-mem-gen.pdf. [Accessed: 12-Apr-2017].

[6] “AXI-4 Stream Video IP and System Design Guide”. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_videoip/v1_0/ug934_axi_videoIP.pdf. [Accessed: 12-Apr-2017].

[7] “Video In to AXI-4 Stream v4.0”. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/v_vid_in_axi4s/v4_0/pg043_v_vid_in_axi4s.pdf. [Accessed: 12-Apr-2017].

[8] “AXI-4 Stream to Video Out”. [Online]. Available: https://www.xilinx.com/products/intellectual-property/axi4_stream_to_video_out.html. [Accessed: 12-Apr-2017].

[9] “AXI Video Direct Memory Access v6.2”. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_vdma/v6_2/pg020_axi_vdma.pdf. [Accessed: 12-Apr-2017].

[10] “Memory Interface”. [Online]. Available: https://www.xilinx.com/products/intellectual-property/mig.html. [Accessed: 12-Apr-2017].

[11] “Video Timing Controller”. [Online]. Available: https://www.xilinx.com/products/intellectual-property/ef-di-vid-timing.html. [Accessed: 12-Apr-2017].

[12] “AXI GPIO v2.0”. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_gpio/v2_0/pg144-axi-gpio.pdf. [Accessed: 12-Apr-2017].

[13] “AXI UART Lite v2.0”. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_uartlite/v2_0/pg142-axi-uartlite.pdf. [Accessed: 12-Apr-2017].

[14] “MicroBlaze Soft Processor Core”. [Online]. Available: https://www.xilinx.com/products/design-tools/MicroBlaze.html. [Accessed: 12-Apr-2017].

[15] “Processor System Reset Module v5.0”. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/proc_sys_reset/v5_0/pg164-proc-sys-reset.pdf. [Accessed: 12-Apr-2017].

[16] “AXI Interconnect v2.1”. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_interconnect/v2_1/pg059-axi-interconnect.pdf. [Accessed: 12-Apr-2017].

[17] “AXI Interrupt Controller (INTC) v4.1”. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_intc/v4_1/pg099-axi-intc.pdf. [Accessed: 12-Apr-2017].

[18] “AXI Timer v2.0”. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/axi_timer/v2_0/pg079-axi-timer.pdf. [Accessed: 12-Apr-2017].

[19] “MicroBlaze Debug Module (MDM) v3.2”. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/mdm/v3_2/pg115-mdm.pdf. [Accessed: 12-Apr-2017].

[20] “Integrated Logic Analyzer v6.0”. [Online]. Available: https://www.xilinx.com/support/documentation/ip_documentation/ila/v6_1/pg172-ila.pdf. [Accessed: 12-Apr-2017].

[21] “coetool,” *coetool .coe file to image and vice versa*. [Online]. Available: http://jqm.io/files/coetool/. [Accessed: 12-Apr-2017].