Final Report
Group 4

Nafis Ahbab
Haiqi Xu
Mariko Tatsumi

IoT Air Draw

Table of Contents

1. Overview

This section gives an introduction into our design and what we as a team hoped to accomplish through this project

1.1 Motivation

From the very start of the project, we were challenged to come up with a project that composed the theme of IoT (Internet of Things) and utilized the strengths of the FPGA to perform tasks that would be very slow in a embedded processor. We also wanted our project to be interactive to have that “wow” factor whenever a user used it.

Given all of this features in mind, we decided to go with IoT Airdraw: a project that would allow a user to draw on a 2D graphical interface, like Paint, by using a controller that they can move in 3D space. To accomplish the IoT theme, our project would use a wireless controller and the FPGA would be connected via ethernet to a server that would handle the drawing. The user could see their drawing in real time and would get a sense of amazement and ownership as their painting gets displayed on the screen.

1.2 Project Description

As discussed in the motivation section, the project aims to provide the ability for user to draw on 2D interface using the controller in 3D space with the help of FPGA. To accomplish this, design contains a controller with LED that the user can move, and a video camera that will capture the movement of the controller and feed video data into FPGA. The FPGA will be responsible for detecting the position of the LED in the video frame. The embedded processor will read the coordinates from the IP block and send that, via ethernet, to the server which will have a drawing application running. The server will use the coordinates and other button commands from the controller to draw on the screen.

1.3 Goals

To build the project described above, goals that were set out for each component of the project are as follows:

Custom IP:

- Multi color detection
- Fast processing speed at 30HZ on HD video
- Simple algorithm with high detection accuracy
- Capability of detecting size of LED (using bounding box)

Embedded Processor:

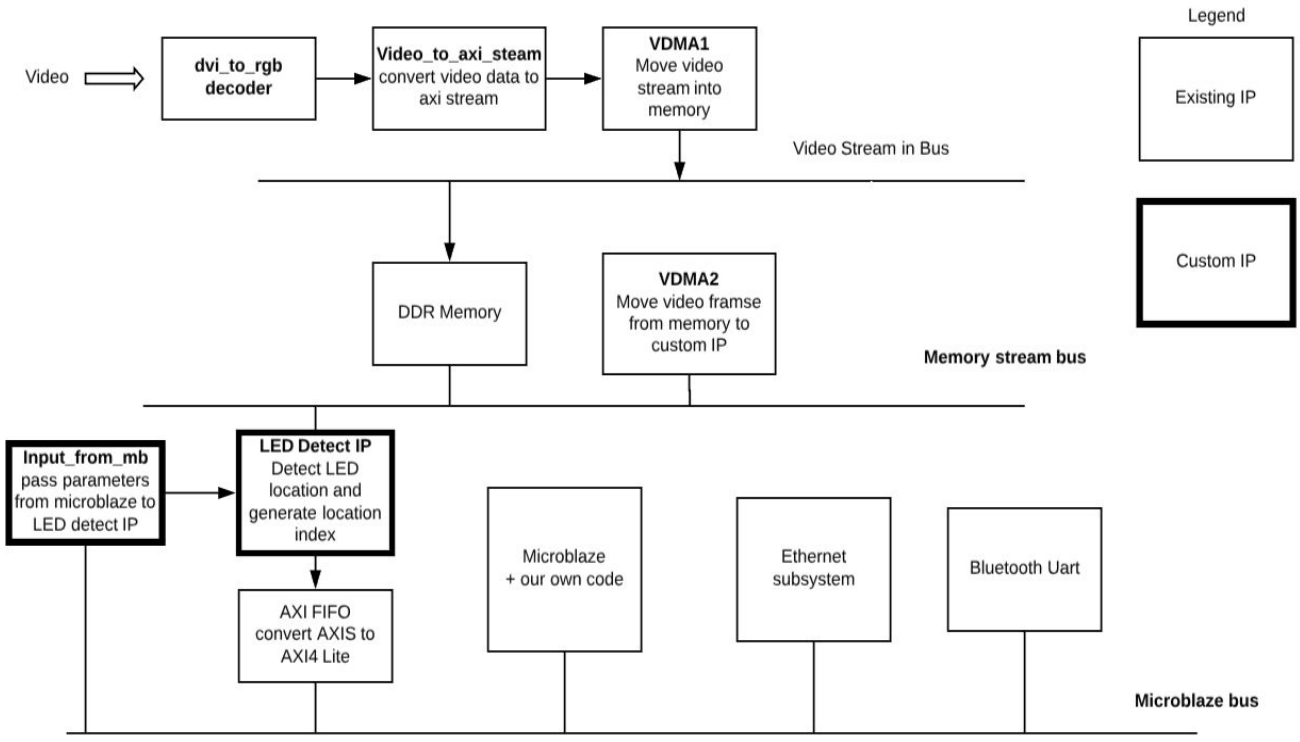
- Ability to read and send LED location via ethernet
- Ability to communicate via Bluetooth to Wireless Controller

Controller:

- Wireless and easy to use
- Ability to change LED color and relay LED color to processor
- Ability to communicate via Bluetooth to Embedded Processor
- Several buttons that allows user to perform different functions based on button inputs

1.4 Block Diagram

The block design diagram contains all important IPs used in the design.



1.5 brief description of IP

1.5.1 Custom IP

The Final design contains two custom IPs but both of them serve the same purpose, LED location detection.

DET_COLOR_CORD (LED detect IP)
The DET_COLOR_CORD IP will take video frames as input, detect LED location in the frame and generate one X Y coordinate for each input frame.
INPUT_FROM_MB
This IP is used to pass arguments from Microblaze to DET_COLOR_CORD IP.

1.5.2 Existing IPs

The Final design leveraged many different IPs from Xilinx and Digilent libraries. This part will discuss a few import ones. Common IPs such as clock wizard and MDM will be omitted for simplicity.

VDMA
The VDMA IPs are used in our design to move video stream data in and out from Memory. There are two VDMAs used in our project. One is used to move video stream into memory and the other is used to move frames from memory to our custom IP

AXI FIFO
Because our custom IP uses Axi-stream as output which is not compatible with microblaze, we used AXI FIFO to convert axi-stream data into axi4-lite data.

BLUETOOTH
Using UART with Baud rate of 38400, Data Bits: 8 and no Parity and setting the TX and RX pins to the appropriate PMOD pins.

AXI 1G/2.5G Ethernet Subsystem + DMA
The ethernet subsystem is used to establish the communication path between the FPGA and software frontend.

MIG(Memory interface generator)
Provides AXI-4 DDR memory access for microblaze and VDMA.

Microblaze
Microblaze is used as the main logic handler in our design. It handles the communication with external controller and frontend, and controls VDMA and our custom IP base on user commands.

AXI Interrupt controller
Provide interrupt hardware interface for microblaze. All peripheral interrupts will be handled by the interrupt controller before processed by microblaze

AXI Interconnect

Two AXI interconnects are used in our design. One is used to connect Microblaze to multiple AXI4 slaves. The other one connects multiple AXI masters to memory interface

DVI2RGB
The IP is provided by Digilent. It converts input DVI data to RGB video data.

Video into AXI4-Stream
Converts RGB video input data into 24 bits AXI Stream which can be used by custom IP and VDMA

AXI GPIO
Directly connects to one HDMI pin. Used to detect if camera input is connected to FPGA.

AXI Timer
Provides timer functionality for LWIP

ILA
Provide direct internal signal runtime access for our debugging purpose.

AXI Uartlite
Used to send printed data to SDK console

Referenced design
For hardware block design, we referred to Digilent video board's design[1] for video path setup and Ethernet tutorial[2] for Ethernet subsystem setup. Our own work includes integrating the two hardware designs and customizing the SDK code. For VDMA setup in SDK, we also referred to one past project[3] and leveraged their VDMA functions wrapper files vdma.c and vdma.h[4].

2. Outcome

2.1 Result

Our final design is successful in terms of meeting our goals. It meets all of our goals except ability to accurately detect LED's distance from camera.

Our custom IP is able to detect LED in the video frames at our required speed 30Hz. The custom IP uses a simple greedy algorithm in the custom IP for LED detection to avoid over complex computation on video frames to ensure the high processing speed while ensuring a relative accurate detection of LED location. With a proper distance and background, the bias of LED's location is within 10 pixels out of 720. By adjusting the RGB threshold used in the custom IP, we can detect different colors of LED as well. We weren't able to detect distance between

LED and FPGA because the size of LED appearing in video heavily depends on angle and color of LED, which can not be used to approximate the distance directly.
The IoT aspect of the design also achieves all our requirements. The design uses ethernet and bluetooth module to effectively communicates with external software and arduino and not causing any noticeable delay to user.

The Embedded processor can successfully read the LED coordinates from the customIP and send/receive data via Bluetooth from the controller. It can forward that information via ethernet to the server.

The server can run a graphical program programmed using PyOpengl to draw on the screen based on information provided by the client (i.e - Embedded Processor). It also performs smoothing functions to smooth out any jitter during the drawing process.

The controller was designed as shown below -

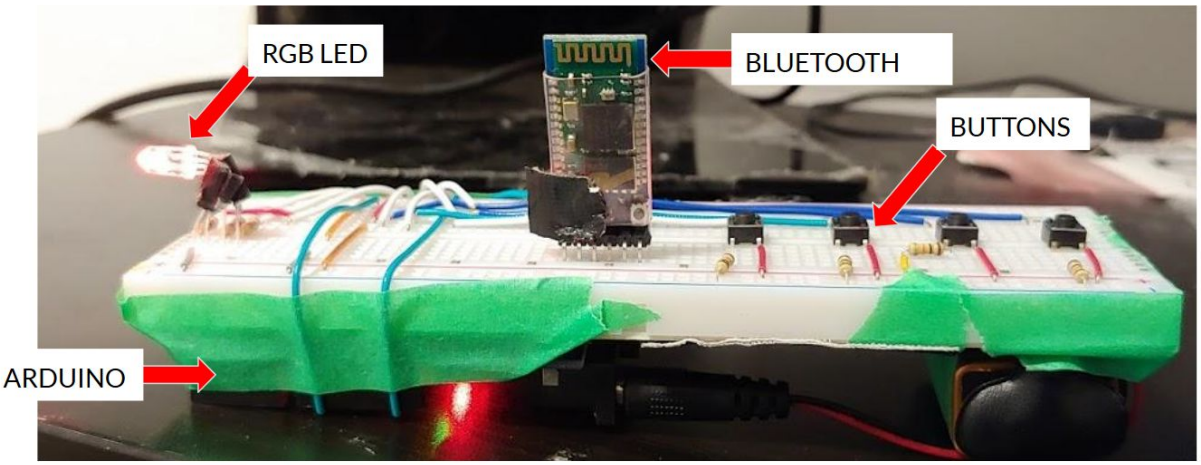


Figure 1. Controller

An Arduino was used to control all of the components - RGB LED, Bluetooth module, 4 push-buttons. Push buttons were used to change the color of the LED, start/stop drawing, use eraser or reset the drawing.

2.2 Change in project

Originally, we planned to use interrupt for our custom IP to signal microblaze whenever a coordinate is available. However, at the mid of the project, we found that interrupt is not necessary since polling mode also achieves required processing speed. So instead of implementing interrupt, we chose to add a bluetooth module, which actually improved the usability of our final design.

2.3 Possible improvement

To achieve high LED detection accuracy, our design must be used with a proper background. If the background contain anything that is too bright, the detection on LED could be inaccurate. In addition, we also notice that color in the background could interfere with the detection on LED color. One possible improvement on this issue is to add a functionality to allow custom IP to correlate with background to adjust threshold used in LED detection to avoid interference from background.
In our final design, we also utilize libraries in software to do smoothing and bias correction on collected LED coordinates. We chose to the feature in software because it is beyond our original scope and the remaining time in the project does not allow us to implement the feature in hardware. For people who want to keep working on our project, one improvement can be implementing a hardware IP to replace the software library for smoothing and bias correction. Since the video processing is already done by our custom IP, the work on the new IP can be independent from video processing and only use generated LED coordinates.

3. Project Schedule

Table: Milestones

Milestone	Proposed	Delivered	Discussion
#1	Get Pmod camera working on the FPGA and generate stream of RGB data input from video	Studied how to integrate PMOD camera with MicroBlaze and built a prototype of color detection algorithm in software.	Milestone is completed as planned. After trying PMOD camera during the lab, we decided to switch HDMI camera since the original camera's resolution is too low. Besides that, board was switched to video board as well.
#2	Use VDMA to store video Data into the DDR2 memory. Setup necessary debugging environment such as uart and ILA	Managed to get the video stream from the camera into the board. Ethernet connection was established with video board. Modified the detection algorithm of the prototype a little bit.	Milestone is completed as expected. Since the tutorial for ethernet module given by digilent[2] was not updated for 2018.2, some debugging were needed to make it work. Debugging environment is setup on the process as well.

#3	Implement prototype of our custom IP which processes a fixed rgb frame with LED and send the coordinate of LED pixels to microblaze	Integrated HDMI IPs and ethernet modules. Improved prototype of detection algorithm. Built a first prototype of custom IP module in hardware.	Ahead of original schedule. Although original milestone did not mention about HDMI and ethernet IPs for this week, we finished integration of them. Given that the integration was the task for milestone #7, it can be said that we were ahead of schedule.
#4	Connect the custom IP block and video system to process video frames at 30 fps and configure Microblaze interrupt.	Built a testbench for custom IP. Tested and confirmed that our hardware custom IP works as expected with static image (eg. PNG).	Behind a little bit of schedule. The functionality of custom IP was confirmed with static input but was not tested with streaming input from HDMI camera. Given its frequency, we decided not to use interrupt from custom IP.
#5	**Mid Project Demo:** Setup the Ethernet connection between FPGA and computer. Implement prototype software to receive LED coordinates from FPGA at 30 fps.	Built a mobile painting controller with Arduino. Built a graphic interface in computer. Integrated hardware custom IP with HDMI IPs and ethernet module.	Milestone is completed as planned and we demonstrated the working prototype. Although it is not mentioned in the original milestone, a controller was built to be used as a ‘brush’. Bluetooth module was used for communication with FPGA board.
#6	Improve software prototype to display LED coordinates as pixel on a white canvas	Fixed some bugs on custom IP.	Ahead of schedule. The functionalities for drawing lines and showing the current controller position as a point were implemented at the time of Mid Project demo.
#7	**Final Demo:** Integration of all the parts and prepare for final demo.	Added new features such as display scrolling on graphic interface and adjusting thickness of drawing line.	Ahead of schedule. Since the first milestones and requirements had already been met, we have added some other features on graphic interface.

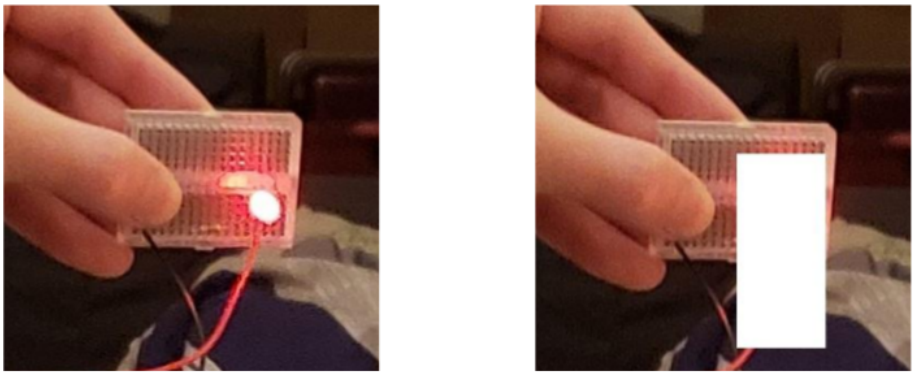
4. Description of the Blocks

4.1 Custom IP Block

The custom IP block consist of two custom IPs: LED_DET and INPUT_FROM_MB.

4.1.1 DET_COLOR_CORD

This IP takes video axi-stream as input and generates the coordinates of center and bounding box of the LED as axi-stream output. It also has several wire input ports which is used to accept threshold values and color code from microblaze because we didn't implement AXI-Lite interface on the IP. The implementation of the IP is done with HLS.
The DET_COLOR_CORD IP calculates the coordinates of the center and bounding box of each frame[Figure1] by processing every pixel one by one.



(a) Original Picture with red LED

(b) Same Picture with bounding box of the LED

Figure 2. Input image with bounding box of the LED

The DET_COLOR_CORD IP investigates each frame by repeating following several steps for every pixel in the frame: First, it checks the target color, which is color of the LED the IP tries to detect chosen from red, green and blue. Then it checks whether the current pixel is a part of the LED in this frame or not. If the RGB value of the pixel satisfy the threshold values passed by INPUT_FROM_MB, the pixel is recognized as a part of LED in the frame and the coordinates for bounding box of the LED is updated to take this pixel into consideration. If it does not meet the condition, the process on the current pixel is completed. After repeating the whole process for every pixel, the DET_COLOR_CORD IP calculate the center coordinate of the LED by computing the middle points of the bounding box and outputs it. If no LED is detected in the frame, DET_COLOR_CORD IP outputs invalid values (0xFFFF) for both of the coordinates of center and bounding box of the LED. The whole flowchart is shown below.

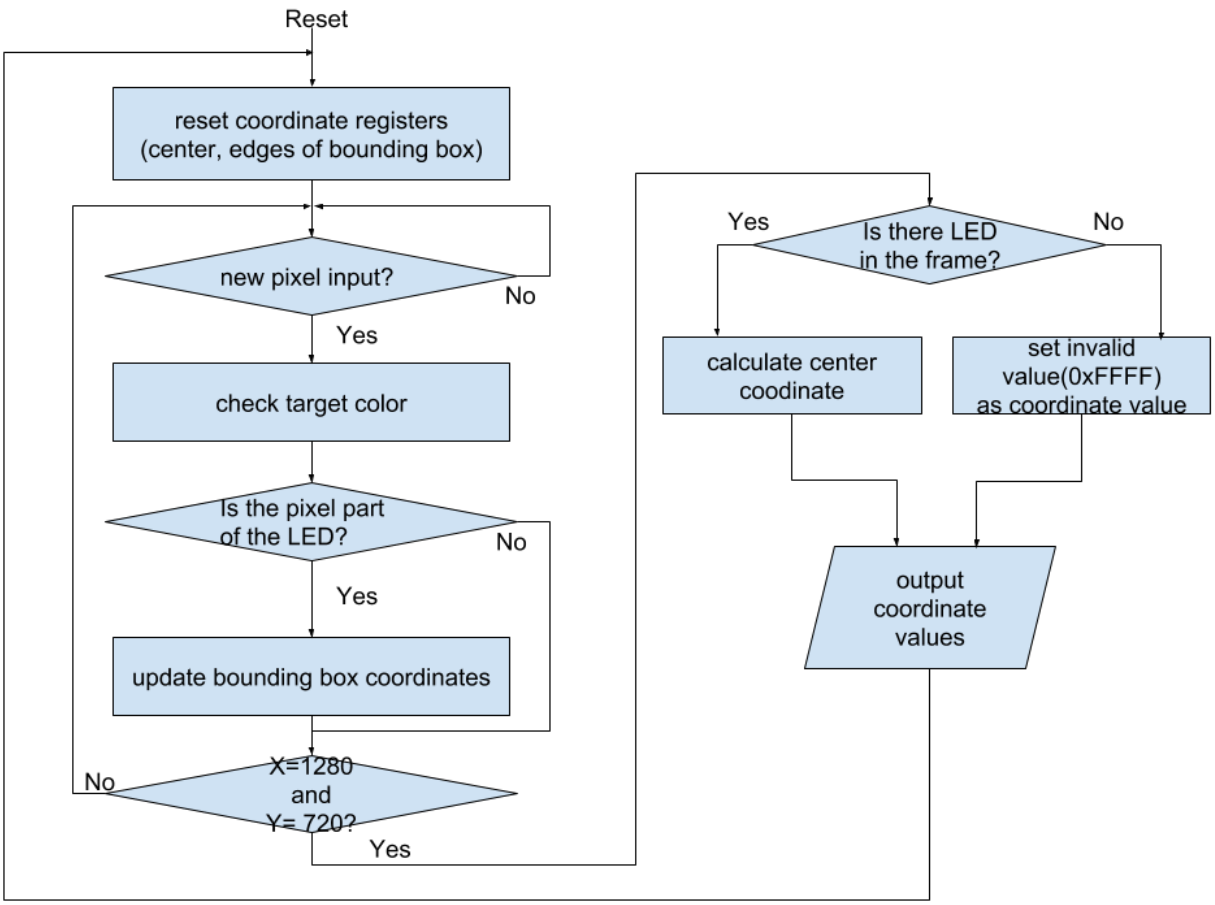


Figure 3. Flow chart of the LED_DET IP

4.1.2 INPUT_FROM_MB

The INPUT_FROM_MB IP provides a write-only AXI-Lite interface for microblaze and outputs the data written by microblaze as wire output to pass into DET_COLOR_CORD IP.
The following table shows the register of INPUT_FROM_MB module and their address.

Address Offset	Register name	Access Type	Description
0x0000	slv_reg0	Read	RGB threshold value for target color
0x0004	slv_reg1	Read	RGB threshold value for other colors
0x0008	slv_reg2	Read	Target color (red, green or blue)
0x000C	slv_reg3	Read	Width of the frame
0x0014	slv_reg4	Read	Height of the frame

0x0018	slv_reg5	Read	Custom reset

Table1. Register assignments

Target Threshold (slv_reg0) and Other Threshold (slv_reg1)
First 8bits of each registers are used as a threshold value for target color and other color. Although it passes value with width of 32bits, only The pixel is identified as a part of LED if and only if both following conditions are satisfied: 1. The value of target channel (R, G or B) is larger than target threshold. 2. The value of both of two channels are less than other threshold. In our design, target threshold is set to 240 and other threshold is set to 170.

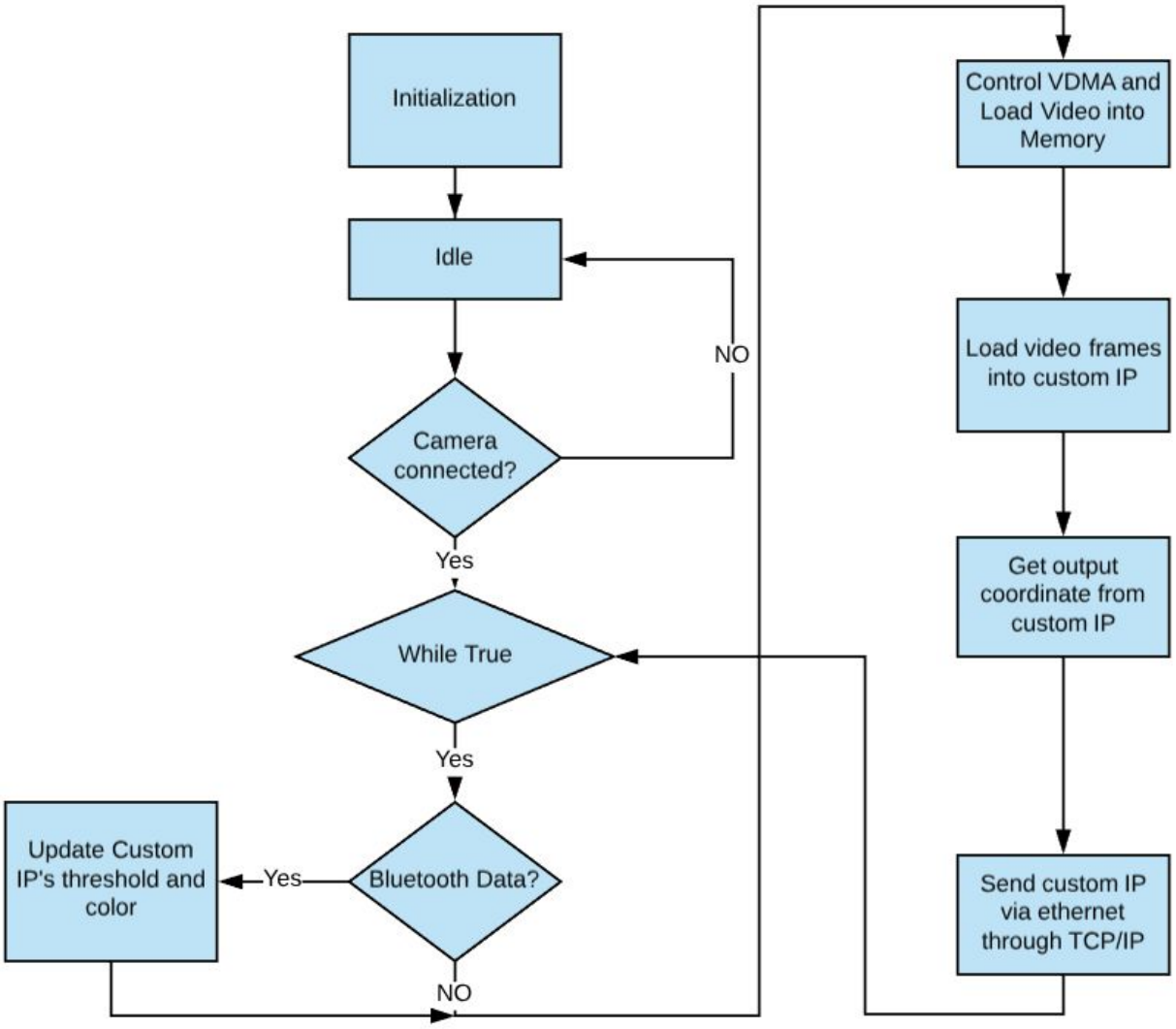
Target Color(slv_reg2)
First 8bits are used to identify target color(R,G or B). The DET_COLOR_CORD IP decides LED with which color to keep track based on the value.

Width(slv_reg3) and Height(slv_reg4)
Whole 32bits are passed to DET_COLOR_CORD IP as a width and height of the frame size of streaming input. On our design width is set to 1280 and height is set to 720 respectively.

Custom reset(slv_reg5)
Only bit0 is used as a custom reset signal to DET_COLOR_CORD IP. When the value is set to 1, the X,Y coordinate counters in DET_COLOR_CORD IP are reset to 0.

4.2 Microblaze

Our final design use microblaze as a top logic controlling unit. It properly initialises other IP blocks at the bootup, and handle the data communication in the FPGA. More specifically, the microblaze will execute a infinite loop, where it checks incoming TCP & bluetooth connection as well as controls VDMAs to move stream data in and out from memory. The logic flow of microblaze is illustrated as the flowchart below



4.3 Video block

The video block is built based on reference design provided by Digilent. Compare to the original reference design, we added one extra VDMA to move video frame data to our custom IP. The key IPs used in the block are Video In to AXI4-Stream IP, VDMA and AXI GPIO. The AXI GPIO is connected to HDMI pins to detect if the camera is connected. Once camera is connected, the GPIO will signal microblaze by interrupt. The incoming video data will be converted into AXI stream by Video In to AXI4-Stream IP and moved to memory by VDMA. Another VDMA will then move frames stored in the memory into the custom IP.

4.3 Ethernet block

The Ethernet block is also built based on reference design provided by Digilent[2]. The key IPs in the block are AXI DMA Controller, AXI 1G/2.5G Ethernet Subsystem and Timer. Ethernet Subsystem handles the communication with Ethernet hardware interface and provide AXI-stream interface for network data. The DMA is auto generated with Block automation in vivado to handle the communication between microblaze and ethernet subsystem. Since Lwip use timer for TCP connection, timer is required in the design as well. Compare to the original design, we made several changes. On the hardware side, we had to create a new clock wizard in order to integrate the ethernet block with our video block. For SDK code, we refactored the initialization code of LWIP and ethernet hardware so there was no conflict between ethernet and video initialization. In addition, we created our own TCP code instead of using the echo server. Last, it turned out there is a bug in 2018.2 vivado and we had to comment out a few lines of code in platform.c to resolve the issue.

```
    /* For detecting Ethernet phy link status periodically */
    /* commenting this line out otherwise SDK will keep showing LINK UP AND DOWN
    if (DetectEthLinkStatus == ETH_LINK_DETECT_INTERVAL) {
        eth_link_detect(echo_netif);
        DetectEthLinkStatus = 0;
    }
    */
}
```

4.4 Server Application

The server was designed to be able to generate the image based on the coordinates and button commands sent by the embedded processor. We used PyOpenGL, which is a open source graphical program, to achieve the drawing functionality.

The program was multithreaded so that one thread reads data via ethernet (since read is a block function) and another thread performs the actual drawing. Note that locks were used to ensure points being added by the ethernet thread and those being read by the drawing thread were performed atomically.

Each time the client would send X,Y location, it would be stored in memory in a list of class -

```
#class to store points
class Points():
        "Stores the x,y location and color of each point"
        def __init__(self, x, y, color):
        self.x = x
        self.y = y
        self.color = color
```

```
#list containing points that will be drawn in each frame
points_to_draw = []

#points get added when the button to draw is pressed
 if start_drawing:
        #insert into mem
        points_to_draw.append(Points(led_x,led_y,current_color))

#draw in each frame
 for i in range(0,len(points_to_draw)):
        drawOneCircle(points_to_draw[i].x, points_to_draw[i].y,
point_radius,points_to_draw[i].color)
```

The current color variable was set by the client and because of that, we can draw with the same color as is being shown on the controller LED.

Smoothing of the points were done when the person stopped drawing that particular segment. The code is as follows -

```
def smooth_points(start,end):

        points_list = []

        # generate a list with only X,Y points
        for index in range(start,end):
                points_list.append([points_to_draw[index].x,points_to_draw[index].y])

        #perform smoothing
        a = np.array(points_list)
        x, y = a.T
        t = np.linspace(0, 1, len(x))
        t2 = np.linspace(0, 1, len(y))

        x2 = np.interp(t2, t, x)
        y2 = np.interp(t2, t, y)

        sigma = 3 #determines the degree of smoothing
        x3 = gaussian_filter1d(x2, sigma)
        y3 = gaussian_filter1d(y2, sigma)

        x4 = np.interp(t, t2, x3)
        y4 = np.interp(t, t2, y3)
```

```
#update points_to_draw with the new smoothed points
i = 0
for index in range(start,end):
points_to_draw[index].x = x4[i]
points_to_draw[index].y = y4[i]
i = i + 1
```

The full code is included in our github repo

5. Description of Your Design Tree

See below for breakdown of the github directory which highlights the important folders and files within the project -

```
.
|---arduino
|   \---rgb_test/rgb_test.ino
|---docs
|   \---Final_Report.pdf
|   \---Demo_Video.mp4
|   \---532 FInal Demo.pdf
|---src
    \---draw_server/draw_server.py
    \---final_demo
            \---initial_demo.sdk
                \---hdmi_wrapper_hw_platform_1
                \---pure_led_detect
                \---initial_demo
    \---initial_demo.xpr
|---customIP
    \---HLS_source
    \---IP
        \---det_color_cord/impl
        \--- input_from_mb_extended_1.0
|---README.md
```

The arduino/ folder contains the source code that controls our wireless controller. It is used to change the color of the RGB LED, send data across Bluetooth and detect which button is pressed

The docs/ folder contains the final report, final slide deck and video taken during our final presentation

The src/draw_server folder contains the python code that acts as the server. It will start a server and open a graphical window, using PyOpenGL, that is used to draw the images on the screen.

The src/final_demo contains the Vivado project that instantiates our custom IP, which is used to detect the LED. Within the SDK folder, there are two main projects - the inital_demo is the FULL source code that is used to connect to the server, read from Bluetooth and custom IP and send packets across ethernet. The pure_led_detect is used to just test the custom IP by printing out the XY location of the detected LED. Please use hdmi_wrapper_hw_platform_1 when loading onto the board

The customIP folder contains the two custom IPs used in our project and their source. The customIP/IP folder contains two IPs that can be directly imported from Vivado. The customIP/HLS_source folder contains the cpp file used in the high level synthesis to generate the DET_COLOR_CORD IP. The source files for INPUT_FROM_MB can be found in customIP/IP/input_from_mb_extended_1.0/

6. Tips and Tricks

- Try to test each block/feature independently before trying to integrate.
- It is important to play around with ethernet buffer sizes on the server side. If buffer size is too big, it will buffer several inputs together and parsing the data becomes very difficult
- Don't fully trust the tool. Sometimes bugs are actually caused by the tool itself. Be prepared to dig into the provided code from the tool and debug in there.
- It is worthwhile to try to implement custom IP by using Vivado HLS if the project requires custom IP with AXI Stream port. Although it may take some time to learn how to use the tool, it saves much time once you know how to use it.

7. References

[1] Digilent/Nexys-Video-HDMI. [Online]. Available: https://github.com/Digilent/Nexys-Video-HDMI
[2] Digilent. Nexys Video - Getting Started with Microblaze Servers. [Online]. Available: https://reference.digilentinc.com/learn/programmable-logic/tutorials/nexys-video-getting-started-with-microblaze-servers/start
[3] arasht94/G5_HandGestureComputerInterface. [Online]. Available: https://github.com/arasht94/G5_HandGestureComputerInterface
[4]arasht94/G5_HandGestureComputerInterface. [Online]. Available: https://github.com/arasht94/G5_HandGestureComputerInterface/blob/master/src/hdmi.sdk/videodemo/src/vdma