

Winter 2017

ECE 544 Final Project Report: IoT based Control



Parimal Kulkarni, Chaitanya Deshpande,
Joel Jacob, Nishad Saraf
Portland State University
Winter 2017

Project Overview:

The purpose behind the project is to setup an IoT control interface via the internet with the help of the Nexys 4 DDR system. We used an Android phone as well as Gesture input from a webcam as a Control Interface and ESP 8266 Wi-Fi Module as a communication medium for controlling a Bot and provide accelerometer input to a game. As building the system with the MicroBlaze would have added more complexity to the project, we developed an easier interface to program Nexys 4 DDR which helped us focus on achieving the best results rather than investing time to program it.

We were able to implement MIPS-based Arduino core onto the nexys4 FPGA which helped interfacing and quick troubleshooting of our project modules.

We developed the Android interface using the “Blynk API” which is connected to “Blynk Server” to establish communication using the ESP 8266 module.

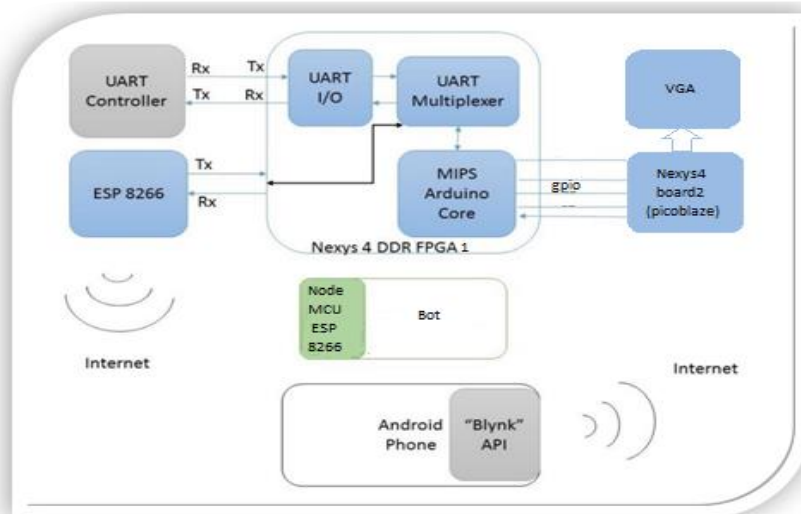
List of Components:

- The Nexys4DDR FPGA Board
- The Blynk API
- The Adafruit Huzaah ESP 8266 Wi-Fi Module
- The MIPS-based Arduino core
- The NodeMCU ESP 8266 based Bot

List of Software Used:

- Xilinx Vivado 2015.4
- Arduino IDE 1.8.1
- Python 2.7.13
- OpenCV 3.2.0
- Numpy 1.12.1
- Python Pip 9.0.1
- PyCharm 2016.2.2

High Level Block Diagram:



System Block Diagram

The block diagram above gives us an overview of the entire project, we have two Nexys4 FPGA boards, the first board is the **Hub** that connects to **board 2** as well as the **bot**. **FPGA 1** contains the **FPGAurduino** MIPS based core while the **second** board contains the “**Hunter’s Paradise**” game running on the **PicoBlaze** core. This board is connected to a **VGA** display.

The **ESP8266** connected to the **Hub** communicates with the **Bot**, while the second board is connected via the **GPIO** ports. The **Laptop** communicates with the **Hub** using a **serial** connection. The **Hub** accepts controls signals from both the **Android** phone as well as **Gesture** input from the laptop.

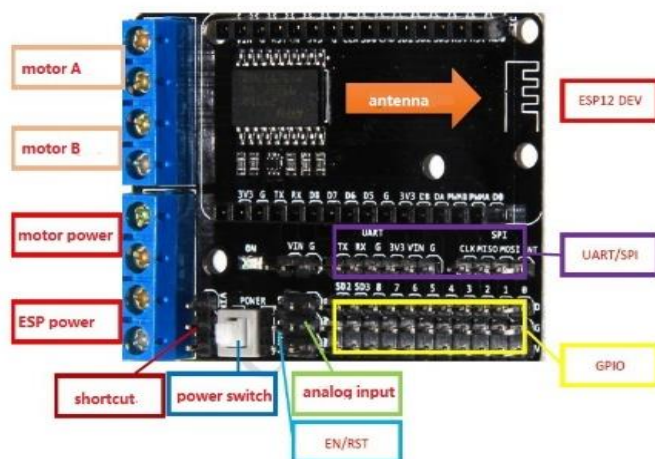
FPGAurduino Core:

- We came across the [FPGAurduino](#) port on Digilent & Altera boards during our ECE 540 Project.
- We studied the details and planned to use the core since it opened up opportunities of using Arduino libraries directly for programming any peripheral as well as saving our development time.
- Since the port for Nexys 4 DDR was not available online, it gave us opportunity to port FPGAurduino on Nexys 4 DDR board for the first time.
- After studying available documents and analyzing resources, we successfully ported FPGAurduino on Nexys4 DDR.
- We were able to successfully implement support for UART, GPIO, Switches, Pushbutton and LEDs on board.

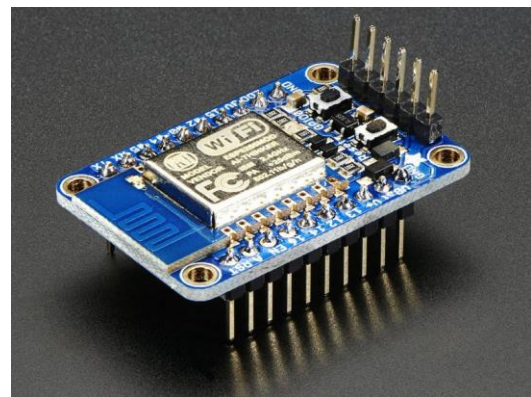
ESP 8266 Wi-Fi module:

The ESP8266 is a low-cost Wi-Fi chip with full TCP/IP stack and microcontroller capability produced by Shanghai-based Chinese manufacturer, Espressif.

- Some of its features are: 32-bit RISC CPU: Tensilica Xtensa LX106 running at 80 MHz
- 64 KiB of instruction RAM, 96 KiB of data RAM
- External QSPI flash - 512 KiB to 4 MiB (up to 16MiB is supported) IEEE 802.11 b/g/n Wi-Fi



NodeMCU ESP 8266 Breakout Board with Drivers



Adafruit Huzzah ESP 8266 Breakout Board

The ESP runs in two modes, a “**boot**” mode and a “**run**” mode. The boot mode is employed to flash a version of the program while the run mode loops the current version of the program. Thus, the ESP retains its current configuration even after a power down sequence.

The Blynk API:

Blynk is a Platform with iOS and Android apps to control Arduino, Raspberry Pi and the likes over the Internet. It's a digital dashboard where one can build a graphic interface for projects by simply dragging and dropping widgets.

We utilized this API to communicate between our android device and the ESP8266 Wi-Fi module.

The Blynk android app connects to the Blynk server via the internet. The ESP 8266 is programmed to connect to a Wi-Fi network, after which it talks to the Blynk server via a specific hardware port (Port: 8442 in our case). The commands sent by the android app are received by the ESP module via the Blynk server.

Latency is minimal and we can achieve real time control of the hub, which is very beneficial for our use case.

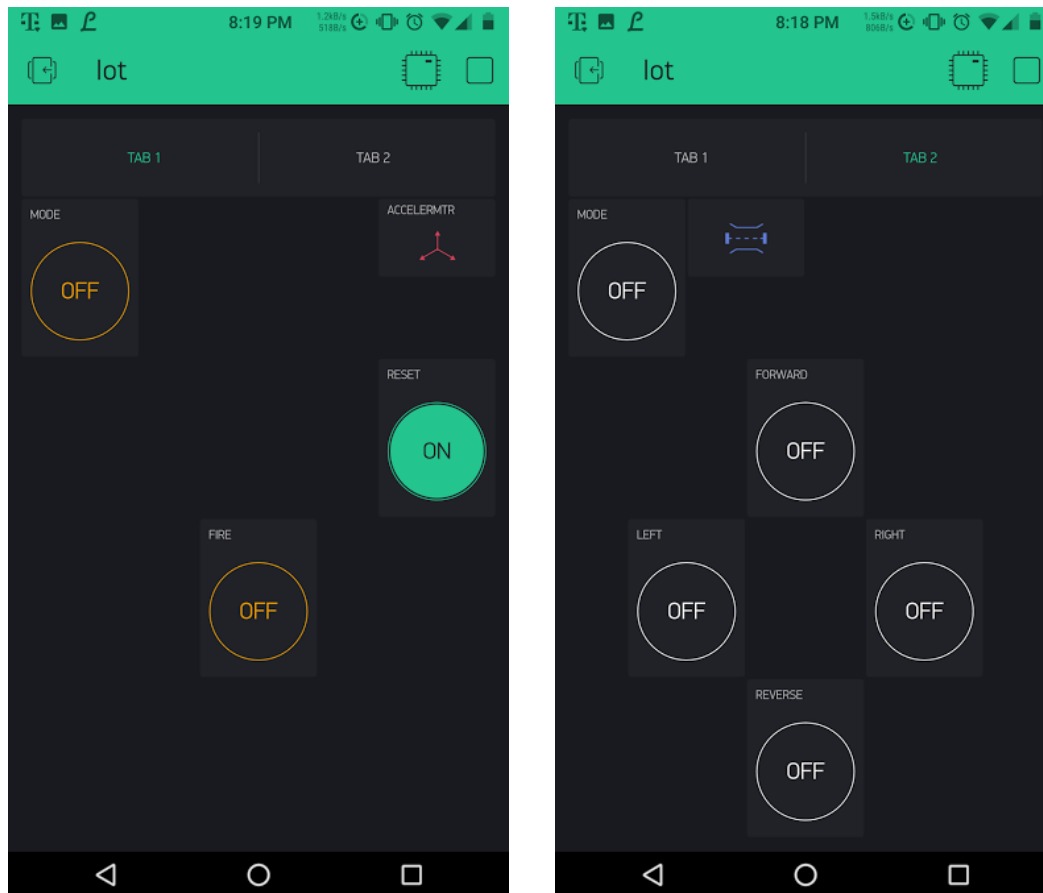
The Blynk android app is simple to use and provides a depth of options and supported features.

Blynk Bridge

Bridge can be used for Device-to-Device communication (no app. involved). We can send digital/analog/virtual write commands from one device to another, knowing it's auth token.

We can use multiple bridges to control multiple devices. We utilize this API to connect our main FPGA

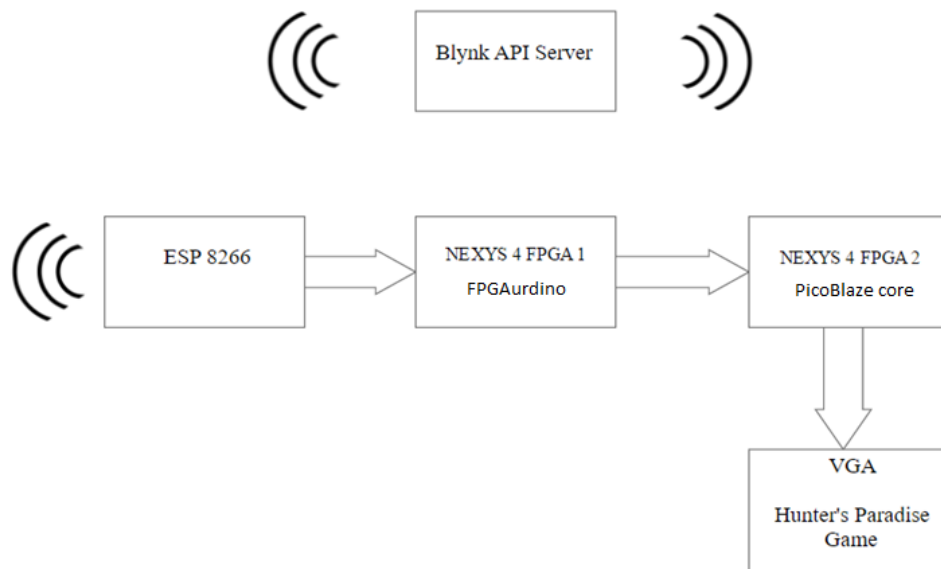
Board' ESP8266 with the NodeMCU ESP8266 on the Bot to establish connection and control via our Android phone as well as the Gesture control input.







The attached app screenshots describe how we mapped the controls for the Game and the Bot.

Hunter's Paradise game:

The VGA displays a hunter in a forest path with a bow and an arrow. He looks out for birds and animals to hunt, and when he spots one, he is expected to kill them before they disappear. The orientation of his bow is controlled by the 3-axis accelerometer available on the Nexys4 DDR board. The center push button is used to fire an arrow after aiming at the bird or the animal. The arrow follows a straight-line trajectory and when hits the animal/bird it scores a point. After a specific time, period the rate of appearance and disappearance of animals/birds increases, which acts as next level for the player.

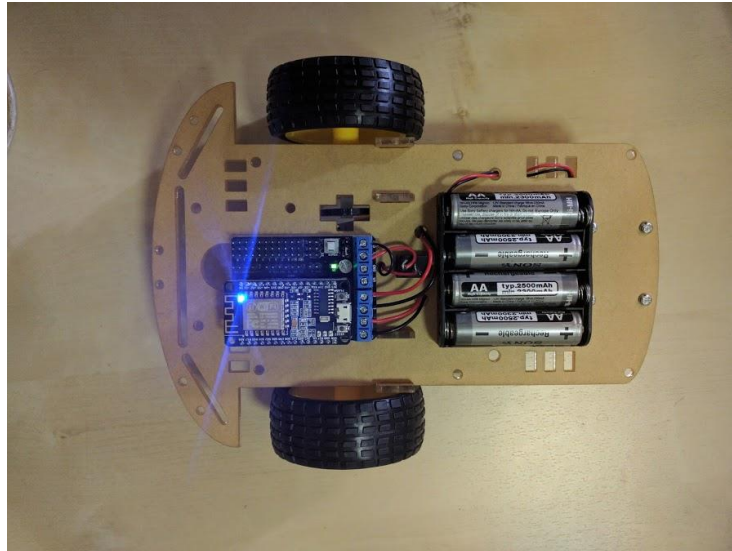


We modified the top module of game to get the input from the other board(FPGAduino) which is connected to android device via ESP8266 module. After checking the Y axis value of accelerometer that we got from the phone we found it varied from- 0 to 10 when the phone's orientation was varied from east to north. So, we mapped the values in the ESP8266 module for actual values used in the game.

Orientation	Accelerometer Value(board)	Phones value (as integer)	ICON
EAST-90°	0 or FF	All other values greater than -1 mapped to 0 in ESP8266 (when phone is pointing east)	
NORTH-EAST 30°	1	-4 to -2 mapped to 1 in ESP8266 (when phone is rotated anticlockwise)	
EAST-NORTH 60°	2	-7 to -5 mapped to 2 in ESP8266 (when phone is again tilted further)	
NORTH 0°	3 or 4	-10 to -8 mapped to 3 in ESP8266 (when phone is pointing north)	

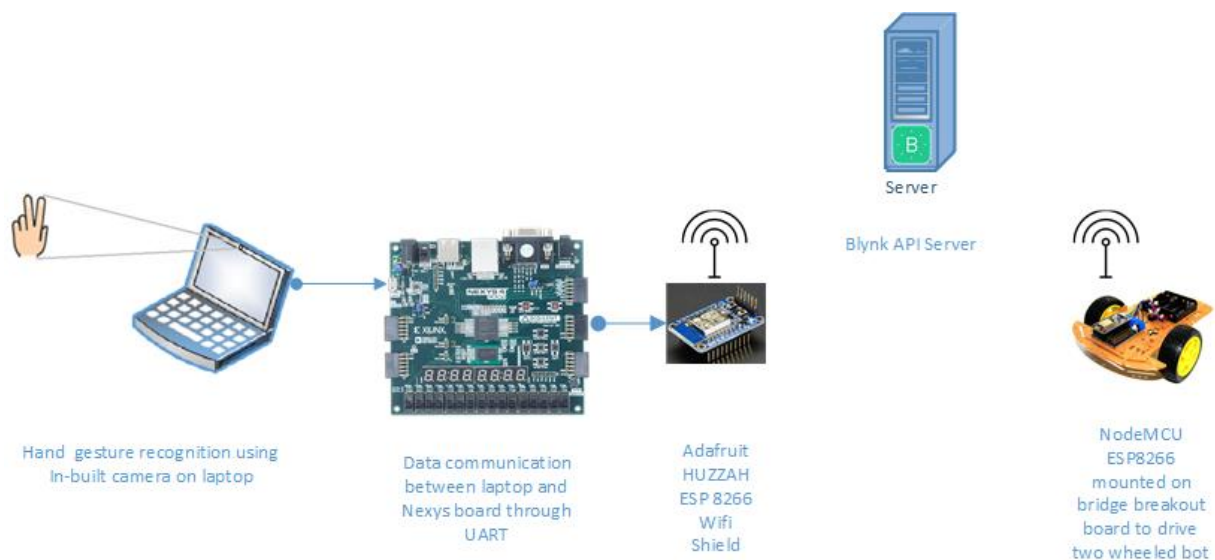
The Bot:

We used a bot that contained a NodeMCU ESP8266 with a Dual Hbridge Driver for out gesture control application. We used 4 AA Batteries to power the bot.



The Bot

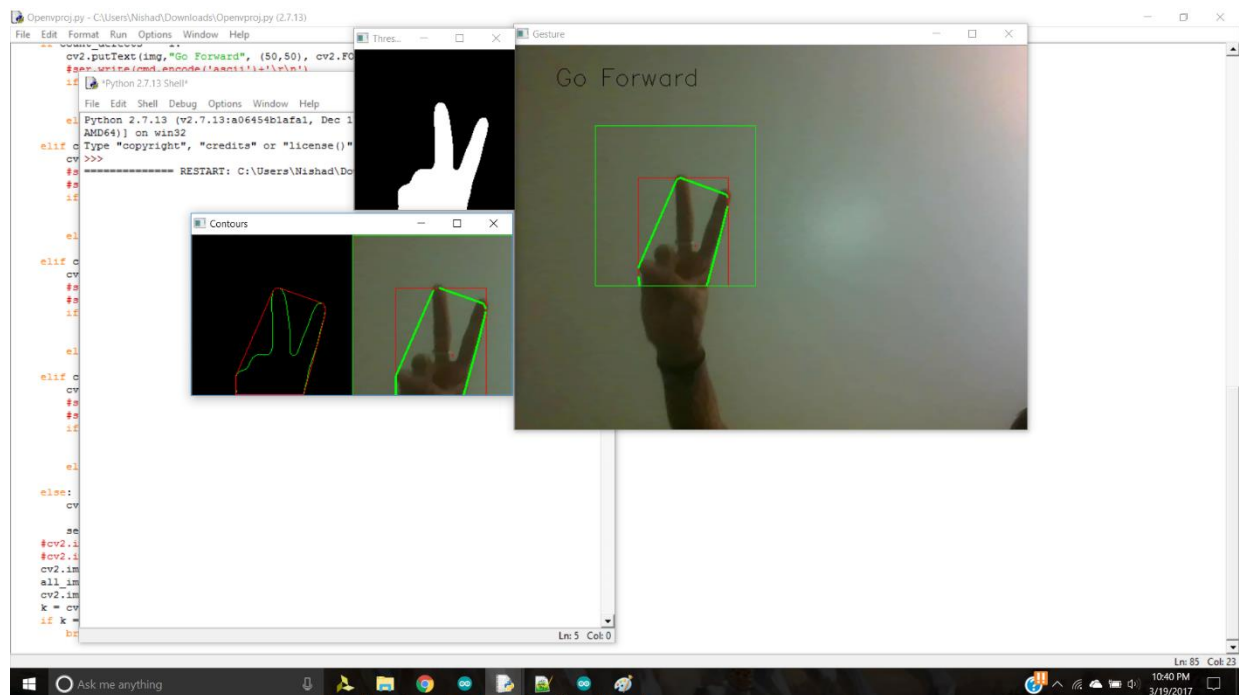
We utilize the **Blink Bridge API** to establish control between the main Nexys4 FPGA and the bot. We were able to send gesture values via the Serial Port on the laptop or control the bot via the Android phone with a flick of a mode switch.



The appropriate control gesture is captured on the laptop webcam and then is recognized using the image processing program in python. We then send the control signals via the Serial COM port to the main Nexys4 FPGA device running the FPGArduino MIPS core. The control signals are propagated to the onboard ESP 8266 and then the **Blynk Bridge API** passes the control information wirelessly to the bot's ESP8266.

Image Processing:

We have written the gesture recognition algorithm in python using OpenCV and Numpy libraries. We used python 2.7.13 for image processing algorithm.



Gesture Control Interface Using OpenCV with Python

Gesture recognition Algorithm: -

- 1) Start
- 2) Draw a red rectangle of fixed size for deciding contour area.
- 3) Count the number of objects coming in fixed size rectangle.
- 4) Detection of the object is done based on value of threshold, i.e. if the size of the object is above the value of threshold then only object is detected (counted).
- 5) If the size of the object is above the threshold value, then contour is drawn around the objects.
- 6) Gaussian filter is used for removing high frequency noise by making image blur.
- 7) Stop

Functions used from OpenCV: -

- **cv2.rectangle(image, pt1, pt2, color [, thickness [,lineType[, shift]]])**

Explanation: - For defining boundary where image processing will take place.

So basically, we are restricting the area of operation of image processing by drawing rectangle.

- **image** – Image.
- **pt1** – Vertex of the rectangle.
- **pt2** – Vertex of the rectangle opposite to pt1.
- **rec** – Alternative specification of the drawn rectangle.

- **color** – Rectangle color or brightness (grayscale image).
- **thickness** – Thickness of lines that make up the rectangle. Negative values, like CV_FILLED, mean that the function has to draw a filled rectangle.
- **lineType** – Type of the line. See the [line\(\)](#) description.
- **shift** – Number of fractional bits in the point coordinates.

➤ **cv2.GaussianBlur**

This function is used as a low pass filter which will blur the image by removing high frequency noise signals from the image. So only image of hand will be prominent and all other background noise will be removed.

GaussianBlur (src, dst, Size (i, i))

➤ **cv2.drawContours**

This function is used for drawing contour around edge of fingers depending on position of fingers shown in front of webcam of laptop.

To draw all the contours in an image:

cv2.drawContours (image, contours, -1, (0, 255, 0), 3)

➤ **cv2.circle**

For drawing circle, we need center co-ordinates of circle as well as radius of the circle.

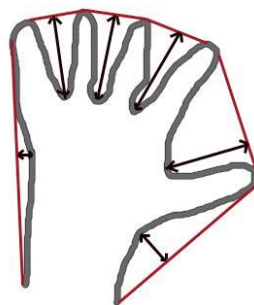
Circle is drawn inside a rectangle for defining contour area.

Example: - **cv2.circle** (crop_img, far, 1,[0,0,255] ,-1)

- **cv2.threshold:-** If pixel value is greater than a threshold value, it is assigned one value (may be white), else it is assigned another value (may be black).). So here the gestures which comes under the contour will get assigned as value 1 and other background noise will get assigned to zero value.
- cv2.threshold**(blurred, 127, 255, cv2.THRESH_BINARY_INV+cv2.THRESH_OTSU)

➤ **cv2.convexHull(): -**

Function checks a curve for convexity defects and corrects it. Generally speaking, convex curves are the curves which are always bulged out, or at-least flat. And if it is bulged inside, it is called convexity defects. For example, check the below image of hand. Red line shows the convex hull of hand. The double-sided arrow marks show the convexity defects, which are the local maximum deviations of hull from contours.



Example: - **hull = cv2.convexHull(points,returnPoints = False)**

points are the contours we pass into.

hull is the output, normally we avoid it.

returnPoints : By default, True. Then it returns the coordinates of the hull points. If False, it returns the indices of contour points corresponding to the hull points.

➤ **cv2.convexityDefects:-**

It returns an array where each row contains these values - [start point, end point, farthest point, approximate distance to farthest point].

Example: - cv2.convexityDefects(cnt,hull)



Challenges Faced:

- In default case, when presence of no fingers, our gesture recognition algorithm was sending the value which will be a condition for stop. But this caused lot of data to be transmitted over UART and we could not control the bot properly. So instead, we limited the transmission by adding a counter and checking its terminating value for transmission, thus, limiting the number of transmission values.
- While processing the control signals on bot due to optimized if-else ladder the bot used to stop before receiving the next acceleration signal in the queue. We found that the if-else ladder used to take a significant amount of time to reiterate through the entire loop which caused to bot to slow down. Optimizing the loop solved the issue.

Work Distribution:

Team Member	Work	Team Member	Work
Nishad Saraf	Serial Data communication, Image processing, Software Programming, hardware setup.	Joel Jacob	ESP8266 Analysis, Setup, and programming with FTDI USB Serial. Blynk API interface and Software Programming. Setup of FPGAurdino
Parimal Kulkarni	Accelerometer Interface with game, Image processing, Blynk API setup	Chaitanya Deshpande	Image processing, Serial Data communication, Bot control programming

Result:

We were able to successfully control the FPGA board and the output of the VGA display via the Android device. The boards were able to communicate with each other. Beyond what was proposed, we implemented gesture based control for a wireless bot using a camera connected to a computer and utilizing OpenCV to process the input image. In addition to this, we implemented FPGArduino which is a MIPS based core on the Nexys 4 FPGA.

References:

- <http://www.nxlab.fer.hr/fpgarduino/>
- <https://learn.sparkfun.com/tutorials/esp8266-thing-development-board-hookup-guide/example-sketch-blink-with-blynk>
- <https://github.com/f32c/f32c>
- <https://github.com/vipul-sharma20/gesture-opencv>
- <http://www.nxlab.fer.hr/fpgarduino/>
- <http://www.blynk.cc/>
- <http://docs.opencv.org/3.0-beta/index.html>
- <https://www.python.org/>
- <http://www.numpy.org/>