## 

**CprE 388 Android Lab Development**

Team May 15-20

## 

## Table of Contents

[Team Overview](#h.31j5qaja4lmi)

[Problem Statement](#h.gcvmgqov46xu)

[System Description](#h.hxmoq6u2xce2)

[Lab Breakdown](#h.ecz5tq2zckud)

[Lab 1: Morse Code](#h.njwmb8vabiwg)

[Lab 2: Sketchpad](#h.34lgp5xiifz9)

[Lab 3: SpyCam](#h.x6rtppmltb6j)

[Deliverables](#h.5w00bv1qapos)

[Schedule](#h.mhgmyhxfbc4q)

[First Semester](#h.1emhjdwv46uh)

[Second Semester](#h.vrxt10ntierr)

[Specifications](#h.497394usrkms)

[Resources Required](#h.42ye0crvsgjc)

[Standards](#h.5ippt6g51kz8)

[Risks](#h.lvwvpz7pc83v)

[Testing Approach](#h.fkg1dxgm9y7m)

[Testing Results](#h.qy3ota9p1w7g)

[Morse Code](#h.eiui405n4mnd)

[SketchPad - Part I](#h.5lvkkynlyzdm)

[SketchPad - Part II](#h.jch9ojto3995)

[SpyCam](#h.amm0wrhsheyq)

[Analysis of Results](#h.7dfjkc487k40)

[Appendix I: Operation Manual](#h.l929qhp6talp)

[Setting up Arduino IDE:](#h.h2dwvr4s70f0)

[Setting up Android Studio:](#h.n9js2zqxpwb4)

[Appendix II: Alternative Designs](#h.lypxpyoe70sn)

[Board Selection:](#h.fvwnjzwbfjaq)

[Lab Graveyard:](#h.qpcnixu0oyqw)

[Appendix III: Other Considerations](#h.hpgezwe7kjjp)

[Appendix IV: Code and Lab Manuals](#h.rrhj4hkajgqn)

## Team Overview

Angela Borgman - Computer Engineering

aborgman@iastate.edu

Alex Jensen - Software Engineering

abjensen@iastate.edu

Ben Turbes - Computer Engineering

bcturbes@iastate.edu

Ryan Dincher - Computer Engineering

rdincher@iastate.edu

## Problem Statement

Currently, the CprE 388 class is an Android development class. The course is listed as ‘Embedded Systems II’ and while Android is a form of embedded systems it is lacking some of the lower level experience gained by working with hardware components. Our project aimed to develop a hardware infrastructure and 3 labs to be added to the existing class. This required finding hardware that worked with the Android devices currently used (Nexus 7 tablets) as well as allowed students the opportunity to use interesting hardware that showed them how the labs are applicable to technology seen on the market today.

## System Description

The setup will consist of a Nexus 7 Android tablet connected to an Arduino ADK board attached to a computer running the Android Studio SDK.The Arduino board will be used to communicate with external hardware in the form of an LED board.

The LED board comes with libraries to be used with the Arduino that will make some communication easier for students. This should make the expected completion time for the labs more reasonable. There are also libraries provided by the Arduino for communication between Android and Arduino. The LED board also requires more power than the Arduino can provide, so an external power supply is necessary.

The computers themselves will be Windows machines. They must have USB drivers installed on them for the tablets to work with them.

## Lab Breakdown

Our project requires that we come up with 3 labs that involve the Arduino ADK board and some form of external hardware. This includes a rough breakdown of our current plan and at what completion point each of the labs is at.

### Lab 1: Morse Code

This will be a 2-part lab aimed at introducing students to the LED board that they will be using for most of the labs following. The first part is a simple “blinker app” that will involve a simple button on the Nexus Tablet that can toggle an LED on the board on or off.

Part 2 is creating a “Morse Code translator” . Users should be able to type a string on the Nexus 7 tablet and the LED board should flash with the Morse code equivalent in long or short blinks. This is very similar to part 1, but takes it one step further requiring the use of the LED board.

Learning Objectives:

* Communication between the tablet and Arduino
* Experience writing Arduino Code
* Adding basic hardware to the Arduino

### Lab 2: Sketchpad

This is a two-week lab requiring that students create an application on their tablet that allows them to light up a corresponding light on the LED board by clicking a button.

The first week is focused on the hardware side of the lab. Students must create a very simple app that allows a user to type in an X and Y coordinate and RGB values. This will then light up the LED at that X-Y location and the colors.

The second week is more focused on the UI of the android application. Students will be required to create a 32x32 grid of buttons that will be used to select the coordinates. It will require 3 sliding bars, for the red, green and blue values of the colors.

*Learning Objectives:*

* Learn how to use additional hardware with the Arduino
* Create own format for serial messages
* Create different user interfaces to accomplish the same task
* Show real world applications of embedded systems

### Lab 3: SpyCam

This lab uses a wireless IP camera that students will be able to stream video and audio from. Students will be required to create an application where they can see the live video and move the camera. This lab requires the use of asynchronous tasks in order to stream the video and audio.

*Learning Objectives:*

* Experiences wirelessly connecting to hardware
* Using asynchronous tasks
* Complicated UI design

## Deliverables

There are a variety of deliverables that will be given to our client at the end of CprE 492. These materials will be provided to our client/advisor Dr. Tyagi in a folder. All hardware materials will be given to for use in the future class.

* Complete lab manuals for four weeks of labs.
* Four evaluation forms outlining the important deliverables students will be graded on.
* Skeleton code for each lab in a zip file.
* A completed example of each lab to be used by future TA’s.
* A setup guide outlining what libraries, software packages and other materials are needed for a lab station. This can be used by the University for future lab station setup and by students wanting to setup a development environment on their personal computers.
* Materials guide including models and descriptions of each piece of hardware used in the lab setup.

## Schedule

### First Semester

We will have full understanding of the Arduino ADK board being used and the components.

All hardware will be ordered by mid-November.

All labs will be roughly planned, with one lab roughly written and completed. The is will allow us ample time to test the labs in the next semester.

Draft of setup guide complete.

### Second Semester

All labs written and initial testing done by the middle of March so that user testing can be done.

Will have students complete labs to evaluate their usability and understandings gained from them.

Labs will be re-evaluated after reading through feedback from students and re-written to reflect this feedback.

Finished and polished guide is complete for future use with the class.

## Specifications

The project should include 4 labs working with the Nexus 7 tablets and the Arduino Mega ADK board. These labs will also involve LED boards that will require students to interact with hardware outside of the LED board.

The labs will be written out with clear requirements, but should require students to think critically about the task. The physical setup for each lab should be as simple as possible so that there are as few issues as possible and the labs can focus more on learning and less of troubleshooting.

## Resources Required

|  |  |  |
| --- | --- | --- |
| **Resource** | **How will we get it?** | **Estimated cost** |
| Arduino Mega 2560 board | Provided by client | $59 x N |
| Printed Circuit Board Shield | Provided by client | $12 x N |
| Nexus 7 Tablets | Provided by department | $0 x N |
| LED Board | Provided by client | $40 x N |
| IP Camera | Provided by client | $50 x N |

\*N indicates the number of lab stations required by the department.

## 

## Standards

Halfway through our project, Google changed the official recommendation for a development environment from Eclipse to Android Studio. As such, we changed our write-ups and what we used during our test phase to reflect this change.

While writing our lab documents and creating other supplementary materials to be used by students we followed the form of the previous CprE 388 lab write ups. As these documents are intended to be used for one class, we kept the layout and general format the same so it would make sense to students who had already gotten used to the previous format.

## Risks

* Documentation that we have found to interact with the Arduino is not sufficient, then we will have to research further for proper documentation.
* Reliability of the hardware for consistency in lab is important and may pose a challenge.
* Getting the USB drivers installed on the lab computers is a challenge as it currently requires Administrative access.

Solutions

* Through reading documentation and testing on our own we have put together lab documents that explain everything needed to know. We have also provided a “Getting Started Guide” that can be used for the setup of future lab stations.
* By selecting hardware with good reviews and thorough testing we have found reliable hardware that works consistently.
* After working with ETG extensively, we have found a solution that allows any tablet to use the drivers on any of the lab computers. Previously, each student could only use one lab computer for the whole semester.

## Testing Approach

The testing process for this project involved two parts. The first, was testing hardware and evaluating difficulty throughout the project. As hardware was selected and ordered we would test it by trying to do basic labs and establish communication. Once the specific details of each lab were decided, the group working on that lab would go through it and create a working prototype of the lab. This was used as the basis for time estimate for the lab.

Once there were working version and lab manuals written up for all of the labs, the next phase of testing was setting up and experimental class. This class allowed us to have students, like those that would be taking CprE 388 next semester, do the labs and provide feedback on them. Each student was required to fill out a feedback form to provide us with statistics and information about how long it took them.

Once the information from the test class was collected, we edited the labs taking this feedback into consideration. This testing approach gave us the opportunity to evaluate how well we completed our functional and nonfunctional requirements.

### Testing Results

The results of the feedback forms collected from the experimental class are summarized below along with what about the lab we changed. In our feedback form we requested students give us a combination of quantitative and qualitative measurements of the labs. The data we used to improve our lab documents are as follows:

Average time spent outside lab - how many hours outside of lab it took to complete the lab.

Average rating - What the student’s overall rating of this lab was on a 1 to 5 scale (1 being hated it, 5 being couldn’t love it more).

Average Difficulty - The rating on how challenging the problem given in the lab was. This was also on a scale of 1 to 5 (1 being way too easy, 5 being way too hard).

Lab manual suggestions - common concerns and improvement suggestions given by students.

#### Morse Code

Average time spent outside lab: 1 hour

Average rating: 4.09/5

Average Difficulty: 3.36/5

Lab manual suggestions:

* Wording when describing how Morse Code works.

What was changed: Updated description and added a title to break the text out from the rest of the lab description.

#### SketchPad - Part I

Average time spent outside lab: 1 hour

Average rating: 4.5/5

Average Difficulty: 3.06/5

Lab manual suggestions:

* Add the requirement to use as few bytes as possible in communication between Arduino and LED board.

What was changed: Ultimately it was decided that the manual should remain unchanged. We felt this was a design choice students needed to make for themselves, and it we added that it would undermine the learning objectives we were aiming for.

#### SketchPad - Part II

Average time spent outside lab: 0.87 hours

Average rating: 4.47/5

Average Difficulty: 3.65/5

Lab manual suggestions:

* A better explanation of the *generateGrid()* method and what it is supposed to do.
* Include a diagram of the grid for a clearer explanation of the calculations needed to find the coordinates for the addition of drag and draw.

What was changed: A diagram showing the grid of buttons and the dimensions is now provided. This is a better way to convey how to calculate based on an x and y coordinate of a touch event what button the event happened over.

The description of the *generateGrid()* method has also been updated to clear up some of the questions students had when completing this lab.

#### SpyCam

Average time spent outside lab: 1.82 hours

Average rating: 3.97/5

Average Difficulty: 3.7/5

Lab manual suggestions:

* Add TODO tags to the skeleton code provided for the lab.
* Provide additional notes on what is wrong in the data sheet.

What was changed: TODO tags were added to the skeleton code to make it clearer where the methods students need to complete can be found. Another hint was added to clear up the confusion regarding unnecessary brackets found in an example in the datasheet.

### Analysis of Results

Overall, we were quite happy with our results as was our advisor. These labs were designed to take between 2 and 4 hours and each lab’s average fell into that range. In addition to this, the overall opinion of the labs was consistently quite high, with the Sketchpad App being the clear favorite. The average difficulty on each lab fell between 3-4 out of 5. A 5 on our scale would indicate that the lab was “way too difficult” and a 1 would indicate the lab was “way too easy”. On this scale, the overall challenge rating is skewed just a little towards the challenging side. It seems that the data collected from our test class shows that our labs were both interesting to students as well as challenging without being so hard it became overwhelming. The labs we created balance challenging problems that require critical thinking with interesting problems that leave students with a program that does something unique and interesting.

## Appendix I: Operation Manual

## Setting up Arduino IDE:

Download the Arduino IDE from <http://www.arduino.cc/en/Main/Software> and run the installer.

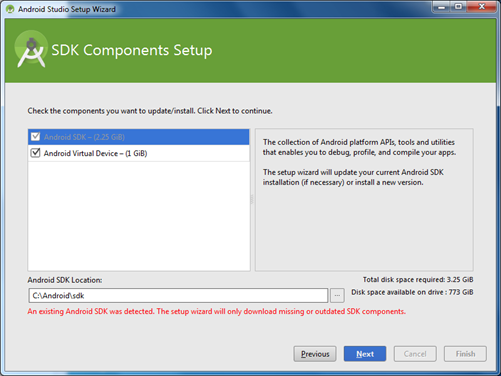
After installing the IDE, there are some some additional libraries that need to be installed in order to complete the labs. First, the Android Accessory libraries need to be downloaded from <https://dl-ssl.google.com/android/adk/adk_release_20120606.zip>. Extract the zip and navigate to the arduino\_libs folder. Copy AndroidAccessory and USB\_Host\_Shield, navigate to the Arduino IDE install location, and place them in Arduino\libraries. These libraries should now be installed.

Next, go to <https://github.com/adafruit/RGB-matrix-Panel>, download the project as a zip, and install the RGB-matrix-Panel library the same way as before.

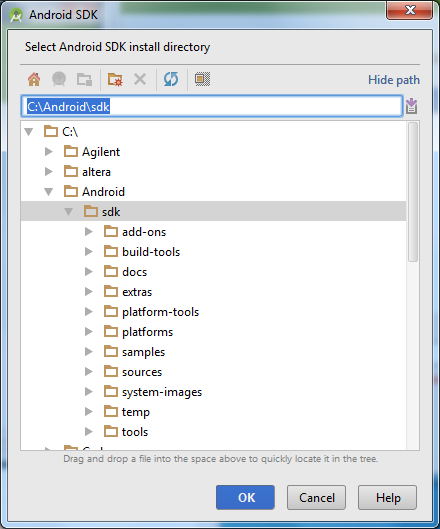
## Setting up Android Studio:

Download Android Studio from <https://developer.android.com/sdk/index.html> and run the installer.

After installing Android Studio, the Android SDK also needs to be installed. This can be done the first time you launch Android Studio. Run Android studio and you will be prompted to select a location for the SDK Install.



For lab computers use *C:\Android\sdk* as the SDK location. If the SDK is already on the lab computer, it will perform an update of the SDK, otherwise it will download and install it. Using this location ensures that the SDK will only be downloaded once per lab station, and each student will not re-download the SDK.



Finally, the Google USB driver needs to be installed. Instructions for installing the most recent version of the driver can be found at <http://developer.android.com/tools/extras/oem-usb.html>.

## Appendix II: Alternative Designs

### Board Selection:

The external hardware selection was one of the most important features of this project. The board selected needed to allow a variety of hardware to connect to it and needed to have extensive documentation and support to allow for future support.

Alternatives Considered:

Green Bean Board - Board developed by GE, and would only work with their appliances. It would have required lab stations featuring small (or large) appliances. Money and space considerations made this not a feasible solution.

Google ADK - A development board given out at Google I/O several years ago. Not really a feasible option, because while it allowed for a lot of potential projects they are not available for purchase and had very little support available.

Ultimately, the Arduino Mega ADK was chosen as it had a lot of projects available to it and a large number of support resources.

### Lab Graveyard:

Through the design process, there were several labs that were considered before the final selection was made. For a variety of reasons, it was decided these labs did not meet our requirements and were not included in the final selection.

Heart Rate Monitor Lab: The previous 388 class already uses heart rate monitors that connect to the Nexus 7 tablet using bluetooth. This lab would add an additional portion of creating a visual representation of the heart rate. For example, a target, high, and dangerous level would be given and the LED board would display a corresponding color. This lab was not used because it did not add any experiences for students and did not have the interest level we were aiming for.

Coffee Pot Lab: This lab would be used to demonstrate how these skills could be used outside of lab to create an at-home “smart” coffee pot. Students would be able to turn the coffee pot on or off using the Nexus 7 tablet. Ultimately, this would have required building a coffee pot that would allow the tablet to connect to it. There were safety concerns associated with this as the heating element would be exposed, and the relay required to build it would always have a current flowing through it. The lab itself would also have become trivial as students would have done essentially the same thing in the Morse Code lab. It was decided that this lab would be replaced by the SpyCam lab.

## 

## Appendix III: Other Considerations

Order hardware early: In order to test the project, there was an experimental class created where these labs could be used by real students in a lab environment. One problem that occurred fairly early on was ordering hardware for this experimental class. The order was placed one month before the class was to begin, however our advisor did not give explicit permission for these items to be purchased. In addition, after two weeks of not receiving any information about the item whereabouts, it was discovered that the order had never been placed. As soon as this was discovered, the order was placed but by then the LED boards required had gone out of stock.

Additionally, the day of the first lab using the Arduino boards, we realized the boards we ordered were not the correct ones. In all of the ordering confusion, a link for the Arduino Mega 2560 was given instead of the Arduino Mega ADK. The problem had to be solved quickly with only the resources on hand. Unfortunately, the lab had to be run with the ten lab groups sharing two Arduinos and one LED board. In the end, all of the hardware needed arrived two days after the last lab. Ordering early and triple checking what is being ordered was a lesson learned.

## 

## 

## Appendix IV: Code and Lab Manuals

In this lab you will be creating an application that will translate sentences into Morse Code. The application should allow you to type a message into your application and then blink the Morse Code translation using the LED on the Arduino.

**Learning Objectives:**

* Learn how to use a USB accessory with the Android Tablet
* Create a functional dual component application
* Get experience with Android and Arduino programming

**Before You Start:**

Download blink\_app.ino, BlinkApp.zip, morse\_code\_shell.ino and MorseCodeShell.zip.

**Part 1 (Blink App):**

The code for the Blink App has already been written for you. It is a simple app with a button that will toggle an LED on the Arduino on and off. Installing and running this app will help you to understand how to create new Android accessories using the Arduino and complete the rest of this lab.

**Arduino:**

To start, connect the Arduino to a USB port on your computer using the USB A to B cable. Check to make sure the green “ON” LED is lit.

Open blink\_app.ino with the Arduino IDE. You will be prompted to move the file into a folder called “blink\_app”, press OK to continue.

Notice the setup() and loop() functions. The setup function is called once when the Arduino is powered on, or when the the reset button is pressed. After that, the loop function will be called continuously.

To compile and upload code to the Arduino, you first must select the correct board and serial port.

Board selection can be found under***Tools*** *>* ***Board*** *>* ***Arduino Mega 2560 or Mega ADK****.*

To determine the correct serial port, open the ***Control Panel***on your computer and select ***Hardware and Sound*** > ***Devices and Printers***. On this screen, you should see a device labeled *“Arduino Mega ADK”* followed by the correct COM port. Finally, in the Arduino IDE, select ***Tools*** *>* ***Serial Port*** *>* ***COM?***(where ? is number you just located).

Finally, click the “Upload” button in the top left corner of the Arduino IDE.

**Android:**

In Android Studio, import the the BlinkApp project for Android by going to ***File*** > ***Import Project…*** and following the dialog to complete the import.

After importing the project, review the code in Blinker.java and try to understand it. Pay close attention to the blinkLED() method as you will be implementing something similar in the next part of the lab.

Connect your Android device to the computer, and then install the app on your device.

Finally, disconnect your Android device from the computer and connect it to the USB port on the Arduino. This should launch the BlinkApp on your device. Press the button to toggle LED on the Arduino board.

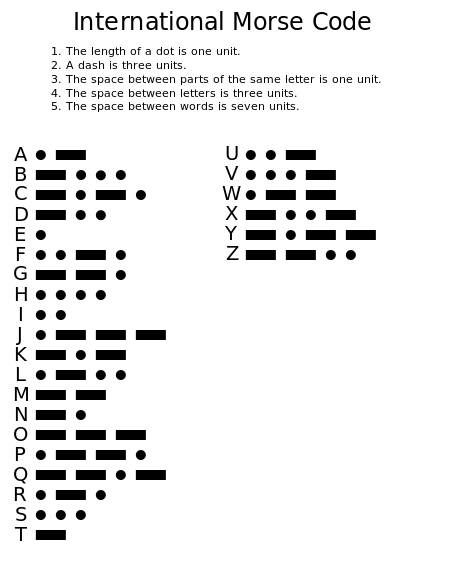
Demo this to your TA.

**Part 2 (Morse Code Generator):**

For this part of the lab, you will be creating an app which allows you to enter a message on your Android device and display it in Morse Code on the Arduino by toggling the same LED as the BlinkApp.

**What is Morse Code?**

A message in Morse Code consists of a series of dashes and dots (or longs and shorts). These dashes and dots can then be displayed by turning the LED on for the correct number of time units *(longs and shorts)* and then off for the correct number of units between parts of a letter, whole letters, and words. The rules for displaying morse code can be found in the figure below. *(Note that for this lab the code to convert between text and morse code has already been written for you. Your task will be to interpret the morse code and display it correctly).*



**Arduino:**

Open morse\_code\_shell.ino with the Arduino IDE like in the previous part. You should notice a very similar layout to blink\_app.ino.

For the Arduino portion of this lab, you will be required to complete the *displayMorseCode()* function, to toggle the LED on and off to display the message in morse code. The*displayMorseCode()*method accepts the message and the length as parameters. We need to interpret the message toggle the LED on and off to display the morse code.

The message consists of the following values, which have been defined as constants:

* SHORT: *a dot (or short) in morse code.*
* LONG: *a dash (or long) in morse code.*
* LETTER: *the end of a letter in morse code.*
* WORD: *the end of a word in morse code.*
* STOP: *the end of the morse code message.*

For example, the message “SOS” would be encoded as (SHORT, SHORT, SHORT, LETTER, LONG, LONG, LONG, LETTER, SHORT, SHORT, SHORT, LETTER, WORD, STOP).

Implement this method by turning the LED on and off the same way it is done in blink\_app.ino. In order to toggle the LED for a certain length of time, use the *delay()* function. In your code, call *delay(UNIT)* to pause execution of your code for UNIT milliseconds. Using the UNIT constant will allow the length of one unit to easily be changed without altering your code.

You can now compile and upload your code to the Arduino.

**Android:**

In Android Studio, import the the MorseCodeGeneratorShell project for Android by going to ***File*** > ***Import Project…*** and following the dialog to complete the import.

Now that you have imported the project, you will need to add a EditText and Button to the application’s layout. Pressing the button should call the *blinkLED()* method in your java code.

The EditText should be configured so that it only accepts letters and spaces. It will also not accept more than 25 characters of input. This can be done by using the ***android:digits*** and ***android:maxLength*** attributes in your EditText in the layout.

*Writing openAccessory() Method:*

The openAccessory method is used to create the connection to the USB accessory, there are a few instance variables available to use.

1. To start we need to open the connection to do this add the following line

*mFileDescriptor = mUsbManager.openAccessory(accessory);*

mFileDescriptor is a file descriptor to handle the accessory. mUsbManager contains a method to detect if there is an accessory connected to the tablet and attempts to connect to the device.

1. Below this line we need to make sure that the file descriptor actually contains the opened accessory so we can use it. To do this we want write an if statement to check that mFileDescriptor is not null. If it is not null, then the accessory was opened successfully. If it is null, then the accessory failed to open and we should log this by adding the line: *Log.d(TAG, “accessory open fail”);* This will display in Android Studio that the accessory failed to open.
2. If the descriptor isn’t null, we need to save the accessory and open an input and output stream. To do this, add the following lines inside your if statement.  
      
   *mAccessory = accessory;*

*FileDescriptor fd = mFileDescriptor.getFileDescriptor();*

*mInputStream = new FileInputStream(fd);*

*mOutputStream = new FileOutputStream(fd);*

*Log.d(TAG, “accessory opened”);*

*Writing blinkLED() Method:*

The *blinkLED()* method gets the text from your EditText, encodes it, and send its to the Arduino to be displayed.

The message can be encoded by using the *encodeMessage()* method. This will convert the message to the morse code format and return a byte array containing the encoded message.

*private byte[] encodeMessage(String msg);*

Look at the BlinkApp example above for ideas on how to send the encoded message to the Arduino. Finally, upload the app to your Android device, and then connect it to the Arduino board as before.

Demo the lab to your TA.

In this lab you will be creating an app which allows you to draw on a 32x32 RGB LED board through the Arduino. Your app will have two different user interfaces to choose from. The first one will be a simple interface where you will type in values to control the board. The second interface will be more complicated but will allow you control the board by drawing on a grid in the app.

**Learning Objectives:**

* Learn how to use additional hardware with the Arduino
* Create own format for serial messages
* Create different user interfaces to accomplish the same task
* Show real world applications of embedded systems

**Before You Start:**

Download sketch\_pad\_shell\_arduino.zip and open sketch\_pad.ino with the Arduino IDE.

Download SketchPadShell.zip and import the project in Android Studio.

Connect LED board to Arduino if it is not already.

**Arduino:**

For the Arduino portion of this lab, you will be creating your own serial message format and writing code to interpret this format. Later you will be writing Android code to send a message in this format. While implementing the serial message think about how you sent data in the Morse Code Lab.

Your serial message should include the following information.

* A signal to erase the entire board.
* A signal to display a pre-loaded bitmap image.
* An x, y location of a pixel to modify.
* RGB values for the color of the pixel.

*Erasing the Board:*

Your message should contain a signal to erase the entire LED board. When this signal is activated, call the *blankEasel()* function to clear the board.

*Displaying Pre-Loaded Bitmap:*

This is very similar to how you erase the LED board. This time, when your signal is activated, call the *loadBitmap()* function.

*Reading X, Y Location:*

When you read the x and y values from your message, you should set the global variables cursorX and cursorY to those values. This is important because the drawing function relies on these values.

*Reading RGB Values:*

When you read the RGB values from your message, set the global red, green, and blue variables to their corresponding values. Again for use in the drawing function *(Note that the LEDs on the board only support RGB values from 0-7. This means that your message should only include RGB values that are also in this range).*

*Drawing a Pixel:*

This should be done last and only if you neither erased the LED board or loaded the bitmap. After you have set cursorX, cursorY, red, green, and blue, you can call drawDot() to set the pixel at (cursorX, cursorY) to the current color. *(Note that cursorX and cursorY should be limited to the dimensions of the LED board, 0-31).*

**Android** - **Part 1 (Coordinate Editor):**

The Android portion of this lab is divided into two parts. In each part, you will be creating a different user interface to draw on the LED board. In this part, you will be creating a simple interface which allows you to manually enter coordinates and RGB values to draw on the board.

*Starting the Coordinate Editor:*

When the SketchPad app is started, the user is presented with a screen with two buttons to choose which interface they want to use, Coordinate Editor or Sketchpad. In MainActivity.java, create a method to launch the CoordinateEditor activity. *(Hint: Remember that Intents can be used to start a new activity).*

*Erasing the Board:*

In CoordinateEditor.java, there is a method called *erase()* which is called when the “Erase” button is pressed. Complete this method, so that it will send the erase signal that you created earlier to the Arduino board. This should be done similarly to the *blinkLED()* method in the Morse Code Lab.

*Displaying Pre-Loaded Bitmap:*

Repeat the process you used to complete *erase()* method, this time to complete *loadBitMap()*.

*Drawing a Pixel:*

In this step, you will complete the *changeColor()* method. This method will get the x, y coordinates and RGB values from their corresponding EditTexts and send them to the Arduino to be displayed on the LED board. *(Remember that RGB values for the LED board must range from 0-7).*

**Demo:**

After you have completed these steps, upload the app to your Android device and connect it to your Arduino board.

Demo this part to your TA.

**Android** - **Part 2 (SketchPad Interface):**

For this part of the lab, you will be creating a much more powerful user interface, which will allow you draw directly on your android device and display your drawing to the LED board.

**Initializing UI Components:**

The first step to this part of the lab is to ensure that the different UI components and instance variables are initialized and add the required listeners. You will do this in the *initControls()* and *generateGrid()* methods, which are called in *onCreate()*.

*Initialize Controls:*

The majority of the initControls() method has already been completed. The instance variables for the red, green, and blue, SeekBars have already been initialized as well as eraserButton.

Your job is to initialize the *color* instance variable *(private int color)* and *colorPreview (private ImageView colorPreview)* based on the progress of the SeekBars. There are a few methods that will help you complete this task.

Calling *getProgress()* on your SeekBar will return its progress. Since RGB values on the LED board must be between 0 and 7, the value returned will also be from 0-7. Calling *Color.rgb()*, will convert a red, green, and blue, value to a single integer. You should be able to use these methods to initialize *color*. *(Note Color.rgb() is expecting RGB values from 0-255. Multiplying the progress of your SeekBars by 36 is a good approximation of this value).*

After you have initialized *color*, you will need to update *colorPreview* so it will display your color. To do this simply add the following lines.

*colorPreview = (ImageView) findViewById(R.id.color\_preview);*

*colorPreview.getDrawable().setColorFilter(color, PorterDuff.Mode.SRC);*

This will initialize the instance variable and then apply a color filter to the Drawable contained in the view. By using your color and the filter mode *PorterDuff.Mode.SRC*, you are telling the Drawable that the new color should become the source and instead of applying another filter based on your color.

Finally, complete *onProgressChanged()* for the new Listener that is defined in this method. This should update *color* and *colorPreview* based on the updated progress.

At this point, you should be able to run your app, and see the color preview change when you move the SeekBars.

*Generate Grid:*

The goal of *generateGrid()* is to generate a 32x32 grid of buttons which you will later use to draw on your LED board. This grid must be generated programmatically since it is unreasonable to add a this many buttons directly to the layout.

The grid will be generated using a TableLayout. In order to add items to a TableLayout, you must first add TableRows to it, and then add items to the TableRow.

1. Begin by creating a loop which will iterate over all the rows that will be in the grid and create/add a new TableRow for each row in the grid. In order to add a TableRow to the grid, use *grid.addView()*. *(For your convenience, constants ROWS and COLS have been defined in this class).*
2. The next step is to add buttons to each TableRow and save those buttons in a 2d array so they can be accessed later. Create another loop which will iterate over all the columns in each row. Create a new Button and at it to the TableRow by calling *addView()*, this time on the TableRow. Add the new button to *Button[ ][ ] buttons* at the current row and column.
3. Now the buttons need to be formatted before we add them to the grid. First, change the color of the button to black, using the method *setBackgroundColor*. Finally, the button must be resized so all the buttons will fit on the screen. This can be done by calling *setLayoutParams()* on the button. In order to create the LayoutParams, add the following line of code. *(Note that the units for the width and height of the buttons are in dp, which means that Android will scale them based on the size of your screen, ensuring the buttons fit on any device).*  
     
    *TableRow.LayoutParams lp = new TableRow.LayoutParams(35, 35);*
4. Lastly, add an OnClickListener to each of the buttons. In the *onClick()* method, call *changeColor(row, col)* which you will implement later. *(Hint: Use the type final int” for the row and column when calling changeColor()).*

Running your app now, should show a large black box on the screen.

**Drawing Functionality:**

Now that everything has been initialized, you can begin to add functionality to your UI. Here you will complete *changeColor()*, *erase()*, and *toggleErasing()*.

*Drawing to the Grid:*

In order to draw to the grid, you will complete *changeColor()*. This method will accomplish two tasks, updating the grid and updating the LED board. These will be updated to the currently selected color or to black if the instance variable *erasing* is set to true.

1. The grid can be updated simply by updating the color of the button at *(r, c)*. Get this button from the array of buttons you created earlier and then update the color.
2. Update the LED board the same way you did in the Coordinate Editor, using the progress from the SeekBars as your RGB values.

*Toggle Erasing Mode:*

The method *toggleErasing()* is called when the eraser button is clicked in the app. This should update the instance variable *erasing* by toggling it between true and false. This method will also add a red border to the eraser button when *erasing* is true. Do this by calling *eraserButton.setActivated()*.

In order to understand why calling *setActivated()* will add a red border around the eraser button, look at the file ***res/drawable/border\_selector.xml***. In the activity layout, this is set as the background of the eraser button. By using *border\_selector.xml* as the background of the button, the background can be changed based on conditions defined in the xml file. In this case *<item android:state\_activated="true">*...*</item>* tells it to only draw the border if the state is activated.

*Erasing the Board:*

Here you will complete the *erase()* method. This should be identical to the Coordinate Editor, except it should also set the color of all the buttons in the grid to black again.

At this point, running your app should allow you to click buttons on the grid and draw both on the grid and the LED board.

**Drag and Draw:**

The app could be considered complete at this point, but clicking each button one at a time is tedious and does not feel like drawing. Fortunately, this problem can be solved by intercepting touch events which occur inside of the grid and then handling those events yourself instead of Android.

*Intercepting Touch Events:*

Touch events can be intercepted by overriding *dispatchTouchEvent()*, which has partially been done for you. In this method, you will determine whether or not the MotionEvent occurs inside of the bounds of the grid. If it does, call *handleGridEvent()* to handle the event yourself, otherwise call *super.dispatchTouchEvent()* to allow Android to handle the event. *(For your convenience, the boundaries of the grid have been calculated in dispatchTouchEvent())*.

*Handling Grid Events:*

Here, you will implement *handleGridEvent()*. This method should call *performClick()* on whichever button the MotionEvent occurs at. Use the ratio of the MotionEvent’s location to its width/height to determine which button to click.

Once these methods are completed, you should be able to draw by clicking and dragging on the grid.

**Demo:**

Upload your Android and Arduino code and draw your TA a picture.

In this lab you will be creating a home monitoring system using an IP camera and your Nexus 7 tablet. Once completed, you will be able to watch the video stream and move the camera from wherever you and your tablet are.

**Learning Objectives:**

* Learn how to use AsyncTasks in Android
* Learn how to use create and handle HTTP Requests in Android
* Practice reading datasheets
* Practice creating Android user interfaces

**Before You Start:**

Download SpyCamShell.zip and the Foscam IP Camera document *(*[*http://www.foscam.es/descarga/ipcam\_cgi\_sdk.pdf*](http://www.foscam.es/descarga/ipcam_cgi_sdk.pdf)*).*

**Displaying Video Stream**

The camera you will use in this lab streams video in the MJPEG format. MJPEG is a simple video format which will send different JPEG images for each frame of the video. In order to play video in this format, the classes MjpegInputStream and MjpegView have been provided for you.

An MjpegView has already been added to the Layout, however, you need to create a new MjpegInputStream and then send that to the MjpegView. Before a new MjpegInputStream can be created, you must first request the stream from the camera.

**Creating an AsyncTask**

Since requesting the stream is a blocking operation, it must be done on its own thread otherwise, the app would become unresponsive until the request was completed. An AsyncTask is one way to accomplish this. The class StreamReader extends AsyncTask<String, Void, MjpegInputStream>. When you run StreamReader, by calling *execute(url)*, the *doInBackground()* method will be called. After *doInBackground()* finishes executing, *onPostExecute()* will be called. This will allow you to request request the video stream, and after the request has been completed, start the MjpegView.

***Before you continue***, be sure you understand how an AsyncTask works, especially the methods *doInBackground()* and *onPostExecute()*.

See <http://developer.android.com/reference/android/os/AsyncTask.html> for more information about AsyncTasks.

In order to complete StreamReader, the first step is to request the stream in *doInBackground()*. Notice how *(String… urls)* is the parameter for *doInBackground()*. This allows an unknown number of Strings to be passed into this function and then stored as an Array called “urls”. In this lab however, you will only need to pass one String to this function at a time.

To request the video stream, first create a new DefaultHttpClient. To request the the url add the following line.

*HttpResponse res = httpclient.execute(new HttpGet(URI.create(url[0])));*

This creates an HTTP GET request with whatever url you provide when you execute StreamReader and saves the response. Next create and return a new MjpegInputStream, passing in *res.getEntity().getContent()* as the parameter. *(Note that you will have to add the necessary try/catch blocks for this code)*.

Finally, the MjpegInputStream that you returned, will be passed as a parameter in *onPostExectute()*. Set this as the source of the MjpegView in SpyCam.java, set the display mode of the MjpegView to ***MjpegView.SIZE\_BEST\_FIT***, and enable showing the FPS of the stream in MjpegView. *(Look at the methods in MjpegView for help).*

**Creating the Request URL:**

The url for the camera stream is ***http://cpre388cam.ece.iastate.edu/videostream.cgi***. However, this url requires additional parameters be provided. In the Foscam IP Camera document, look at the videostream.cgi section and figure out how to modify the url with the following parameters. *(Hint: The url “www.example.com?a=foo&b=bar” sets parameters “a” and “b” to “foo” and “bar”)*.

* Use a username of “admin”
* Use a blank password
* Stream the video at a 320x240 resolution.

After you have created this url, in the *onResume()* method of SpyCam.java, create a new instance of StreamReader and call *execute(yoururl)*.

At this point, running your app should display the camera stream.

**Controlling the Camera**

For this part of the lab, you will be designing a user interface to control the camera. In the Foscam IP Camera document, read the decoder\_controller.cgi and try to understand how the camera is controlled.

Create a new implementation of AsyncTask (similar to StreamReader) which will send the control urls to the Camera. After creating this AsyncTask, build a user interface which meets the following requirements.

* Rotate the camera while a button is pressed
* Rotate the camera a specified number of degrees (you must be able to change this number using your UI)
* Use at least 3 different UI components which add functionality when building your interface.