Eye Pressure Monitor

By: Andrea Levy

Advisor: Tina Smilkstein

Computer Engineering Dept.

California Polytechnic State University, San Luis Obispo, 2018

Table of Contents

List of Figures 3
Abstract 4
Chapter 1 Introduction 5
Eye Pressure Monitor 5
IOIO Board 6
Mobile Application 8
Chapter 2 Interfacing with the Sensor 10
Frequency Counter 10
UART using IOIO board 11
Chapter 3 Eye Pressure System GUI 13
Receiving Measurements 13
Storing Measurements 17
Displaying the Measurements 18
Chapter 4 Conclusion and Future Recommendations 20
Conclusion 20
Future Recommendations 21
Changing the application to work with another protocol 21
Changing the Database to work with Alternate Entities 21
Creating the Contact Lens 21
References 22
Appendix: Firmware Code 24
Appendix: Mobile Application Code 29
Java Files 29
Layout Files 44
Menu Files 49
Values 49
Manifests 50
Gradle Build Files 52
Appendix: Senior Project Analysis 54

List of Figures

Figure 1-1: Project Overview 5
Figure 1-2: IOIO-OTG Board with no connections.....7
Figure 2-1: Frequency Counter Attached to a Function Generator 11
Figure 2-2: Wiring between the IOIO board and the Eye Pressure Monitor 12
Figure 3-1: Database Configuration.....13
Figure 3-2: Measurement with Nothing Connected.....14
Figure 3-3: Measurement with Device Connected 15
Figure 3-4: Toast When Device is Connected.....15
Figure 3-5: Toast When Device is Disconnected 16
Figure 3-6: Raw vs. Time.....18
Figure 3-7: Frequency vs. Time.....19
Figure 3-8: Application Navigation 20

Abstract

The document describes a mobile application that takes information from an attached device which tests eye pressure. The device consists of an IOIO board connected to a custom device that measures the frequency of a given waveform. The device was designed by another student for their senior project, which I am taking over. This device is connected to an IOIO board which is a board designed by a Google employee which works with an android phone in order to create applications that work with embedded systems. The board comes with an API and connects to the phone via a micro-USB. The application takes this data and records it both as a raw number but also as a frequency as measured by the custom device. The application stores the gathered information locally and then displays it on a set of graphs where each graph displays one kind of data calculated from the device.

Chapter 1 Introduction

The Eye Pressure Monitor is an Android application that is designed to connect to a sensor that measures Eye Pressure. In order to do this the Eye Pressure monitor sensor is connected to a IOIO board which transmits the data in the to the phone. This is necessary because the IOIO device connects to the micro-USB of a phone and has an associated API that allows the phone to receive and transmit data to the IOIO board. The diagram in Figure 1-1 shows the data flow between the devices.

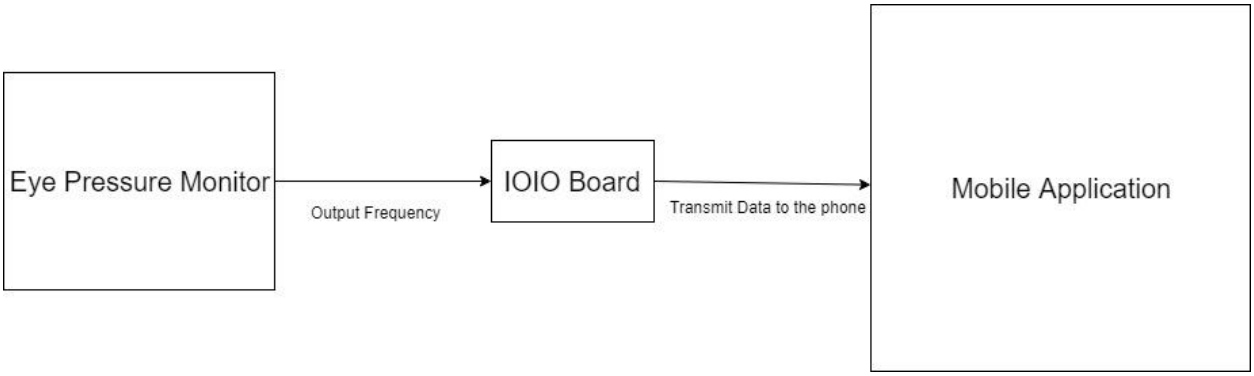

Figure 1-1: Project Overview

Eye Pressure Monitor

The eye pressure monitor is a custom-built board built by Mark Manuel, a former Electrical student from whom I took the project. The board is built to take a frequency of a given waveform and output the number of cycles in 500 microseconds. The original project was a board that was built to measure the frequency of an incoming waveform via an RF device. Since the application did not care whether the RF device was functional or not, my version of the project simply used the calculations provided in his project to calculate the frequency. His project found the frequency of a waveform to be the function below.

$$freq = \frac{clockTicks \times 8}{499\ \mu s}$$

I trusted that his project would accurately calculate the frequency of a given waveform using the following equation. This equation was in the code I received for the board.

The purpose of designing a device that would calculate a basic frequency is to send out a single known frequency to a device and the device would send back another frequency based on the eye pressure. This would be used to track the progress of patients with glaucoma. Eye pressure is an important metric in tracking the progress of glaucoma since fluid buildup in the eye is why Glaucoma can cause vision loss in affected individuals [1]. In the primary open-angle glaucoma, the slow buildup in pressure causes damage to the optic nerve which can lead to vision loss. Although the exact cause of glaucoma is unknown, affected individuals can take medication to reduce eye pressure [1]. One potential use for this device would be to measure the effectiveness of medications at home. Another reason to create a device to measure eye pressure is that many of the tools that are used to measure glaucoma require a specialist to use [2].

Currently, the most accurate method of measuring eye pressure is “Goldmann Applanation Tonometry” which requires a microscope to look into the eye while an orange dye and numbing agent is applied to the eye. In addition to this method, the Tonopen and the iCare are handheld devices that were created for measuring eye pressure. Although both devices are less accurate, they provide a method for testing eye pressure in an office where a proper microscope cannot be stored. However, all these devices require professionals to make sure that they are used in a sanitary way and require that all measurements be stored in a doctor’s office. When the patient controls when and how the data is taken, a doctor or researcher is better able to see how the condition progresses over time.

Since I did not have access to the device that would send and receive these frequencies, I was not actually able to calculate the pressure from the eyeball. The pressure measurement is not included in my project since it was not defined, and I did not possess the contact lens from which the initial frequency would be sent to and a reasonable result would be released. Instead, the raw measurement of the frequency-counter is placed where the pressure measurement is supposed to go. A future designer of the contact lens should be responsible for choosing an appropriate frequency to send out to the eye and to determine what is reasonable to be sent. From these two measurements, the pressure can be calculated and stored.

IOIO Board

The Mobile Application is possible due to an IOIO board. It can be purchased from SparkFun Electronics who also provides some additional documentation for the board. The board can be used for both PC and mobile applications. For PC applications the board is plugged indirectly through the USB port [3]. The board consists of 46 GPIO Pins as well as 3 GND pins, a 5V pin, a 3.3V Pin, and a Vin pin, and an LED. The circled GPIO pins on the diagram can support a 5V input while the rest of the pins are only able to support a 3.3V input. The LED can be used as a normal digital output in most cases. It is used in the bootloader mode to indicate entry into bootloader mode [3]. Figure 1-2 shows the board with no pins connected.

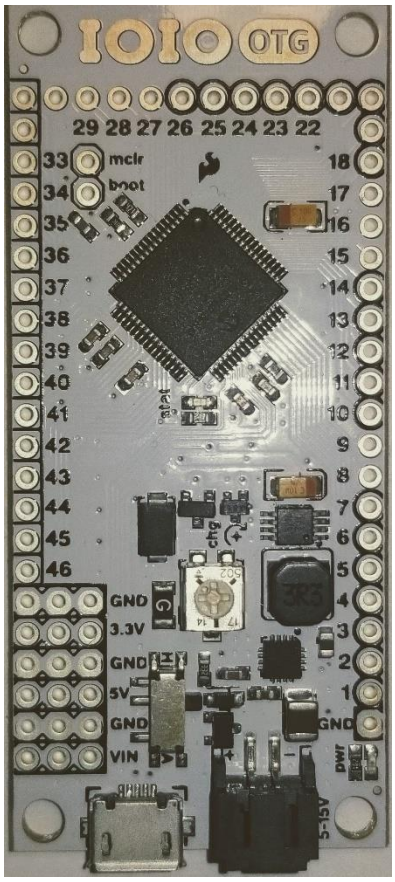

Figure 1-2: IOIO-OTG Board with no connections

The MCLR and BOOT pins are only used to upgrade the firmware [3]. The firmware is loaded onto the board via IOIOdude [4]. The IOIOdude software loads any firmware onto the board when the board is in bootloader mode. In this project, the standard IOIO firmware was used that was compatible with Android Studio 3.

I had to do this at the start of the project to make the board that I had compatible with the current version of Android Studio and Gradle. Having an up to date Gradle built can make sure that the board can take advantages of newer Android features such as the Room Persistence library.

The Charge Current Trimpot changes the amount of current supplied on the VBUS line of the USB. This is used to lower the amount of current supplied for battery supplied operations of the board in order to prevent battery drain. The board also contains a host/device switch that can manually determine whether the board is a device or host. In Android applications, moving this switch will cause the phone to no longer communicate with the device properly. By using a data USB connector, the problem of host mode vs device mode is automatically determined.

Apart from the USB board the board also contains a 2-pin JST female power Jack [3]. This can be used for applications of the board that require a power supply. This is not the case in this application as the device is expected to be powered by the phone. However, this was not the case for development.

For development, I connected both the phone and the device to my computer and used a bridge to connect the devices to enable debugging via Android Studio over the USB. The IOIO bridge connected the IOIO bridge in one port and the phone on another port [5]. The bridge simulated a direct connection between the two devices while powering the device via a laptop. This way I could get information from the logs such as when data was read, how much data was read at once and what kind of data was being read. This is critical to finding out if the bytes that are being read are like those that come directly off the eye pressure monitor and can be read from a program like RealTerm which is commonly used for reading UART directly to a computer. This was also crucial for checking if my code was correct.

In addition to the IOIO bridge, the IOIO board comes with example code and a Java API [4]. The IOIO board supports many different protocols for their boards including PWM, UART, I2C, SPI, and Digital I/O. Each of these protocols contains classes associated with the protocol and provide the necessary operations for reading and writing to these devices. For the UART protocol, there is a Uart class that allows the developer to create an object that contains the Rx pin, the Tx pin, the baud rate, and the parity. This allows the board to be read from accurately with the custom settings. This allows for a developer to use the pins that they know are appropriate and to connect different types of devices to the same device. For the LED, it is implemented as a Digital I/O pin where false means that the light is lit while true means that the light is not lit.

Mobile Application

The mobile application was made specifically for Android phones using the Android Studio platform and Java. This decision was made since the IOIO board was compatible only with Android phones, and iPhones do not have official support for open-source peripherals. The IOIO board also had an API designed for it in Java [4].

Android Studio also had a lot of advantages for development including USB debugging via the Developer Options [6]. The USB debugging tools are unlocked on the phone by first going to the Settings app. For Android phones that have Android 8.0 and above, the user must go into the System menu as well. Then all users select “About phone” and tap “Build number” 7 times. When the user returns to the previous screen, the developer options are unlocked. Then the USB debugging can be enabled through the Developer Options menu [6]. USB debugging allows for the use of Android logging to gracefully report system information to the developer such as what is being read off of the Eye Pressure Monitor. In addition to this it allows a developer to see when an exception is being thrown. The developer can then decide whether to just add it to the logs or to cause a problem for the user. The logs can also tell the developer what the code of an application is doing at a moment in time. This allows for a robust development strategy which gives the application developer a lot of freedom in deciding how a given device.

Android Studio not only is an IDE for Java Development with logging but also has a framework for creating User Interfaces. The User Interfaces are created using XML files that contains all the properties of the User Interface. Android Studio also allows these designs to be seen in a preview so that most of the work of developing the application was in the back end. Once the UI had a mock up, the user could drag and drop the necessary elements into the correct positions. Android

also has several classes and predefined functions that handle the operation of the User Interface and allowed the focus of the application to be on getting and storing the measurements.

The app also relies heavily on System messages to inform the user on the state of the device and of the Application. These messages are known as toasts. They are messages that are displayed in a grey rounded box and are displayed for a certain number of seconds before going away on their own.

Chapter 2 Interfacing with the Sensor

The mobile application uses the IOIO board to receive data while the Frequency Counter calculates the number of clock ticks which it transmits to the phone. This section discusses the process of how this occurs.

Frequency Counter

The Eye Pressure monitor is a board that takes the frequency of a waveform. This is the calculation that I am taking from the board in order to get a frequency measurement of a given waveform. Each of these frequencies is given by a 16-bit number. Thus, the frequency given is taken from the reading of two UART cycles. Since this is a small amount of data, the speed of UART is not an issue for receiving real-time information.

I redesigned the firmware of the board to work to better work with a mobile application. I changed the output of the board to only be the number of ticks collected in 500 microseconds. This is outputted immediately so the phone can read the most output from the board as possible. Although this is a lot of bits to be outputted, this makes sure that only one type of information is outputted to the board so that the bits coming into the board only has one possible interpretation, which is the number of ticks. The only thing the Frequency-Counter does is output the bits of the counter to the board. Figure 2-1 shows the Frequency Counter.

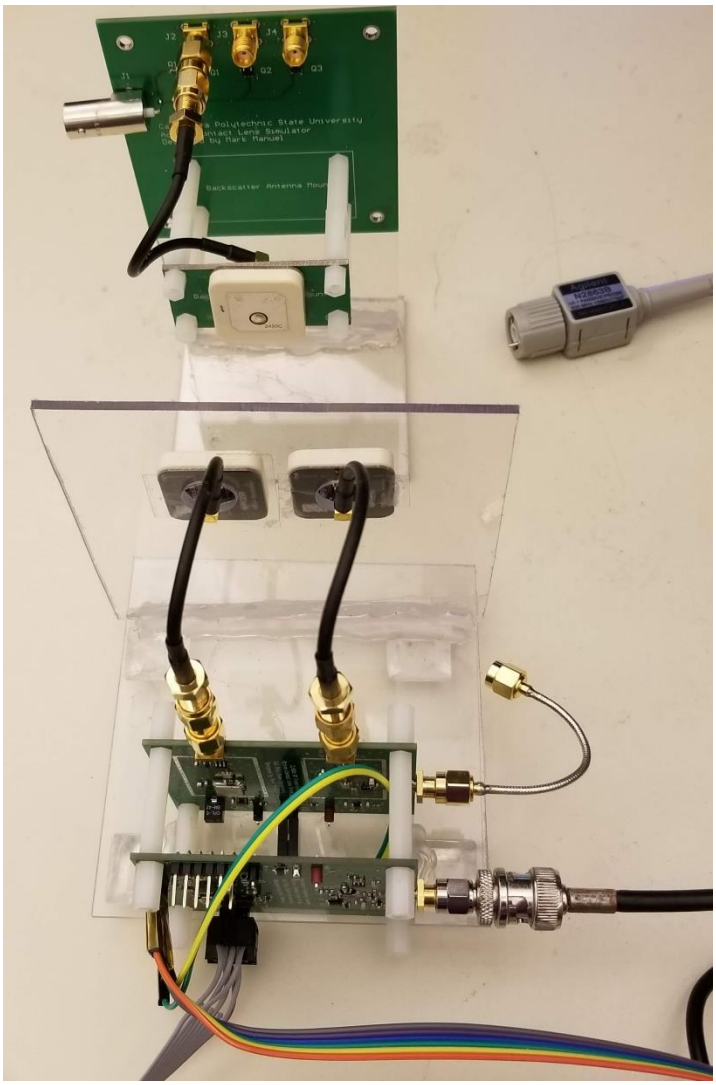

Figure 2-1: Frequency Counter Attached to a Function Generator

UART using IOIO board

An IOIO board is a board designed by a Google employee to work with an Android device with a micro-USB. The board implements any standard protocol to a USB device. The protocol I chose was UART because it was implemented by the person that I inherited the project from. The UART uses a baud rate of 38400. This is hard-coded into both the application and the board's firmware and into the application. Using a previously agreed upon baud rate is good for both the developer of the firmware and the developer so that the measurements can be sent appropriately and received appropriately. This rate was chosen so that the most accurate data can be sent as quickly as possible.

The challenge with implementing UART is that the original protocol was built for a 3.3V pull-up pin [7] but the board needed 5V. Luckily there was a 5V pin on the board that only transmits that value. Any of the GPIO pins can be used for implementing UART if they support 5V inputs [8].

I used pins 2 through 5 because they were part of a long string of 5V tolerant pins. The wiring diagram in Figure 2-2 shows all the pins that were used.

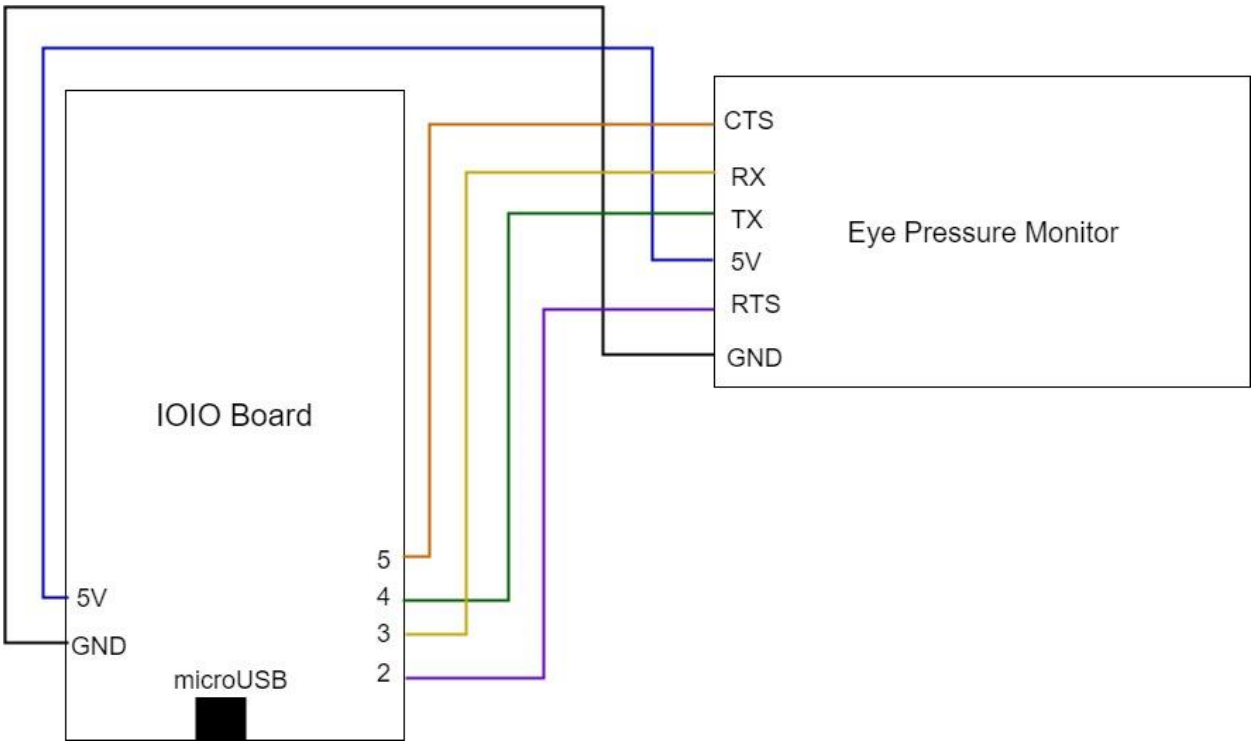

Figure 2-2: Wiring between the IOIO board and the Eye Pressure Monitor

The Board collects the number of ticks of the clock from the board via the Rx pin. This pin is read by the application. The TX pin is also defined in the application, but no data is written to it because the application does not require it. The RTS and CTS pins are not defined in the application but must be placed in pins 2 and 5 respectively. This is not explicitly defined in the software, but I found it essential through testing to place the pins in these positions in order to transmit data to and from the board correctly. The 5V pin is connected to a 5V pin that always outputs 5V when connected to a phone. The GND pin is also always connected to a special GND pin.

The board also contains an LED. The LED is used for when a read is blocking, and the board is powered on. The LED was chosen for hardware debugging as it was the most visible sign on the board that something was off. If all is working correctly the LED should never be visibly lit. However, the connection between the pins and the board can become weak, so the LED is a good indicator of when the connection is not performing as it should. This is also a signal that the board is connected to the phone but not to the frequency counter.

Chapter 3 Eye Pressure System GUI

The Eye Pressure Monitoring Application consists of am activity for reading measurements, Database, and Graphs. The Measurement Reader is in the NewMeasurement activity. An activity in Android refers to a screen on the application and its associated code. The Database stores all the measurements. The Graphs display the data collected over time. Figure 3-1 shows how the data in the database is accessed by various classes. The insertion into the Database is shown with the solid arrows and the getting functions are shown with the dashed lines.

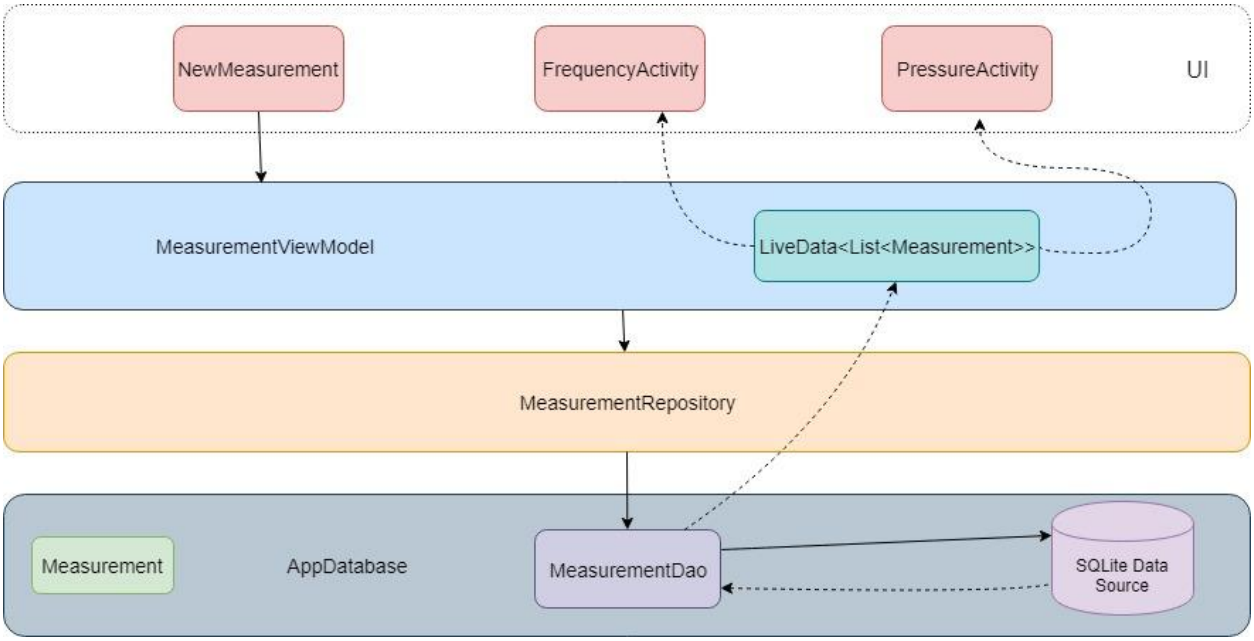

Figure 3-1: Database Configuration

Receiving Measurements

The mobile application receives input from the micro-USB from the IOIO board. The IOIO board has an accompanying API for Android Studio that has an object for receiving data via UART. The Rx pin was set to pin 3 per the wiring diagram. The Tx pin was set to 4 per the wiring diagram. The baud rate was set to the agreed upon 38400. Then the UART was open to receiving data through a function that is run in a loop while the board is connected. The functionality of this loop is set by the API. In order to display the current measurement, the start button is pressed. If the board is not connected to the application, a toast is displayed that explains that since there is no board connected, there must be no current measurement. The toast looks like the screenshot in Figure 3-2.

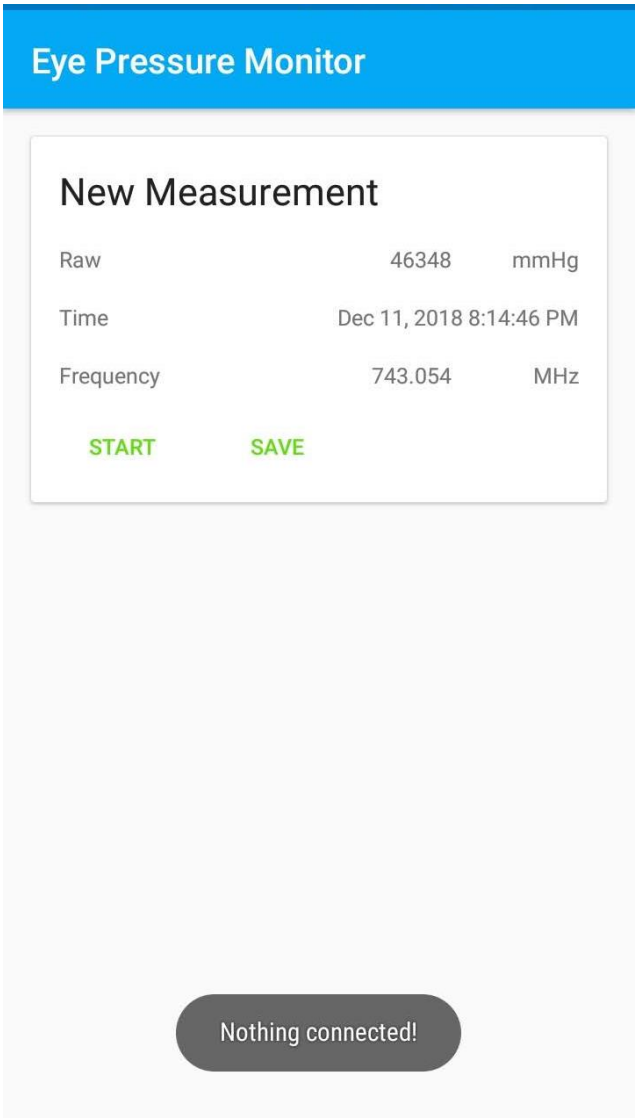

Figure 3-2: Measurement with Nothing Connected

Otherwise, the measurement from the most recent reading is displayed on the screen. This includes the raw data presented as an integer, the frequency, and the current time. The raw value is displayed as a Pressure measurement.

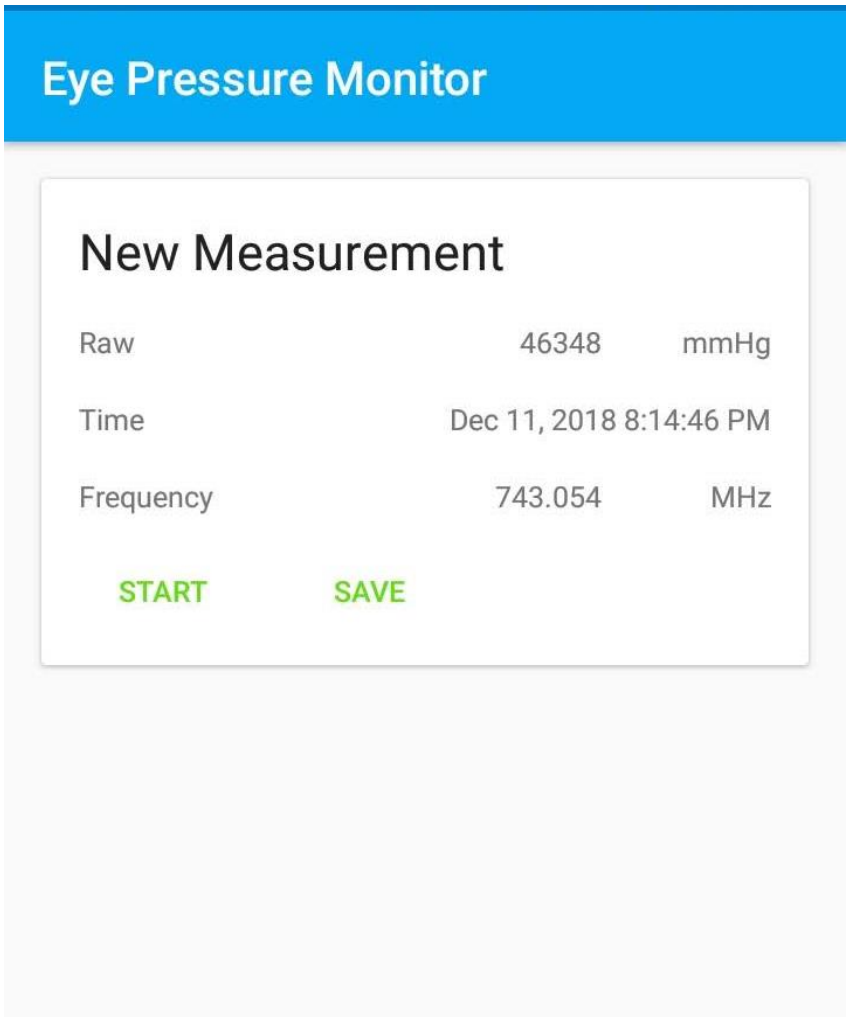

Figure 3-3: Measurement with Device Connected

When the device is connected to the phone another toast message is displayed that shows the device information. Figure 3-4 shows the message from the IOIO board used. It shows the version of IOIOLib that is used as well as the Application Firmware version and the Bootloader firmware. I chose these as they were the most up to date versions that were compatible with the IOIO OTG board [4].

Figure 3-4: Toast When Device is Connected

As when the device is disconnected from the phone another toast message is displayed. The IOIO is disconnected when the user manually disconnects the device or when the user switches to one of the two provided graphs.

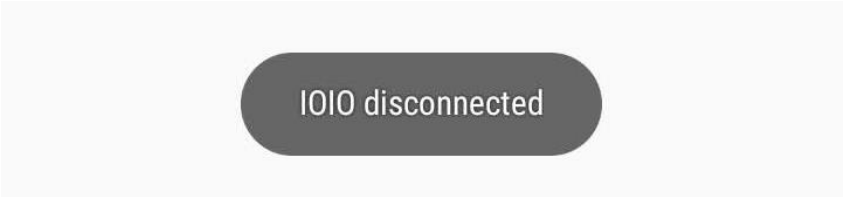

Figure 3-5: Toast When Device is Disconnected

Storing Measurements

The Measurements are stored in a local database using the Room Persistence Library which provides a layer of abstraction over SQLite [9]. This is highly recommended by Google, so SQLite is the database framework used by Android to store user data locally. This was chosen over maintaining a web server for both security and cost concerns. The Room Database consists of several classes and annotations that create the Room Database [10]. In the application, the class AppDatabase creates the Room Database which contains the entities within the database. The entities are representations of the tables in the database as Objects. In this project, the Measurement entity is the only table in our database. This is done because there is only one type of data that we are storing in our table for our database.

In addition to a normal database, we also needed to store converter functions for the Dates because the Room Library does not directly support dates. Instead, a date is stored as the number of milliseconds since the epoch. I created a simple converter class, Converters.java, which the database calls to convert in between java.util.Date and the long used for the system times. The database also contains a Database Access Object or a Dao that manages how the database is accessed [10]. In this application, MeasurementDao.java is the interface that manages the Database Access Object.

The Measurement Database supports getting all the Measurements and inserting a single measurement into the database. Since the Data Access Object is an interface and not an object by itself, it cannot be used directly. Instead, a ViewModel and a Repository [11]. The Repository can be used to manage multiple entities, but in the case of this application, it handles getting all the data after a given date and inserting the data. Insertion into a database is different than getting data in that it must be done asynchronously. This requires that this part of the application be done in a different thread than the rest of the application. This is done so that inserting the information into the database doesn’t interrupt the UI if it takes a long amount of time.

Then the ViewModel is placed on top of the Repository object as in Figure 3-1 [11]. The ViewModel makes sure that getting all the measurements are done at creation and inherits from AndroidViewModel. The ViewModel makes sure that other data sources can be at a later date=. This allows us to take advantage of inheritance from the Room API. Each of the graph displays interacts with only the ViewModel.

Displaying the Measurements

The frequency and Raw data are displayed on a series of graphs. Each of these graphs is its own activity so that as much of the screen as possible can be dedicated to displaying the data. For each graph, the x-axis is time while the y-axis is either the frequency or the raw data. This was done so that the measurements over a day can be viewed along with longer-term results. The graphs were created using an API called GraphView [12]. This API has several types of graphs including line graphs, bar graphs, and point graphs. The API also allows the views to be scalable which allows the user to see as much or as little of the data as they need in order to make an accurate judgment of their measurements. Thus, the amount of data shown can be done in a variable way depending on the data stored. This is done so to keep the memory footprint of the application to a reasonable level. In addition, it creates a more interactive user experience.

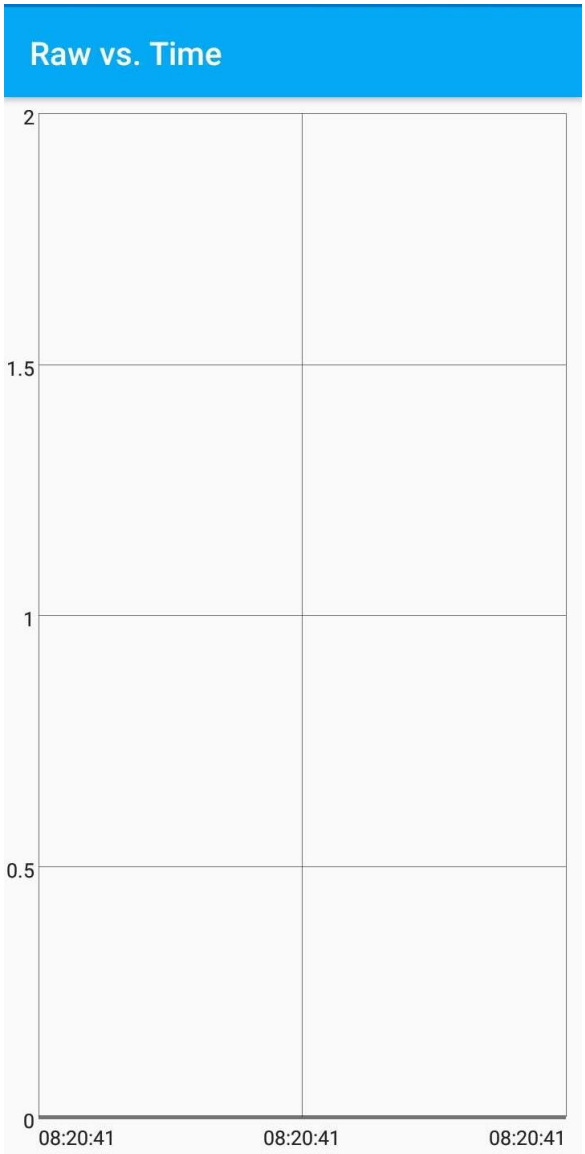

Figure 3-6: Raw vs. Time

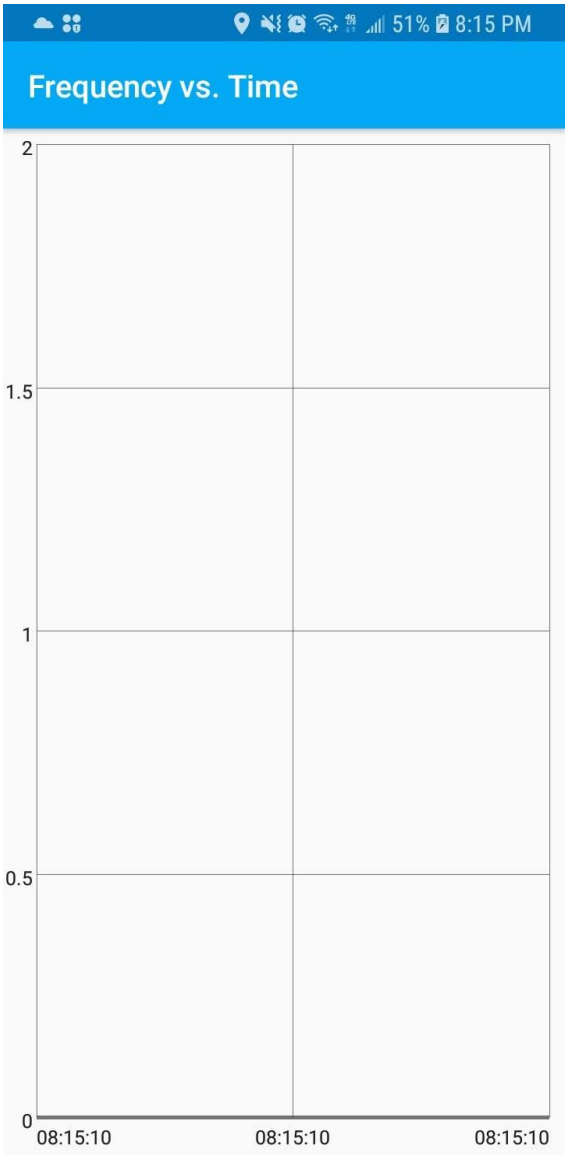

Figure 3-7: Frequency vs. Time

There is a space in the navigation bar for each measurement graph. The navigation showing each graph is shown in Figure 3-8. Each of the graphs has its own activity so that they can take up the entire screen so that the most data can be displayed as possible. This is advantageous for a mobile application where the screen is already small. This is important to be able store data in a way that makes use of the data.

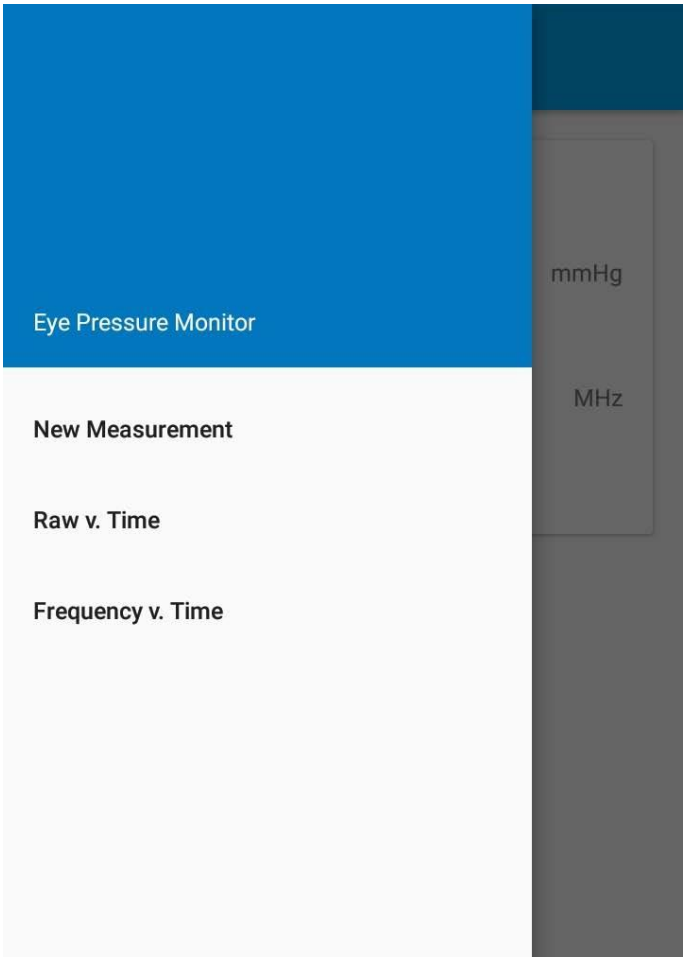

Figure 3-8: Application Navigation

Chapter 4 Conclusion and Future Recommendations

Conclusion

I successfully created an app which communicates with an eye pressure sensor, displays a frequency and raw value, and stores it for later use. The data stored is used to create a graph of each measurement over time. The user is thus able to judge the effectiveness of their treatment program

Future Recommendations

Future Work on the application can be done to make the app work with another protocol, create the contact lens, or change the data type all of these recommendations can be used together in order to create a project.

Changing the application to work with another protocol

Currently, the application works with UART which is an older protocol from getting data from a board. A future developer can adapt the application to use another protocol such as SPI or I2C. If this change were to occur, the developer should reflash the board with the new code and test whether it outputs the correct data. Once that is complete, the uart_ object should be changed to an object that represents the new protocol as described by the API for that protocol. Then the proper read function must be implemented in the loop function that reads from the device into the volatile cur value. This value is volatile to alert the developer that it does not change in the UI thread and must be copied into another value to be worked with in a stable manner.

Changing the Database to work with Alternate Entities

In order to work with a different kind of measurement, the database columns would need to be altered in order to work with new kinds of data. This would be useful if a different device were connected to the IOIO board. The column names would need to be a string that reflects the type of information that would be collected. For debugging purposes, it is useful to keep a raw version of your data in the table in case there is an issue with the calculations that you need to fix later. This can easily be done from another application that has access to the database. If you are using the same code, the Measurement class would need to be changed to reflect these new column names. Then, the version of the database would need to be incremented so that the new table would not interfere with the old data. This is important so that the results of grabbing a measurement are not inconsistent. In addition, the number of graphs may need to be changed if you are adding a column to the database, the x-axis of any graph should always reflect the time at which the data was collected while the y column should always reflect the type of data being taken. This is important to make sure that both the x and y values show up appropriately on the graph.

Creating the Contact Lens

Since the Eye Pressure Monitor is a device that is designed to get the pressure of the eyeball based on two frequencies, there needs to be a contact lens designed to send and receive these different frequencies. This needs to be integrated into the Eye Pressure Monitor. A safe frequency needs to be delivered to the eye as not to damage the eye or to be detectable to the eye. Then the eye pressure must be able to receive this frequency in order to make calculations. Then the application must be informed as to what frequency is being sent to the eye so that it can make an accurate measurement of the pressure.

References

[1] American Optometric Association, "Glaucoma," 2018. [Online]. Available: https://www.aoa.org/patients-and-public/eye-and-vision-problems/glossary-of-eye-and-vision-conditions/glaucoma. [Accessed 1 December 2018].

[2] GEM Clinic - Glaucoma & Eye Management, "About Glaucoma," [Online]. Available: http://gemclinic.ca/about_glaucoma.php. [Accessed 1 December 2018].

[3] SparkFun Electronics, "IOIO-GTG Hookup Line," [Online]. Available: https://learn.sparkfun.com/tutorials/ioio-otg-hookup-guide/ioio-board-overview. [Accessed 1 December 2018].

[4] Y. Ben-Tsvi, "home," 14 October 2016. [Online]. Available: https://github.com/ytai/ioio/wiki. [Accessed 1 December 2018].

[5] Y. Ben-Tsvi, "IOIO Bridge," 26 November 2012. [Online]. Available: https://github.com/ytai/ioio/wiki/IOIO-Bridge. [Accessed 1 December 2018].

[6] "Configure on-device developer options," 21 September 2018. [Online]. Available: https://developer.android.com/studio/debug/dev-options. [Accessed 1 December 2018].

[7] Y. Ben-Tsvi, "UART," 6 January 2015. [Online]. Available: https://github.com/ytai/ioio/wiki/UART. [Accessed 1 December 2018].

[8] Y. Ben-Tsvi, "Getting To Know The Board," 25 January 2013. [Online]. Available: https://github.com/ytai/ioio/wiki/Getting-To-Know-The-Board. [Accessed 1 December 2018].

[9] Google, "Room Persistance Library," 23 October 2018. [Online]. Available: https://developer.android.com/topic/libraries/architecture/room. [Accessed 20 November 2018].

[10] Google, "Save data in a local database using Room," 6 November 2018. [Online]. Available: https://developer.android.com/training/data-storage/room/. [Accessed 20 November 2018].

[11] Google, "Android Room with A View," [Online]. Available: https://codelabs.developers.google.com/codelabs/android-room-with-a-view. [Accessed 20 November 2018].

[12] J. Gehring, "GraphView," 2018. [Online]. Available: http://www.android-graphview.org/. [Accessed 21 November 2018].

[13] M. Manuel, "High-Bandwidth IOP Sensing Backscatter Transceiver," San Luis Obispo, 2018.

Appendix: Firmware Code

Firmware Code available at https://github.com/AndreaLevy238/Frequency-Counter but also below

UART.h

```
/*-----------------------------------------------------------------------------/
* File Name    : UART.h
*
* Project      : Active RFID Wireless Contact Lens Transceiver
* Organization : California Polytechnic State University
* Hardware     : Atmel ATmega328P
* Description  : USART function header for printing to serial terminal
*
* Hardware     : 1. ATMEL ATmega328P Microcontroller
*
* Created      : 4/20/2018
* Engineers    : Mark Manuel and Andrea Levy
*-----------------------------------------------------------------------------*/

#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <util/twi.h>
#include <stdlib.h>                             // Standard C library
#include <string.h>
#include <math.h>

#define BAUD_PRESCALE 103

void usart_init(uint32_t baudin, uint32_t clk_speedin);
void usart_send( uint8_t data );
uint8_t  usart_recv(void);
uint8_t  usart_istheredata(void);
void serial_print(uint8_t low, uint8_t high);
```

UART.c

```
/*-----------------------------------------------------------------------------/
* File Name    : UART.c
*
* Project      : Active RFID Wireless Contact Lens Transceiver
* Organization : California Polytechnic State University
* Hardware     : Atmel ATmega328P
* Description  : USART function lib for printing to serial terminal
*
* Hardware     : 1. ATMEL ATmega328P Microcontroller
*
* Created      : 4/20/2018
* Engineers    : Mark Manuel and Andrea Levy
*-----------------------------------------------------------------------------*/

#include "UART.h"


void usart_init(uint32_t baudin, uint32_t clk_speedin)
```

```
{
	uint32_t ubrr = (clk_speedin/16UL)/baudin-1;
	UBRR0H = (unsigned char)(ubrr>>8);
	UBRR0L = (unsigned char)ubrr;
	/* Enable receiver and transmitter */
	UCSR0B = (1<<RXEN0)|(1<<TXEN0);
	/* Set frame format: 8data, 1stop bit */
	UCSR0C = (1<<USBS0)|(3<<UCSZ00);
	UCSR0A &= ~(1<<U2X0);
}

/*the send function will put 8bits on the trans line. */
void usart_send( uint8_t data )
{
	/* Wait for empty transmit buffer */
	while ( !( UCSR0A & (1<<UDRE0)) );
	/* Put data into buffer, sends the data */
	UDR0 = data;
}

/* the receive data function. Note that this a blocking call
Therefore you may not get control back after this is called
until a much later time. It may be helpful to use the
istheredata() function to check before calling this function
	@return 8bit data packet from sender
*/
uint8_t  usart_recv(void)
{
	/* Wait for data to be received */
	while ( !(UCSR0A & (1<<RXC0)) )
	;
	/* Get and return received data from buffer */
	return UDR0;
}

/* function check to see if there is data to be received
	@return true is there is data ready to be read */
uint8_t  usart_istheredata(void)
{
	 return (UCSR0A & (1<<RXC0));
}

void serial_print(uint8_t low, uint8_t high) {
	usart_send(low);
	usart_send(high);
}
```

main.c

```
/*-----------------------------------------------------------------------------/
* File Name    : main.c
*
* Project      : Active RFID Wireless Contact Lens Transceiver
* Organization : California Polytechnic State University
* Hardware     : Atmel ATmega328P
* Description  : This firmware interfaces the ATMega328 to the
*                Sparkfun IOIO V.2 breakout board and detects
*                the frequency of an external clock signal.
*                Frequency detected is transmitted over SPI to
```

```
*                the IOIO subsystem.
*
* Hardware     : 1. ATMEL ATmega328P Micro controller
*                2. Sparkfun IOIO V.2 Breakout Board
*
* Created      : 4/20/2018
* Engineers    : Mark Manuel and Andrea Levy
*-------------------------------------------------------------------------------*/

#define F_CPU 16000000UL
#include <avr/io.h>
#include <util/delay.h>
#include <avr/interrupt.h>
#include <stdlib.h>                 // Standard C library
#include <string.h>
#include <math.h>
#include "UART.h"

//function prototype
void initTimer0(void);
void initTimer1(void);
void initTimer2(void);

// global variable declaration
volatile int fs_timer_status = 0;   // Sampling Frequency Status
volatile int FC_L = 0;              // Lower Frequency Counter Value
volatile int FC_H = 0;              // Higher Frequency Counter Value
volatile int FC = 0;                // 16 Bit Frequency Counter Value
volatile int samp_dur_status = 0;   // Sampling Duration Status

int main(void)
{
       initTimer0();
       initTimer1();
       initTimer2();
       usart_init(38400, F_CPU);       //Initialize UART bus at 38.4kbaud


       DDRC |= 1<<PC1;                 // Pin B5 output
       DDRC |= 1<<PC0;                 // Pin B0 output
       sei();                          // Enable global interrupts

       while(1)
       {
              /*---------------------- Stop ALL Timers/Counters  ----------------------*/
              TCCR0B = 0x0;    // Stop Sampling Rate Timer
              TCCR1B = 0x0;    // Stop Frequency Counter
              TCCR2B = 0x0;    // Stop Sampling Duration Timer
              /*-----------------------------------------------------------------------*/

              /*--------------------- Reset ALL Timers/Counters ----------------------*/
              TIFR0 = 0x2;     // Reset Sampling Rate Timer
              TIFR1 = 0x2;     // Reset Frequency Counter
              TIFR2 = 0x2;     // Reset Sampling Duration Timer

              /*--------------------------- Begin Measurement -------------------------*/

              TCCR0B = 0x03;          // Start Sampling Rate Timer: SysCLK/64
```

```
            PORTC |= 1<<PC1;        // Turn on C1: total sampling begins
            TCCR1B = 0x07;          // Start Freq. cnt: External CLK on T0 pin, RET
            TCCR2B = 0x04;          // Start Sampling Duration Timer: SysCLK/64
            PORTC |= 1<<PC0;        // Turn on C0: sampling period begins

            while(!fs_timer_status)        // Wait until 1000uS passes
            {
                  while(!samp_dur_status)    // Wait until 500uS passes
                  {
                  }
                  TCCR1B = 0x00;             // Stop frequency counter
                  TCCR2B = 0x00;             // Stop measurement frame timer
            }
            TCCR0B = 0;                    // Stop sampling rate timer
            PORTC &= ~(1 << PC1);          // Turn off C1 after sample is taken
            samp_dur_status = 0;           // Unset sampling duration status
            fs_timer_status = 0;           // Unset sampling period status

            _delay_ms(1);
            serial_print(FC_L, FC_H); // Print Data to Terminal
            FC_L  = 0;
            FC_H = 0;
      }
      /*---------------------------- End Measurement -----------------------------*/

      return 0;

} // end main


void initTimer0(void)
{
      TCCR0A = 0x02;  // timer overflow mode
      TCCR0B = 0x03;  // timer clk = system clk / 64
      OCR0A  = 249;   // Overflow every 1000 Hz
      TIFR0 = 0x02;   // clear previous timer overflow
      TIMSK0 = 0x02;  // timer overflow interrupt enabled
}
void initTimer1(void)
{
      TCCR1A = 0x00;  // timer overflow mode
      TCCR1B = 0x07;  // Start Freq. cnt: External CLK on T0 pin, RET
      OCR1A  = 0;     // Overflow every 1000 Hz
      TIFR1 = 0x02;   // clear previous timer overflow
      TIMSK1 = 0x00;  // timer overflow interrupt enabled
}
void initTimer2(void)
{
      TCCR2A = 0x02;  // timer overflow mode
      TCCR2B = 0x08;  // timer clk = system clk / 64
      OCR2A  = 124;   // Stop recording after 500us
      TIFR2 = 0x02;   // clear previous timer overflow
      TIMSK2 = 0x02;  // timer overflow interrupt enabled
}

ISR(TIMER0_COMPA_vect)
{
      fs_timer_status = 1;    // Kick out of total sample
```

```
}

ISR(TIMER2_COMPA_vect)
{
	samp_dur_status = 1;	// Kick out of sampling period
	FC_H = TCNT1L;
	FC_L = TCNT1H;
	FC = TCNT1;		// Store counts into global var
	TCNT1 = 0x00;
	PORTB &= ~(1 << PB0);	// Turn off B0 when sampling duration closes
}
```

Appendix: Mobile Application Code

Mobile Application Code available at: https://github.com/AndreaLevy238/Eye-Pressure-Monitor

Java Files

AppCompatIOIOActivity.java (from IOIO board developer)

```
package io.github.andrealevy238.eyepressuremonitor;

import android.content.Intent;
import android.os.Bundle;
import android.support.v7.app.AppCompatActivity;

import ioio.lib.util.IOIOLooper;
import ioio.lib.util.IOIOLooperProvider;
import ioio.lib.util.android.IOIOAndroidApplicationHelper;

public abstract class AppCompactIOIOActivity extends AppCompatActivity implements
IOIOLooperProvider {
    private final IOIOAndroidApplicationHelper helper_ = new
IOIOAndroidApplicationHelper(this, this);

    public AppCompactIOIOActivity() {
    }

    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        this.helper_.create();
    }

    protected void onDestroy() {
        this.helper_.destroy();
        super.onDestroy();
    }

    protected void onStart() {
        super.onStart();
        this.helper_.start();
    }

    protected void onStop() {
        this.helper_.stop();
        super.onStop();
    }

    protected void onNewIntent(Intent intent) {
        super.onNewIntent(intent);
        if ((intent.getFlags() & 268435456) != 0) {
            this.helper_.restart();
        }

    }

    protected IOIOLooper createIOIOLooper() {
        throw new RuntimeException("Client must override one of the createIOIOLooper
overloads!");
    }

    public IOIOLooper createIOIOLooper(String connectionType, Object extra) {
        return this.createIOIOLooper();
    }
}
```

AppDatabase.java

```java
package io.github.andrealevy238.eyepressuremonitor;

import android.arch.persistence.room.Database;
import android.arch.persistence.room.Room;
import android.arch.persistence.room.RoomDatabase;
import android.arch.persistence.room.TypeConverters;
import android.content.Context;

@Database(entities = {Measurement.class}, version = 1)
@TypeConverters({Converters.class})
abstract class AppDatabase extends RoomDatabase {
    abstract MeasurementDao measurementDao();

    private static volatile AppDatabase INSTANCE;

    /**
     * @param context the application Context
     * @return an instance of the database based on the RoomDatabase Interface
     */
    static AppDatabase getDatabase(final Context context) {
        if (INSTANCE == null) {
            synchronized (AppDatabase.class) {
                if (INSTANCE == null) {
                    INSTANCE = Room.databaseBuilder(context.getApplicationContext(),
                            AppDatabase.class,
                            "measurement_database"
                    ).build();
                }
            }
        }
        return INSTANCE;
    }
}
```

Converters.java

```java
public class Converters {
    /**
     * @param value the time in millisecconds
     * @return the Date
     */
    @TypeConverter
    public static Date fromTimestamp(Long value) {
        return value == null ? null : new Date(value);
    }

    /**
     * @param date the Java Date
     * @return the time in milliseconds
     */
    @TypeConverter
    public static Long dateToTimestamp(Date date) {
        return date == null ? null : date.getTime();
    }
}
```

Data Converter.java

```java
package io.github.andrealevy238.eyepressuremonitor;

class DataConverter {

    static String toHex(byte[] bytes) {
        StringBuilder sb = new StringBuilder();
```

```
        sb.append("0x");
        for (byte b : bytes) {
            sb.append(String.format("%02X", b));
        }
        return sb.toString();
    }

    /**
     * @param bytes a byte array that specifies some integer
     * @return an iteger represented by the cur
     */
    static int getNum(byte[] bytes) {
        return bytes[0] & 0xFF | (bytes[1] & 0xFF) << 8;
    }

}
```

FrequencyActivity.java

```
package io.github.andrealevy238.eyepressuremonitor;

import android.content.Intent;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.design.widget.NavigationView;
import android.support.v4.app.NavUtils;
import android.support.v4.app.TaskStackBuilder;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.util.Log;
import android.view.MenuItem;

import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.helper.DateAsXAxisLabelFormatter;
import com.jjoe64.graphview.series.DataPoint;
import com.jjoe64.graphview.series.LineGraphSeries;

import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Locale;

public class FrequencyActivity extends AppCompatActivity {
    private MeasurementViewModel model;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_frequency);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        setNav();
        model = ViewModelProviders.of(this).get(MeasurementViewModel.class);
        GraphView pGraph = findViewById(R.id.frequencyGraph);
        DataPoint[] dataPoints = getMeasurements();
        graph(pGraph, dataPoints);
    }

    /**
     * Creates the Graph for this activity which is all the Frequencies in the last 6
months
     *
     * @param graphView  the view for the graph
```

```java
     * @param dataPoints the data in the graph
     */
    private void graph(GraphView graphView, DataPoint[] dataPoints) {
        LineGraphSeries<DataPoint> series = new LineGraphSeries<>(dataPoints);
        if (series.isEmpty()) {
            return;
        }
        graphView.addSeries(series);
        String pattern;
        if (dataPoints.length < 2) {
            pattern = "hh:mm:ss";
            graphView.getGridLabelRenderer().setNumHorizontalLabels(3);
        }
        else {
            pattern = "MM/dd";
        }
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat(pattern, Locale.US);
        DateAsXAxisLabelFormatter d = new
DateAsXAxisLabelFormatter(getApplicationContext(), simpleDateFormat);
        graphView.getGridLabelRenderer().setLabelFormatter(d);
        graphView.getViewport().setScalable(true);
        graphView.getViewport().setScrollable(true);
        graphView.getViewport().setScalableY(true);
        graphView.getViewport().setScrollableY(true);

    }

    /**
     * Sets the navigation drawer for this activity and  the resulting actions
     */
    private void setNav() {
        NavigationView navigationView = findViewById(R.id.nav_view);
        navigationView.setNavigationItemSelectedListener(
                new NavigationView.OnNavigationItemSelectedListener() {
                    @Override
                    public boolean onNavigationItemSelected(@NonNull MenuItem
menuItem) {
                        // set item as selected to persist highlight
                        menuItem.setChecked(true);
                        startNewActivity(menuItem);
                        return true;
                    }
                });
    }

    /**
     * Gets the last 6 months of data from the database
     * @return a list of Data points with all the frequencies and their respective
times
     */
    private DataPoint[] getMeasurements() {
        List<Measurement> measurements = model.getMeasurements().getValue();
        if (measurements == null) {
            Log.e("Null Measurements", "measurements are null");
            DataPoint[] pressures =  new DataPoint[1];
            pressures[0] = new DataPoint(Calendar.getInstance().getTime(), 0);
            return pressures;
        }
        int size = measurements.size();
        DataPoint[] frequencies = new DataPoint[size];
        for (int i = 0; i < size; i++) {
            Measurement m = measurements.get(i);
            frequencies[i] = new DataPoint(m.time, m.frequency);
```

```
        }
        return frequencies;
    }

    /**
     * Starts a new activity based on the item selected
     * @param menuItem the item selected
     */
    private void startNewActivity(MenuItem menuItem) {
        Intent intent = null;
        switch (menuItem.getItemId()) {
            case R.id.nav_new_measurement:
                intent = new Intent(this, NewMeasurement.class);
                break;
            case R.id.nav_frequency:
                intent = new Intent(this, FrequencyActivity.class);
                break;
            case R.id.nav_pressure:
                intent = new Intent(this, PressureActivity.class);
                break;

        }
        if (intent != null) {
            startActivity(intent);
        }
    }


    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            // Respond to the action bar's Up/Home button
            case android.R.id.home:
                Intent upIntent = NavUtils.getParentActivityIntent(this);
                if (upIntent == null) {
                    return false;
                } else if (NavUtils.shouldUpRecreateTask(this, upIntent)) {
                    // This activity is NOT part of this app's task, so create a new
task
                    // when navigating up, with a synthesized back stack.
                    TaskStackBuilder.create(this)
                            // Add all of this activity's parents to the back stack
                            .addNextIntentWithParentStack(upIntent)
                            // Navigate up to the closest parent
                            .startActivities();
                } else {
                    // This activity is part of this app's task, so simply
                    // navigate up to the logical parent activity.
                    NavUtils.navigateUpTo(this, upIntent);
                }
                return true;
        }
        return super.onOptionsItemSelected(item);
    }

}
```

Measurement.java

```
package io.github.andrealevy238.eyepressuremonitor;

import android.arch.persistence.room.ColumnInfo;
import android.arch.persistence.room.Entity;
import android.arch.persistence.room.PrimaryKey;
```

```java
import java.util.Date;

@Entity(tableName = "measurement")
public class Measurement {
    @ColumnInfo(name = "Frequency")
    public double frequency;
    @ColumnInfo(name = "Pressure")
    public int pressure;
    @ColumnInfo(name = "Time")
    public Date time;
    @PrimaryKey(autoGenerate = true)
    @ColumnInfo(name = "mID")
    public int mId;

    Measurement(double frequency, int pressure, Date time) {
        this.frequency = frequency;
        this.pressure = pressure;
        this.time = time;
    }
}
```

MeasurementDao.java

```java
package io.github.andrealevy238.eyepressuremonitor;

import android.arch.lifecycle.LiveData;
import android.arch.persistence.room.Dao;
import android.arch.persistence.room.Insert;
import android.arch.persistence.room.Query;

import java.util.Date;
import java.util.List;

@Dao
public interface MeasurementDao {
    @Query("select * from measurement ORDER BY Time")
    LiveData<List<Measurement>> getAll();

    @Insert
    void insert(Measurement measurement);
}
```

MeasurementRepository.java

```java
package io.github.andrealevy238.eyepressuremonitor;

import android.app.Application;
import android.arch.lifecycle.LiveData;
import android.os.AsyncTask;

import java.util.Date;
import java.util.List;

public class MeasurementRepository {
    private MeasurementDao measurementDao;
    private LiveData<List<Measurement>> allMeasurements;

    /**
     * @param application the application for which the application is for
     */
    MeasurementRepository(Application application) {
        AppDatabase db = AppDatabase.getDatabase(application);
        measurementDao = db.measurementDao();
        allMeasurements = measurementDao.getAll();
    }
```

```java
    /**
     * @return a list of all the measurements
     */
    LiveData<List<Measurement>> getAllMeasurements() {
        return allMeasurements;
    }

    /**
     * Inserts a measurement object into the database
     * @param measurement the item that is being inserted
     */
    public void insert(Measurement measurement) {
        new insertAsyncTask(measurementDao).execute(measurement);
    }

    private static class insertAsyncTask extends AsyncTask<Measurement, Void, Void> {
        private MeasurementDao mAsyncTaskDao;

        insertAsyncTask(MeasurementDao dao) {
            mAsyncTaskDao = dao;
        }

        @Override
        protected Void doInBackground(Measurement... measurements) {
            mAsyncTaskDao.insert(measurements[0]);
            return null;
        }
    }
}
```

MeasurementViewModel.java

```java
package io.github.andrealevy238.eyepressuremonitor;

import android.app.Application;
import android.arch.lifecycle.AndroidViewModel;
import android.arch.lifecycle.LiveData;
import android.support.annotation.NonNull;

import java.util.List;

public class MeasurementViewModel extends AndroidViewModel {
    private MeasurementRepository mRepository;
    private LiveData<List<Measurement>> measurements;

    /**
     * Creates a MeasuementViewModel for the past 6 months worth of data
     *
     * @param application the application for the database
     */
    public MeasurementViewModel(@NonNull Application application) {
        super(application);
        mRepository = new MeasurementRepository(application);
        measurements = mRepository.getAllMeasurements();
    }


    /**
     * Gets the list of Measurements right now
     * @return a list of measurements that is the current state of the database
     */
    public LiveData<List<Measurement>> getMeasurements() {
        return measurements;
```

```
    }

    /**
     * @param m a Measurement  to insert into the database
     */
    public void insert(Measurement m) {
        mRepository.insert(m);
    }
}
```

NewMeasurement.java

```
package io.github.andrealevy238.eyepressuremonitor;

import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.design.widget.NavigationView;
import android.support.v4.view.GravityCompat;
import android.support.v4.widget.DrawerLayout;
import android.support.v7.widget.Toolbar;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

import java.io.IOException;
import java.io.InputStream;
import java.text.DateFormat;
import java.text.DecimalFormat;
import java.util.Date;
import java.util.GregorianCalendar;

import ioio.lib.api.DigitalInput;
import ioio.lib.api.DigitalOutput;
import ioio.lib.api.IOIO;
import ioio.lib.api.Uart;
import ioio.lib.api.exception.ConnectionLostException;
import ioio.lib.util.BaseIOIOLooper;
import ioio.lib.util.IOIOLooper;

import static io.github.andrealevy238.eyepressuremonitor.DataConverter.getNum;
import static io.github.andrealevy238.eyepressuremonitor.DataConverter.toHex;

public class NewMeasurement extends AppCompactIOIOActivity {
    protected static final int rx = 3;
    static final int tx = 4;
    static final int BAUD = 38400;
    protected volatile byte[] cur;
    Button start, save;
    private Date date;
    private int numConnected_ = 0;
    private int ticks;
    private double freq;
    MeasurementViewModel model;
    private DrawerLayout mDrawerLayout;

    /**
     * Gets the frequency in MHz for the number of clock ticks recorded in 500
microseconds
```

```java
     *
     * @param clockTicks the number of clock ticks recorded by the board
     * @return the frequency
     */
    public static double getFrequency(int clockTicks) {
        return (clockTicks * 8.0) / 499;
    }

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_new_measurement);
        freq = -1;
        setNav();
        setDrawer();
        model = ViewModelProviders.of(this).get(MeasurementViewModel.class);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        setStartButton();
        setSave();

    }

    private void setDrawer() {
        mDrawerLayout = findViewById(R.id.drawerLayoutNewMeasurement);
        mDrawerLayout.addDrawerListener(new DrawerLayout.DrawerListener() {
            @Override
            public void onDrawerSlide(@NonNull View drawerView, float slideOffset) {

            }

            @Override
            public void onDrawerOpened(@NonNull View drawerView) {
            }

            @Override
            public void onDrawerClosed(@NonNull View drawerView) {

            }

            @Override
            public void onDrawerStateChanged(int newState) {

            }
        });
    }

    /**
     * Creates the Navigation for this activity
     */
    private void setNav() {
        NavigationView navigationView = findViewById(R.id.nav_view);
        navigationView.setNavigationItemSelectedListener(
                new NavigationView.OnNavigationItemSelectedListener() {
                    @Override
                    public boolean onNavigationItemSelected(@NonNull MenuItem
menuItem) {
                        // set item as selected to persist highlight
                        menuItem.setChecked(true);
                        // close drawer when item is tapped
                        mDrawerLayout.closeDrawers();

                        //open activities for new types of measurements
```

```
                    startNewActivity(menuItem);
                    return true;
                }
            });
}

/**
 * Starts a new activity based on the MenuItem selected
 * @param menuItem the menu item selected
 */
private void startNewActivity(MenuItem menuItem) {
    Intent intent = null;
    switch (menuItem.getItemId()) {
        case R.id.nav_new_measurement:
            intent = new Intent(this, NewMeasurement.class);
            break;
        case R.id.nav_frequency:
            intent = new Intent(this, FrequencyActivity.class);
            break;
        case R.id.nav_pressure:
            intent = new Intent(this, PressureActivity.class);
            break;
    }
    if (intent != null) {
        startActivity(intent);
    }
}


/**
 *  Creates the action for the start button to display the measurement
 */
private void setStartButton() {
    int BUFSIZE = 2;
    cur = new byte[BUFSIZE];
    start = findViewById(R.id.start);
    start.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            if (numConnected_ > 0) {
                byte[] measured = cur;
                ticks = getNum(measured);
                displayPressure(ticks);
                displayTime();
                displayFreq(ticks);
            } else {
                toast("Nothing connected!");
            }
        }
    });
}


@Override
public boolean onCreateOptionsMenu(Menu menu) {
    // Inflate the menu; this adds items to the action bar if it is present.
    return true;
}

@Override
public boolean onOptionsItemSelected(MenuItem item) {
    switch (item.getItemId()) {
        case android.R.id.home:
```

```java
            mDrawerLayout.openDrawer(GravityCompat.START);
            return true;
        }
        return super.onOptionsItemSelected(item);
    }

    /**
     * Saves the currently displayed measurement into the database
     */
    private void setSave() {
        save = findViewById(R.id.save);
        save.setOnClickListener(new View.OnClickListener() {
            @Override
            public void onClick(View v) {
                if (freq < 0) {
                    toast("No measurement gotten! Failed to save");
                } else {
                    Measurement measurement = new Measurement(freq, ticks, date);
                    model.insert(measurement);
                    toast("Saved!");
                }
            }
        });
    }

    /**
     * Displays the current time
     */
    public void displayTime() {
        TextView timeView = findViewById(R.id.time);
        date = new GregorianCalendar().getTime();
        timeView.setText(getDateString());
        timeView.setVisibility(View.VISIBLE);
    }

    /**
     * Displays the raw number that collected from the Rx pin
     * @param raw the number of clock ticks
     */
    public void displayPressure(int raw) {
        TextView textView = findViewById(R.id.pressure);
        textView.setText(String.valueOf(raw));
        textView.setVisibility(View.VISIBLE);
    }

    /**
     * Finds the frequency and displays it
     * @param num the number of clock ticks
     */
    public void displayFreq(int num) {
        TextView textView = findViewById(R.id.frequency);
        freq = getFrequency(num);
        DecimalFormat df = new DecimalFormat("#.###");
        textView.setText(df.format(freq));
        textView.setVisibility(View.VISIBLE);
    }

    /**
     * Formats the date into something that can be displayed
     * @return a Date that can be displayed
     */
    public String getDateString() {
        DateFormat dateFormat = DateFormat.getDateTimeInstance();
```

```java
    return dateFormat.format(date);
}

/**
 * Gets the date displayed
 * @return the Date that is displayed
 */
public Date getDate() {
    return date;
}

/**
 * Creates a message giving the hardware information from the board
 * This is from an example application
 * @param ioio the board connected
 * @param title the Title given to the board
 */
private void showVersions(IOIO ioio, String title) {
    toast(String.format("%s\n" +
                        "IOIOLib: %s\n" +
                        "Application firmware: %s\n" +
                        "Bootloader firmware: %s\n" +
                        "Hardware: %s",
            title,
            ioio.getImplVersion(IOIO.VersionType.IOIOLIB_VER),
            ioio.getImplVersion(IOIO.VersionType.APP_FIRMWARE_VER),
            ioio.getImplVersion(IOIO.VersionType.BOOTLOADER_VER),
            ioio.getImplVersion(IOIO.VersionType.HARDWARE_VER)));
}

/**
 * Displays a toast message with important error or hardware information
 * @param message what is being displayed on the toast
 */
private void toast(final String message) {
    final Context context = this.getApplicationContext();
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            Toast.makeText(context, message, Toast.LENGTH_LONG).show();
        }
    });
}

private void enableUi(boolean enable) {
    // This is slightly trickier than expected to support a multi-IOIO use-case.
    final boolean enabled = enable;
    runOnUiThread(new Runnable() {
        @Override
        public void run() {
            if (enabled) {
                if (numConnected_++ == 0) {
                    Log.d("CONNECT", "First item connected");
                }
            } else {
                if (--numConnected_ == 0) {
                    Log.e("DISCONNECT", "Not connected");
                }
            }
        }
    });

}
```

```java
    @Override
    protected IOIOLooper createIOIOLooper() {
        return new Looper();
    }

    class Looper extends BaseIOIOLooper {
        private Uart uart_;
        private InputStream in_;
        private DigitalOutput led;
        @Override
        protected void setup() throws ConnectionLostException {
            showVersions(ioio_, "IOIO connected!");
            uart_ = ioio_.openUart(new DigitalInput.Spec(rx), new
DigitalOutput.Spec(tx), BAUD, Uart.Parity.NONE, Uart.StopBits.ONE);
            led = ioio_.openDigitalOutput(IOIO.LED_PIN);
            enableUi(true);
            led.write(true);
            in_ = uart_.getInputStream();
            Log.v("SetupUART", "sleep complete");
        }

        @Override
        public void loop() {
            Log.v("UART", "new measurement starting");
            if (uart_ != null) {
                try {
                    readUART();
                } catch (ConnectionLostException | InterruptedException e) {
                    Log.e("UART_Exception", e.getMessage());
                }
            }
        }

        /**
         * Reads information from the UART Rx pin
         * Displays an LED if the read is blocking but the board is connected
         * @throws ConnectionLostException when the board looses connection with the
boad
         */
        private void readUART() throws ConnectionLostException, InterruptedException {
            byte[] raw = new byte[10];
            int i = -1;
            try {
                led.write(false);
                i = in_.read(raw);
                led.write(true);
                Log.d("UART", "read complete, read " + String.valueOf(i) + " bytes");
            } catch (IOException e) {
                Log.e("UART_IO", e.getMessage());
                raw = null;
            }
            if (raw != null && i > 1) {
                Log.d("UART-read", toHex(raw));
                int b0 = i % 2;
                cur[0] = raw[b0];
                cur[1] = raw[b0 + 1];
            }
            Thread.sleep(10);
        }

        @Override
        public void disconnected() {
```

```
            enableUi(false);
            uart_.close();
            toast("IOIO disconnected");
        }

        @Override
        public void incompatible() {
            showVersions(ioio_, "Incompatible firmware version!");
        }

    }
}
```

PressureActivity.java

```
package io.github.andrealevy238.eyepressuremonitor;

import android.content.Intent;
import android.os.Bundle;
import android.support.annotation.NonNull;
import android.support.design.widget.NavigationView;
import android.support.v4.app.NavUtils;
import android.support.v4.app.TaskStackBuilder;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.Toolbar;
import android.util.Log;
import android.view.MenuItem;

import com.jjoe64.graphview.GraphView;
import com.jjoe64.graphview.helper.DateAsXAxisLabelFormatter;
import com.jjoe64.graphview.series.DataPoint;
import com.jjoe64.graphview.series.LineGraphSeries;

import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.List;
import java.util.Locale;

public class PressureActivity extends AppCompatActivity {
    private MeasurementViewModel model;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_pressure);
        Toolbar toolbar = findViewById(R.id.toolbar);
        setSupportActionBar(toolbar);
        setNav();
        model = ViewModelProviders.of(this).get(MeasurementViewModel.class);
        GraphView pGraph = findViewById(R.id.pressureGraph);
        DataPoint[] data = getMeasurements();
        graph(pGraph, data);
    }

    /**
     * Creates the Graph for this activity which is all the raw measurements in the
last 6 months
     *
     * @param graphView  the view for the graph
     * @param dataPoints the data in the graph
     */
    private void graph(GraphView graphView, DataPoint[] dataPoints) {
        LineGraphSeries<DataPoint> series = new LineGraphSeries<>(dataPoints);
```

```
        String pattern;
        if (series.isEmpty()) {
            return;
        }
        graphView.addSeries(series);
        if (dataPoints.length < 2) {
            pattern = "hh:mm:ss";
            graphView.getGridLabelRenderer().setNumHorizontalLabels(3);
        }
        else {
            pattern = "MM/dd";
        }
        SimpleDateFormat simpleDateFormat = new SimpleDateFormat(pattern, Locale.US);
        DateAsXAxisLabelFormatter d = new
DateAsXAxisLabelFormatter(getApplicationContext(), simpleDateFormat);
        graphView.getGridLabelRenderer().setLabelFormatter(d);
        graphView.getViewport().setScalable(true);
        graphView.getViewport().setScrollable(true);
        graphView.getViewport().setScalableY(true);
        graphView.getViewport().setScrollableY(true);
    }

    /**
     * @return a list of all the raw measurements from the last 6 months
     */
    private DataPoint[] getMeasurements() {
        List<Measurement> measurements = model.getMeasurements().getValue();
        if (measurements == null) {
            Log.e("Null Measurements", "measurements are null");
            DataPoint[] pressures =  new DataPoint[1];
            pressures[0] = new DataPoint(Calendar.getInstance().getTime(), 0);
            return pressures;
        }
        int size = measurements.size();
        DataPoint[] pressures = new DataPoint[size];
        for (int i = 0; i < size; i++) {
            Measurement m = measurements.get(i);
            pressures[i] = new DataPoint(m.time, m.pressure);
        }
        return pressures;
    }

    /**
     * Sets the navigation between activities
     */
    private void setNav() {
        NavigationView navigationView = findViewById(R.id.nav_view);
        navigationView.setNavigationItemSelectedListener(
                new NavigationView.OnNavigationItemSelectedListener() {
                    @Override
                    public boolean onNavigationItemSelected(@NonNull MenuItem
menuItem) {
                        // set item as selected to persist highlight
                        menuItem.setChecked(true);


                        // Add code here to update the UI based on the item selected
                        // For example, swap UI fragments here
                        startNewActivity(menuItem);
                        return true;
                    }
                });
    }
```

```java
    /**
     * Starts a new activity based on the MenuItem selected
     * @param menuItem the menu item selected
     */
    private void startNewActivity(MenuItem menuItem) {
        Intent intent = null;
        switch (menuItem.getItemId()) {
            case R.id.nav_new_measurement:
                intent = new Intent(this, NewMeasurement.class);
                break;
            case R.id.nav_frequency:
                intent = new Intent(this, FrequencyActivity.class);
                break;
            case R.id.nav_pressure:
                intent = new Intent(this, PressureActivity.class);
                break;
        }
        if (intent != null) {
            startActivity(intent);
        }
    }

    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
            // Respond to the action bar's Up/Home button
            case android.R.id.home:
                Intent upIntent = NavUtils.getParentActivityIntent(this);
                if (upIntent == null) {
                    return false;
                } else if (NavUtils.shouldUpRecreateTask(this, upIntent)) {
                    // This activity is NOT part of this app's task, so create a new
task
                    // when navigating up, with a synthesized back stack.
                    TaskStackBuilder.create(this)
                            // Add all of this activity's parents to the back stack
                            .addNextIntentWithParentStack(upIntent)
                            // Navigate up to the closest parent
                            .startActivities();
                } else {
                    // This activity is part of this app's task, so simply
                    // navigate up to the logical parent activity.
                    NavUtils.navigateUpTo(this, upIntent);
                }
                return true;
        }
        return super.onOptionsItemSelected(item);
    }

}
```

Layout Files

activity_frequency.xml

```xml
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/drawerLayoutFrequency"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".FrequencyActivity">
```

```
    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <include layout="@layout/content_frequency" />

    </FrameLayout>

    <android.support.design.widget.NavigationView
        android:id="@+id/nav_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:fitsSystemWindows="true"
        app:headerLayout="@layout/nav_header"
        app:menu="@menu/drawer_view" />

</android.support.v4.widget.DrawerLayout>
```

activity_pressure.xml

```
<?xml version="1.0" encoding="utf-8"?>
<android.support.v4.widget.DrawerLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:id="@+id/drawerLayoutPressure"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".PressureActivity">

    <FrameLayout
        android:layout_width="match_parent"
        android:layout_height="match_parent">

        <include layout="@layout/content_pressure" />

    </FrameLayout>

    <android.support.design.widget.NavigationView
        android:id="@+id/nav_view"
        android:layout_width="wrap_content"
        android:layout_height="match_parent"
        android:layout_gravity="start"
        android:fitsSystemWindows="true"
        app:headerLayout="@layout/nav_header"
        app:menu="@menu/drawer_view" />

</android.support.v4.widget.DrawerLayout>
```

content_frequency.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context=".FrequencyActivity"
    tools:showIn="@layout/activity_frequency">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
```

```
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay" />

    </android.support.design.widget.AppBarLayout>

    <com.jjoe64.graphview.GraphView
        android:id="@+id/frequencyGraph"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />
</LinearLayout>
```

content_new_measurement.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context=".AppCompactIOIOActivity"
    android:orientation="vertical"
    tools:showIn="@layout/activity_new_measurement">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay" />

    </android.support.design.widget.AppBarLayout>

    <android.support.v7.widget.CardView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_margin="@dimen/fab_margin"
        android:layout_marginStart="16dp"
        android:layout_marginTop="16dp">

        <LinearLayout
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            android:layout_margin="@dimen/row_margin"
            android:orientation="vertical">

            <TextView
                android:id="@+id/MetricsTitle"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_margin="@dimen/row_margin"
                android:text="@string/new_measurement"
```

```
            android:textAppearance="@style/Base.TextAppearance.AppCompat.Headline"
/>

        <GridLayout
            android:id="@+id/tableLayout"
            android:layout_width="match_parent"
            android:layout_height="wrap_content"
            app:layout_constraintStart_toStartOf="parent"
            app:layout_constraintTop_toTopOf="parent">

            <TextView
                android:id="@+id/row0col0"
                android:layout_row="1"
                android:layout_column="0"
                android:layout_margin="@dimen/row_margin"
                android:text="@string/pressure" />

            <TextView
                android:id="@+id/pressure"
                android:layout_row="1"
                android:layout_column="1"
                android:layout_gravity="end"
                android:layout_margin="@dimen/row_margin"
                android:layout_marginEnd="5sp"
                android:minWidth="150dp"
                android:textAlignment="viewEnd" />

            <TextView
                android:id="@+id/row0col2"
                android:layout_row="1"
                android:layout_column="2"
                android:layout_gravity="end"
                android:layout_margin="@dimen/row_margin"
                android:text="@string/mmhg" />

            <TextView
                android:id="@+id/row1col0"
                android:layout_row="2"
                android:layout_column="0"
                android:layout_margin="@dimen/row_margin"
                android:text="@string/time" />

            <TextView
                android:id="@+id/time"
                android:layout_row="2"
                android:layout_column="1"
                android:layout_columnSpan="2"
                android:layout_gravity="end"
                android:layout_margin="@dimen/row_margin"
                android:maxLines="1"
                android:minWidth="135dp"
                android:visibility="invisible" />

            <TextView
                android:id="@+id/row2col0"
                android:layout_row="3"
                android:layout_column="0"
                android:layout_margin="@dimen/row_margin"
                android:text="@string/frequency" />

            <TextView
                android:id="@+id/frequency"
                android:layout_row="3"
```

```
                    android:layout_column="1"
                    android:layout_gravity="end"
                    android:layout_margin="@dimen/row_margin"
                    android:layout_marginEnd="5sp"
                    android:text=""
                    android:visibility="invisible" />

                <TextView
                    android:id="@+id/row2col2"
                    android:layout_row="3"
                    android:layout_column="2"
                    android:layout_gravity="end"
                    android:layout_margin="@dimen/row_margin"
                    android:text="@string/mhz" />
            </GridLayout>

            <LinearLayout
                android:layout_width="match_parent"
                android:layout_height="wrap_content">

                <Button
                    android:id="@+id/start"
                    style="@style/Base.Widget.AppCompat.Button.Borderless.Colored"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="@string/start" />

                <Button
                    android:id="@+id/save"
                    style="@style/Base.Widget.AppCompat.Button.Borderless.Colored"
                    android:layout_width="wrap_content"
                    android:layout_height="wrap_content"
                    android:text="@string/save" />
            </LinearLayout>
        </LinearLayout>

    </android.support.v7.widget.CardView>
</LinearLayout>
```

content_pressure.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:orientation="vertical"
    app:layout_behavior="@string/appbar_scrolling_view_behavior"
    tools:context=".PressureActivity"
    tools:showIn="@layout/activity_pressure">

    <android.support.design.widget.AppBarLayout
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:theme="@style/AppTheme.AppBarOverlay">

        <android.support.v7.widget.Toolbar
            android:id="@+id/toolbar"
            android:layout_width="match_parent"
            android:layout_height="?attr/actionBarSize"
            android:background="?attr/colorPrimary"
            app:popupTheme="@style/AppTheme.PopupOverlay" />
```

```
    </android.support.design.widget.AppBarLayout>

    <com.jjoe64.graphview.GraphView
        android:id="@+id/pressureGraph"
        android:layout_width="match_parent"
        android:layout_height="match_parent" />

</LinearLayout>
```

nav_header.xml

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="match_parent"
    android:layout_height="192dp"
    android:background="?attr/colorPrimaryDark"
    android:gravity="bottom"
    android:orientation="vertical"
    android:padding="16dp"
    android:theme="@style/ThemeOverlay.AppCompat.Dark">

    <TextView
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:text="@string/eye_pressure_monitor"
        android:textAppearance="@style/TextAppearance.AppCompat.Body1" />

</LinearLayout>
```

Menu Files

drawer_view.xml

```
<?xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">
    <group android:checkableBehavior="single">
        <item
            android:id="@+id/nav_new_measurement"
            android:title="@string/new_measurement" />
        <item
            android:id="@+id/nav_pressure"
            android:title="@string/pressure_vs_time" />
        <item
            android:id="@+id/nav_frequency"
            android:title="@string/frequency_vs_time" />

    </group>
</menu>
```

Values

colors.xml

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="colorPrimary">#03a9f4</color>
    <color name="colorPrimaryDark">#0277bd</color>
    <color name="colorAccent">#64dd17</color>
</resources>
```

dimens.xml

```
<resources>
    <dimen name="fab_margin">16dp</dimen>
    <dimen name="row_margin">8dp</dimen>
    <!-- Default screen margins, per the Android Design guidelines. -->
    <dimen name="activity_horizontal_margin">16dp</dimen>
    <dimen name="activity_vertical_margin">16dp</dimen>
    <dimen name="nav_header_vertical_spacing">8dp</dimen>
```

```
    <dimen name="nav_header_height">176dp</dimen>
</resources>
```

strings.xml

```
<resources>
    <string name="app_name">Eye Pressure Monitor</string>
    <string name="action_settings">Settings</string>
    <string name="mmhg">mmHg</string>
    <string name="time">Time</string>
    <string name="mhz">MHz</string>
    <string name="start">start</string>
    <string name="save">save</string>
    <string name="frequency">Frequency</string>
    <string name="pressure">Raw</string>
    <string name="new_measurement">New Measurement</string>
    <string name="history">History</string>
    <string name="title_activity_history">History</string>
    <string name="pressure_vs_time">Raw v. Time</string>
    <string name="frequency_vs_time">Frequency v. Time</string>
    <string name="title_activity_pressure">Raw vs. Time </string>
    <string name="title_activity_frequency">Frequency vs. Time </string>
    <string name="eye_pressure_monitor">Eye Pressure Monitor</string>
    <string name="navigation_drawer_open">Open navigation drawer</string>
    <string name="navigation_drawer_close">Close navigation drawer</string>
    <string name="nav_header_title">Android Studio</string>
    <string name="nav_header_subtitle">android.studio@android.com</string>
    <string name="nav_header_desc">Navigation header</string>
</resources>
```

styles.xml

```
<resources>

    <!-- Base application theme. -->
    <style name="AppTheme" parent="Theme.AppCompat.Light.NoActionBar">
        <!-- Customize your theme here. -->
        <item name="colorPrimary">@color/colorPrimary</item>
        <item name="colorPrimaryDark">@color/colorPrimaryDark</item>
        <item name="colorAccent">@color/colorAccent</item>
    </style>
    <style name="AppTheme.AppBarOverlay"
parent="ThemeOverlay.AppCompat.Dark.ActionBar" />
    <style name="AppTheme.PopupOverlay" parent="ThemeOverlay.AppCompat.Light" />

</resources>
```

Manifests

AndroidManifest.xml

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="io.github.andrealevy238.eyepressuremonitor">

    <uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE" />
    <uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE" />

    <application
        android:allowBackup="true"
        android:fullBackupContent="@xml/backup_descriptor"
        android:icon="@mipmap/ic_launcher"
        android:label="@string/app_name"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:theme="@style/AppTheme">
        <uses-library
```

```
            android:name="com.android.future.usb.accessory"
            android:required="false" />

        <activity
            android:name=".NewMeasurement"
            android:label="@string/app_name"
            android:theme="@style/AppTheme">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />

                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.intent.action.VIEW" />
            </intent-filter>
            <intent-filter>
                <action
android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED" />
            </intent-filter>
            <intent-filter>
                <action android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
/>
            </intent-filter>

            <meta-data
                android:name="android.hardware.usb.action.USB_ACCESSORY_ATTACHED"
                android:resource="@xml/accessory_filter" />
            <meta-data
                android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
                android:resource="@xml/device_filter" />
        </activity>
        <activity
            android:name=".PressureActivity"
            android:label="@string/title_activity_pressure"
            android:parentActivityName=".NewMeasurement"
            android:theme="@style/AppTheme">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"

android:value="io.github.andrealevy238.eyepressuremonitor.NewMeasurement" />
        </activity>
        <activity
            android:name=".FrequencyActivity"
            android:label="@string/title_activity_frequency"
            android:parentActivityName=".NewMeasurement"
            android:theme="@style/AppTheme">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"

android:value="io.github.andrealevy238.eyepressuremonitor.NewMeasurement" />
        </activity>
        <activity
            android:name=".FrequencyToday"
            android:label="@string/title_activity_frequency_today"
            android:parentActivityName=".NewMeasurement"
            android:theme="@style/AppTheme">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"

android:value="io.github.andrealevy238.eyepressuremonitor.NewMeasurement" />
        </activity>
        <activity
            android:name=".PressureToday"
```

```
            android:label="@string/title_activity_pressure_today"
            android:parentActivityName=".NewMeasurement"
            android:theme="@style/AppTheme">
            <meta-data
                android:name="android.support.PARENT_ACTIVITY"

android:value="io.github.andrealevy238.eyepressuremonitor.NewMeasurement" />
        </activity>
    </application>

</manifest>
```

Gradle Build Files

build.gradle (Project: EyePressureMonitor)

```
// Top-level build file where you can add configuration options common to all sub-
projects/modules.

buildscript {

    repositories {
        google()
        jcenter()
    }
    dependencies {
        classpath 'com.android.tools.build:gradle:3.2.1'
        // NOTE: Do not place your application dependencies here; they belong
        // in the individual module build.gradle files
    }
}

allprojects {
    repositories {
        google()
        jcenter()
    }
}

task clean(type: Delete) {
    delete rootProject.buildDir
}

ext {
    roomVersion = '1.1.1'
    archLifecycleVersion = '1.1.1'
}
```

build.gradle (Module app)

```
apply plugin: 'com.android.application'

android {
    compileSdkVersion 27
    defaultConfig {
        applicationId "io.github.andrealevy238.eyepressuremonitor"
        minSdkVersion 21
        targetSdkVersion 27
        versionCode 1
        versionName "1.0"
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
        // used by Room, to test migrations
        javaCompileOptions {
            annotationProcessorOptions {
                arguments = ["room.schemaLocation":
                                    "$projectDir/schemas".toString()]
```

```
            }
        }
    }
    // used by Room, to test migrations
    sourceSets {
        androidTest.assets.srcDirs +=
                files("$projectDir/schemas".toString())
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles getDefaultProguardFile('proguard-android.txt'), 'proguard-
rules.pro'
        }
    }
}

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])
    implementation 'com.android.support:appcompat-v7:27.1.1'
    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    implementation 'com.android.support:design:27.1.1'
    implementation 'com.android.support:cardview-v7:27.1.1'
    // IOIO Components
    api 'com.github.ytai.ioio:IOIOLibAndroid:5.07'
    api 'com.github.ytai.ioio:IOIOLibAndroidBluetooth:5.07'
    api 'com.github.ytai.ioio:IOIOLibAndroidAccessory:5.07'
    api 'com.github.ytai.ioio:IOIOLibAndroidDevice:5.07'

    // Room components
    implementation "android.arch.persistence.room:runtime:$rootProject.roomVersion"
    annotationProcessor
"android.arch.persistence.room:compiler:$rootProject.roomVersion"
    androidTestImplementation
"android.arch.persistence.room:testing:$rootProject.roomVersion"

    // Visualization Components
    implementation 'com.jjoe64:graphview:4.2.2'

// Lifecycle components
    implementation
"android.arch.lifecycle:extensions:$rootProject.archLifecycleVersion"
    annotationProcessor
"android.arch.lifecycle:compiler:$rootProject.archLifecycleVersion"
    androidTestImplementation 'androidx.test:core:1.0.0'
    androidTestImplementation 'androidx.test:runner:1.1.0'
    androidTestImplementation 'androidx.test:rules:1.1.0'
    androidTestImplementation "android.arch.persistence.room:testing:1.1.1"
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.1.0'
    testImplementation 'junit:junit:4.12'
}
```

Appendix: Senior Project Analysis

- **Summary of Functional Requirements**: Describe the overall capabilities of functions of your project or design. Describe what your project does. (Do not describe how you designed it.)
  The project is a device that measures the interocular pressure of the eye and sends that data to a mobile application. The mobile application stores the data.
- **Primary Constraints**: Describe significant challenges or difficulties associated with your project or implementation. For example, what were limiting factors or other issues that impacted your approach? What made your project difficult? What parameters or specifications limited your options or directed your approach?
  The application was limited by the fact that only Android supports attaching a custom-built embedded device with an API. Apple does not officially support this ability. This constrained the application to Android and its accompanying development platform, Android Studio.
- **Economic**
  - Original estimated cost of component parts (as of the start of your project)
    - $140 [13]
  - Actual final cost of component parts (at the end of your project)
    - $50 [13]
  - Attach a final bill of materials for all components
    - IOIO-GTG Board: $40
    - Micro-USB to USB: $4
    - IOP sensor: Free
    - Wires: $6
  - Additional equipment costs (any equipment needed for development?)
  - Original estimated development time (as of the start of your project)
    15 weeks
  - Actual development time (at the end of your project)
    15 weeks
- **If manufactured on a commercial basis:**
  - Estimated number of devices to be sold per year
    Number of glaucoma patients diagnosed each year which is 27,000 people [13]
  - Estimated manufacturing cost for each device
    - Mobile Application: Free
    - Device: $250
  - Estimated purchase price for each device
    - Mobile Application:
    - Device: $500
  - Estimated profit per year
    - Mobile Application: None
    - Device: 6.75 million [13]
  - Estimated cost for user to operate device, per unit time (specify time interval)
    - The estimated cost to operate the device would be the estimated cost of operating a mobile phone. This can be highly variable due to the chosen smartphone and the cost of electricity in the area that you are operating it in.
- **Environmental**: Describe any environmental impact associated with manufacturing or use.
  Since the mobile application is software, it does not have any physical manufactured parts. This means that the Environmental impact of the app is minimal since the user is assumed to already

possess an Android smartphone. For the device itself, the environmental impact is much greater. In order to manufacture 100,000 devices, 11.5 Million pounds of CO_2 are released into the atmosphere and 12,000 liters of Fresh water are used.

- **Manufacturability**: Describe any issues or challenges associated with manufacturing.
  In order to make the device useful for the consumer, the device and the IOIO board would need to be combined in some way that would allow the manufacturer to have a single package. This would require knowledge of the IOIO firmware and the device itself.
- **Sustainability**
  - Describe any issues or challenges associated with maintaining the completed device or system.
    A user who uses the application over a long period of time may find that the data stored on the database may be too much for a phone. This could be a problem for older model phones with significantly less memory than a higher end phone.
  - Describe how the project impacts the sustainable use of resources.
    The development of the chip to get the interocular pressure would rely heavily on the natural resources to create the chip.