ECE 558 - EMBEDDED SYSTEMS PROGRAMMING
PROJECT 3

AndroidThings - Software

RPI3 Application + User Application + Firebase Database

Prepared By

KIYASUL ARIF ABDUL MAJEETH

Revised By

ROY KRAVITZ

NOTE: SINCE ANDROID AND ANDROID STUDIO ARE CONSTANTLY BEING UPDATED AND REVISED WE CANNOT GUARANTEE THAT THE DETAILED STEP-BY-STEP INSTRUCTIONS AND SCREEN SHOTS PROVIDED IN SECTIONS OF THIS DOCUMENT EXACTLY MATCH THE STEPS AND SCREEN SHOTS THAT MAY BE SPECIFIC TO YOUR INSTALLATION. THESE INSTRUCTIONS WERE UP TO DATE IN MARCH 2018. PLEASE POST ANY DIFFERENCE YOU ENCOUNTER TO D2L TO HELP US ALL OUT.

TABLE OF CONTENTS

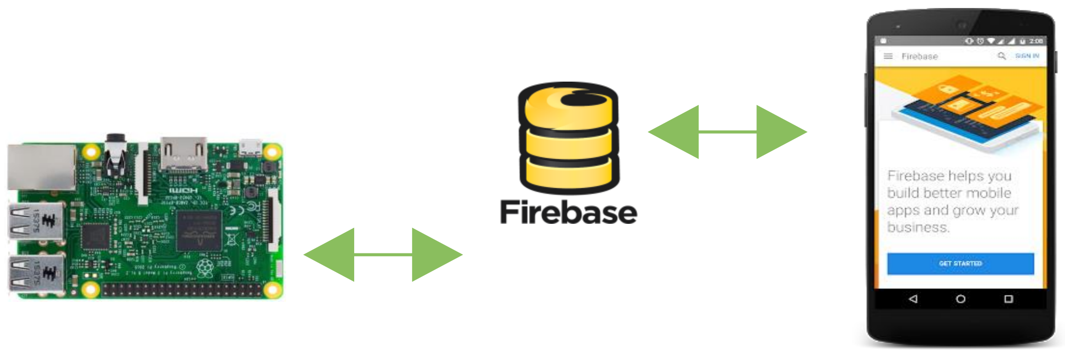
Project Overview:

The goal of project 3 is to give you hands-on experience developing IoT Embedded Firmware from scratch. We will be using a Raspberry PI 3 (RPI3) running *Android Things* as the single-board- computer in this embedded systems application. The RPI3 is connected to your Wi-Fi (or wired Ethernet) network so that it communicates with the real time database included in Google's Firebase. The mobile application communicates with the same Firebase database creating a bi-directional communication path between the app running on the mobile device and the RPI3. You will be using the API provided by Firebase for both of your applications. The RPI3 app will parse commands from the mobile app and send the appropriate transactions to the pre-programmed Microchip PIC microcontroller via I^2C. The PIC will, in turn, perform the necessary action (set DAC output, set PWM duty cycle, send ADC values to RPI3, etc). The RPI3 app will also periodically read the ADC channels and send the data to the Firebase database in "the cloud." Incidentally, an official definition of "the cloud" is:

*"In reality '**the cloud**' is just a shorthand term for 'cloud computing', which itself just refers to the idea of using someone else's computers (usually, but not always, operated by a business) on the Internet for things we previously used our own for like storing data and running programs"*

Source: http://home.bt.com/tech-gadgets/computing/cloud-computing/eight-things-you-need-to-know-about-the-cloud-11363891172534

This document provides detailed information on how to set up the Firebase database and develop the application on both ends (RPI3 and mobile app). Both apps can be developed in the Android Studio IDE using Android API's. We organized Project #3 this way so that you could, for example, leave the RPI3 at home and control & monitor remotely from your Android device. All it takes is an internet connection – this really is an IOT application as the following figure shows:


Firebase
Firebase helps you build better mobile apps and grow your business.
GET STARTED

Google Firebase

Source: https://en.wikipedia.org/wiki/Firebase

Google's Firebase is a cloud-based suite of services for mobile platform and web-based development. It was originally developed by Firebase Inc. which was acquired by Google in 2015. Firebase has grown inside Google and has expanded their services to become a unified platform for mobile developers. It now integrates with various other Google services to offer broader products and scale for developers. Firebase competes with Amazon Web Services (AWS) and numerous other cloud service offerings by Microsoft, IBM, Oracle, and others.

Firebase includes the following major services:

- **Firebase Analytics** is a free app measurement solution that provides insight into app usage and user engagement.
- **Firebase Cloud Messaging** (formerly known as Google Cloud Messaging (GCM)) is a cross- platform solution for messages and notifications for Android, iOS, and web applications, which currently can be used at no cost.
- **Firebase Auth** is a service that can authenticate users using only client-side code. It supports social login providers Facebook, GitHub, Twitter and Google. Additionally, it includes a user management system whereby developers can enable user authentication with email and password login stored with Firebase.
- **Realtime Database** and backend service. The service provides application developers an API that allows application data to be synchronized across clients and stored on Firebase's cloud.
- **Firebase Storage** provides secure file uploads and downloads for Firebase apps, regardless of network quality. The developer can use it to store images, audio, video, or other user-generated content. Firebase Storage is backed by Google Cloud Storage.
- **Firebase Notifications** is a free service that enables targeted user notifications for mobile app developers.

We will only be using the Firebase realtime database services for this project but a number of ECE 558 final project teams have used additional services in their projects.

Setup Database

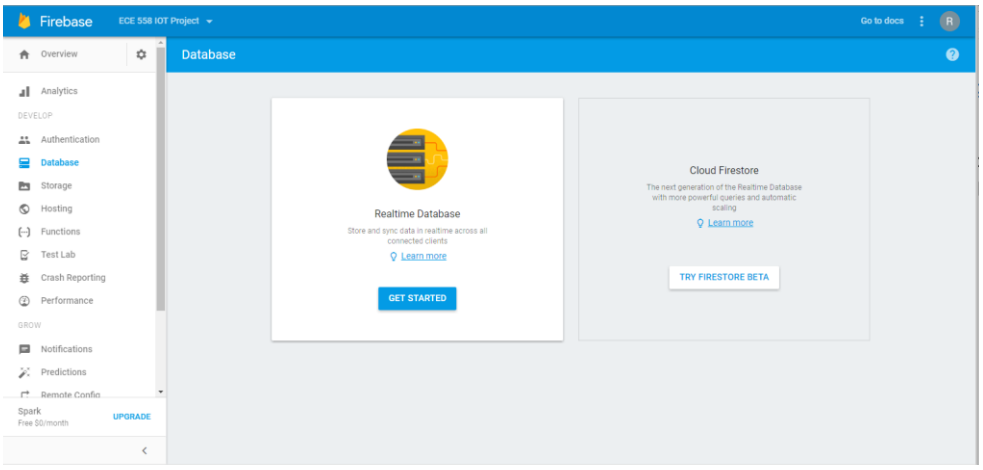
In this part of the project you will set up a realtime database as the medium of communication between the RPI3 and the mobile Android device. The mobile app (user interface) will update these settings in response to changes in the sliders (SeekBar in Android-context) and buttons (Increment and Decrement) that you include in your app. The remote station app running on the RPI3 will listen to those settings to vary the duty cycles of the PWM channels in the PIC, and hence the brightness, of the individual LEDs. Since we have bi-directional communication, the user application should also display the ADC values read by the RPI3 from the PIC. The changes on the database reflect on all the devices that are linked to it.

Read this link in order to help understand the further steps explained below:

https://firebase.google.com/docs/database/android/start

STEP 1: Install the Firebase SDK if it is not installed in Android Studio.
STEP 2: Use a Google account to access the Firebase Console (https://console.firebase.google.com); Add a project, giving it a name and unique Project ID.

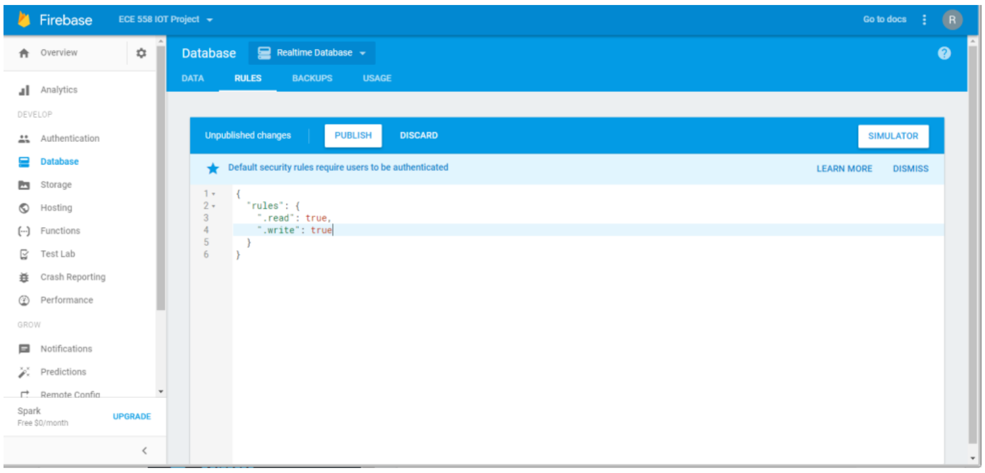
STEP 3: Change the rules of the database so that any app can read and write the values into database (i.e. this project does not require a login to access the database). Rules provide access to your database. Click on Database in your Firebase project and GET STARTED



Store and sync data in realtime across all connected clients

The next generation of the Realtime Database with more powerful queries and automatic scaling

STEP 4: Select the RULES tab as shown in below figure and change the values from “auth != null” to true for .read and .write. PUBLISH and DISMISS the RULES dialog.



Default security rules require users to be authenticated

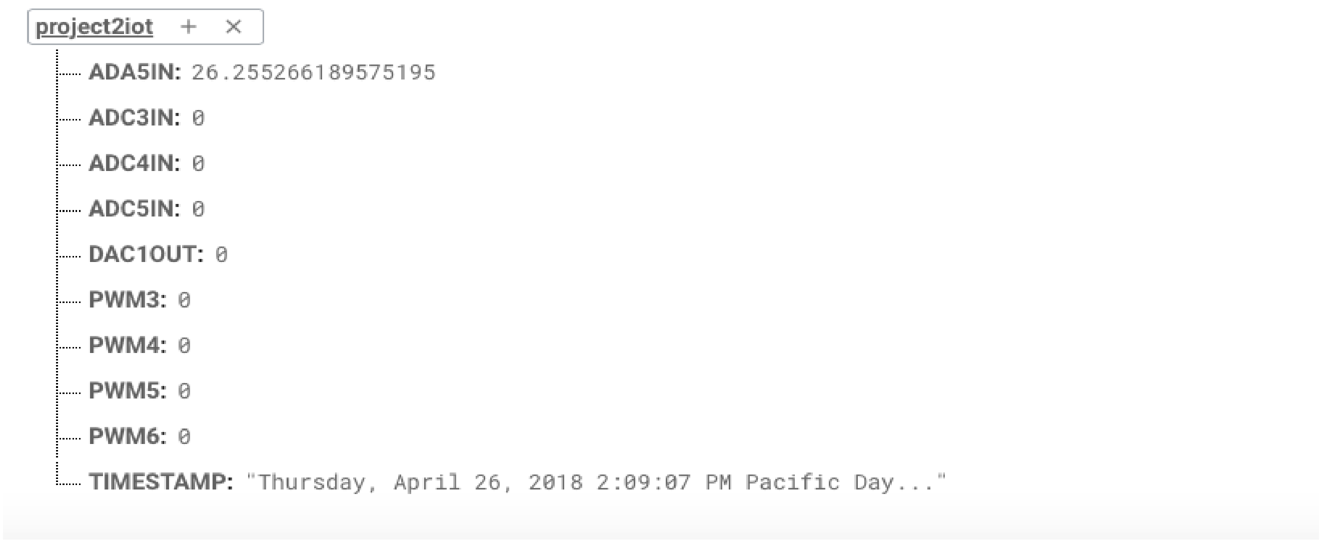
```
{
  "rules": {
    ".read": true,
    ".write": true
  }
}
```

STEP 5: Add your child elements to the database

- 4 - PWM child fields
- 4 - ADC child fields
- 1 - DAC child field
- and a TIMESTAMP which indicates the last connection time & date

These will become the reference names for the field. Select the DATA tab for the Realtime Database and use the + key to add the field and its initial value. You are welcome to choose your own names and initial values.

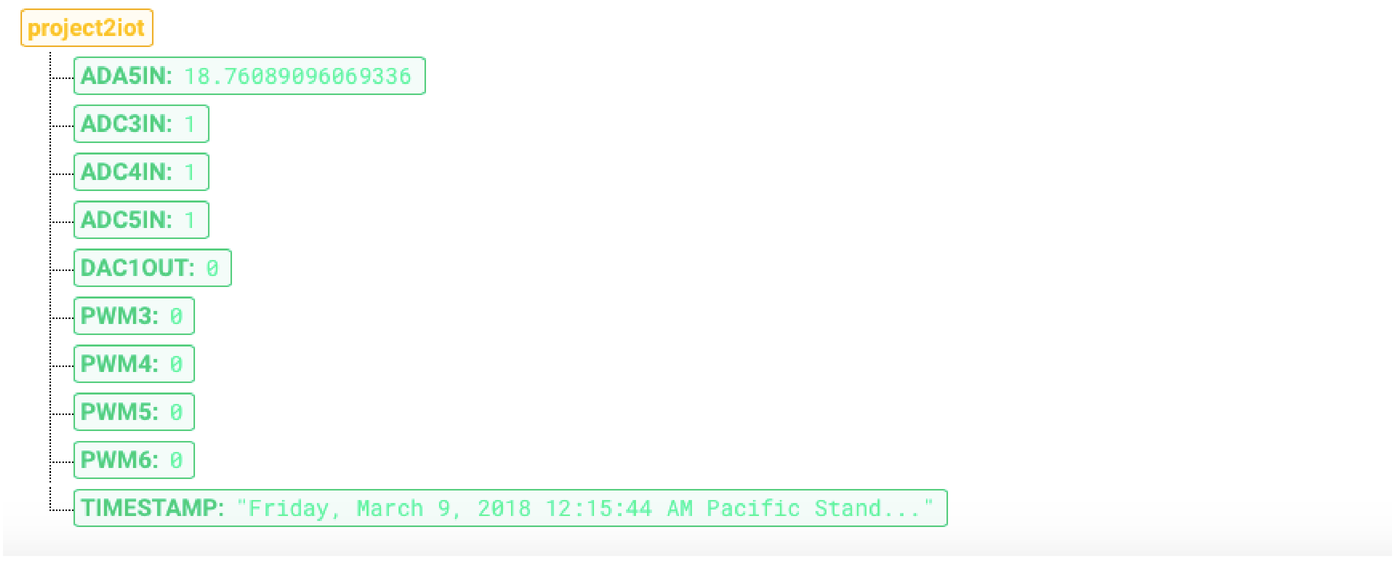
The screenshot below is an example:



```
project2iot  +  ×
  ADA5IN: 26.255266189575195
  ADC3IN: 0
  ADC4IN: 0
  ADC5IN: 0
  DAC1OUT: 0
  PWM3: 0
  PWM4: 0
  PWM5: 0
  PWM6: 0
  TIMESTAMP: "Thursday, April 26, 2018 2:09:07 PM Pacific Day..."
```

Realtime Database:

- The realtime database service provides application developers an API that allows application data to be synchronized across clients and stored in the Firebase cloud. This database acts as the bridge between the mobile app running on an Android device and the RPI3.
- The RPI3 application monitors the changes in Firebase (PWM and DAC) values which in turn sends appropriate signals to be performed on the PIC microcontroller. (changing the brightness (duty cycle) of the LED and to output appropriate voltage signal based on the inputs specified (DAC)).
- A Firebase structure is established and the required {key,value} pair should be created in order to monitor and listen for the changes from both the ends.



```
project2iot
├── ADA5IN: 18.76089096069336
├── ADC3IN: 1
├── ADC4IN: 1
├── ADC5IN: 1
├── DAC1OUT: 0
├── PWM3: 0
├── PWM4: 0
├── PWM5: 0
├── PWM6: 0
└── TIMESTAMP: "Friday, March 9, 2018 12:15:44 AM Pacific Stand..."
```

Raspberry PI 3 B (The Model RPI3B+ does not work)

The Raspberry PI 3 used in this project can be thought of as the central controller of the IOT device. We selected the RPI3 because it is one of the fully supported platforms for Android Things.

RPI3 Application

- In this project, the PWMx and the DAC1OUT values are controlled from the mobile app by writing values to Firebase. Changes in these values are propagated to the RPI3 and the RPI3 communicates with the PIC microcontroller to cause it to perform the requested action.

- In this project the ADA5IN & ADCxIN values from the PIC are monitored by the RPI3 which updates the corresponding values in the Firebase database. Changes in these values are monitored by the mobile app which updates the Textviews, etc. in the layout.

The RPI3 control app performs two major functions

1. Reads ADC values continuously from PIC and updates the appropriate child values in the Firebase.
2. Monitors / Listens to changes in the child values (PWM & DAC) and sends appropriate signals to the PIC.

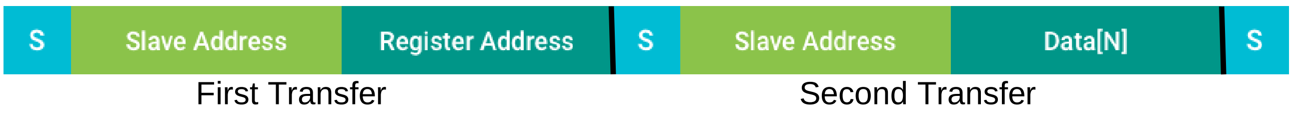
Firebase Integration

The controller app gets the values from the Firebase database using the *addValueEventListener( )* Interface and its *onDataChange( )* method when one or more of the values changes and sets the PWM duty cycle and/or the DAC output when they change.

I²C Protocol and SMBus:

The RPI3 is configured as an I²C Master and the PIC microcontroller is configured as I²C Slave device for this project. Communication between the RPI3 and the PIC is based on SMBus.

System Management Bus (SMBus) is a common protocol implementation that exists on top of I²C to interact with register data in a standard way. SMBus commands consist of two I²C transfers as follows:



The first transaction identifies the register address to access, and the second reads or writes the data at that address. Logical data on a slave device may often take up multiple bytes, and thus encompass multiple register addresses. The register address provided to the API is always the first register to reference.

Note: Per the SMBus protocol, the master will send a "repeated start" condition between the address and data transactions.

Peripheral I/O provides three types of SMBus commands for accessing register data:

- **Byte Data** - `readRegByte()` and `writeRegByte()` Read or write a single 8-bit register value.
- **Word Data** - `readRegWord()` and `writeRegWord()` Read or write two consecutive register values as a 16-bit little-endian word. The first register address corresponds to the least significant byte (LSB) in the word, followed by the most significant byte (MSB).
- **Block Data** - `readRegBuffer()` and `writeRegBuffer()` Read or write up to 32 consecutive register values as an array.

An EEPROM buffer is emulated in the I2C driver on the PIC microcontroller. The RPI3 reads/writes logical data values to this buffer. Based on the values stored / updated in the buffer, appropriate hardware action takes place. First the matching of slave address is done. If the address's match, the next byte transferred is the logical data packet which is placed on to the appropriate register in the PIC controller. The slave addresses and Peripheral address are provided below.

I²C Slave Register Addresses:

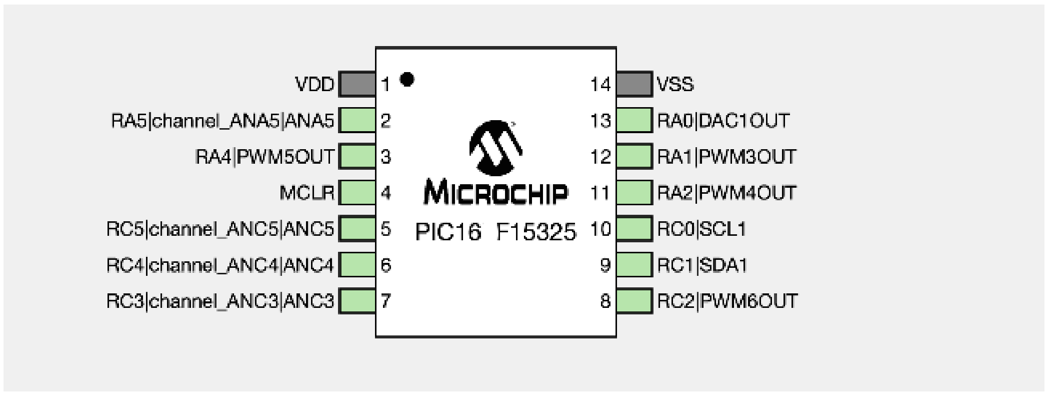
```
/**************************I2C Slave Address**********************************/
I2C_Slave = 0x08

/*********PWM Address (10 bit resolution: Use dutyCycle 0-100%)**************/
PWM3_ADDRESS = 0x00;
PWM4_ADDRESS = 0x01;
PWM5_ADDRESS = 0x02;
PWM6_ADDRESS = 0x03;

/****************DAC Address (5 bit resolution: Use 0-31)*******************/
DAC1_ADDRESS = 0x04;

/***************************ADC Address (10 bit resolution)************************/
ADA5_ADDRESS = LSB : 0x05 ; MSB : 0x06
ADC3_ADDRESS = LSB : 0x07 ; MSB : 0x08
ADC4_ADDRESS = LSB : 0x09 ; MSB : 0x0a
ADC5_ADDRESS = LSB : 0x0b ; MSB : 0x0c
```

PIC Microcontroller (PIC16F15325)



`Fixed Voltage Reference (FVR)` ==> 1.98V (You will require this for calibrating TMP36)

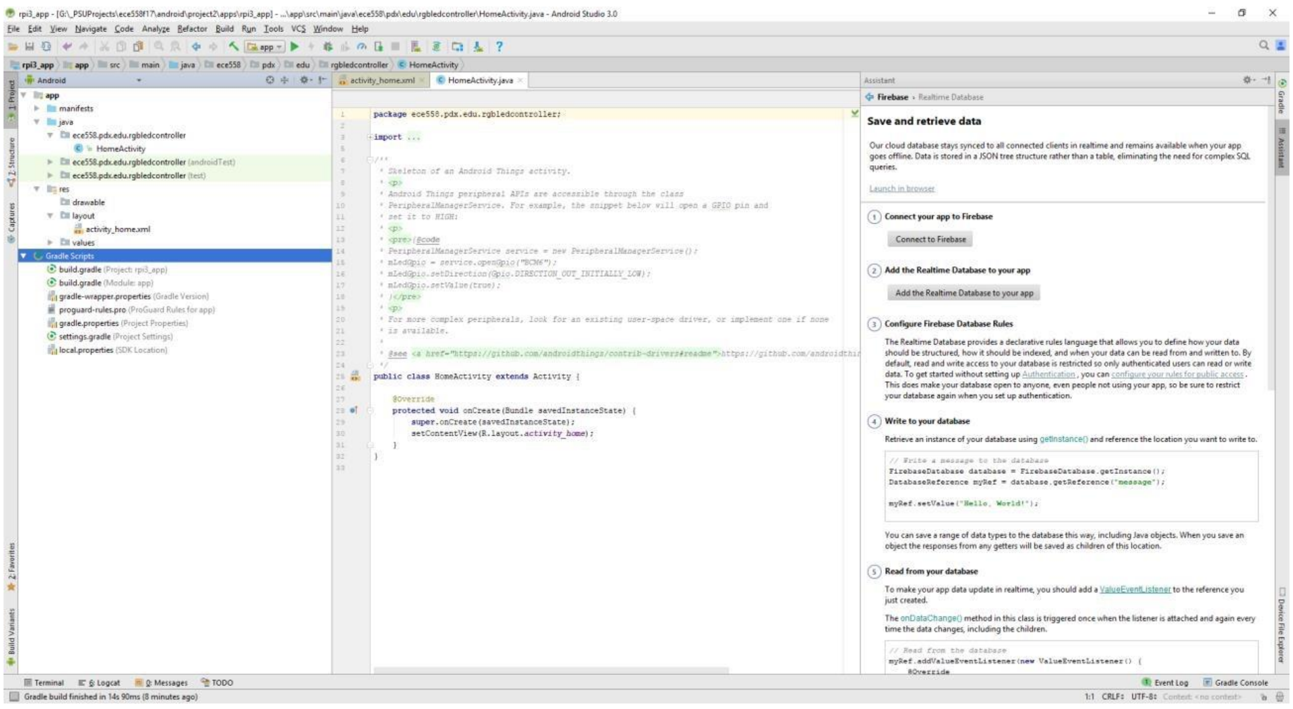
Mobile Application

The user interface for this project is implemented as a single activity Android app which runs on your mobile device and writes and reads values from the Firebase realtime database.

Firebase Integration:

The user application uses the Firebase API to connect and communicate to the Real-Time Database services. Android Studio includes excellent support for integrating Firebase into your app.

The screenshot below is used as a reference to the dedicated Firebase database server established for this project.



```
package ece558.pdx.edu.rgbledcontroller;

import ...

/**
 * Skeleton of an Android Things activity.
 * <p>
 * Android Things peripheral APIs are accessible through the class
 * PeripheralManagerService. For example, the snippet below will open a GPIO pin and
 * set it to HIGH:
 * <p>
 * <pre>{@code
 * PeripheralManagerService service = new PeripheralManagerService();
 * mLedGpio = service.openGpio("BCM6");
 * mLedGpio.setDirection(Gpio.DIRECTION_OUT_INITIALLY_LOW);
 * mLedGpio.setValue(true);
 * }</pre>
 * <p>
 * For more complex peripherals, look for an existing user-space driver, or implement one if none
 * is available.
 *
 * @see <a href="https://github.com/androidthings/contrib-drivers#readme">https://github.com/androidthi
 */
public class HomeActivity extends Activity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_home);
    }
}
```

Firebase › Realtime Database

Save and retrieve data

Our cloud database stays synced to all connected clients in realtime and remains available when your app goes offline. Data is stored in a JSON tree structure rather than a table, eliminating the need for complex SQL queries.

Launch in browser

1. **Connect your app to Firebase**

   Connect to Firebase

2. **Add the Realtime Database to your app**

   Add the Realtime Database to your app

3. **Configure Firebase Database Rules**

   The Realtime Database provides a declarative rules language that allows you to define how your data should be structured, how it should be indexed, and when your data can be read from and written to. By default, read and write access to your database is restricted so only authenticated users can read or write data. To get started without setting up Authentication, you can configure your rules for public access. This does make your database open to anyone, even people not using your app, so be sure to restrict your database again when you set up authentication.

4. **Write to your database**

   Retrieve an instance of your database using getInstance() and reference the location you want to write to.

```
// Write a message to the database
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference myRef = database.getReference("message");

myRef.setValue("Hello, World!");
```

   You can save a range of data types to the database this way, including Java objects. When you save an object the responses from any getters will be saved as children of this location.

5. **Read from your database**

   To make your app data update in realtime, you should add a ValueEventListener to the reference you just created.

   The onDataChange() method in this class is triggered once when the listener is attached and again every time the data changes, including the children.

```
// Read from the database
myRef.addValueEventListener(new ValueEventListener() {
    @Override
```

Gradle build finished in 14s 90ms (8 minutes ago)

STEP 1: Create a new project in Android Studio.

STEP 2: Connect your app to your Firebase project using either the Firebase Assistant in Android Studio or the Add Firebase to your Android app manually. To use the Firebase Assistant Select `Tools –> Firebase/Realtime Database/Save and retrieve data` from the Android Studio menu. Follow the steps. You should see something similar to this (depending on how you created the project) if the app has been successfully connected to the Firebase project.

STEP 3: Implement the user interface app.

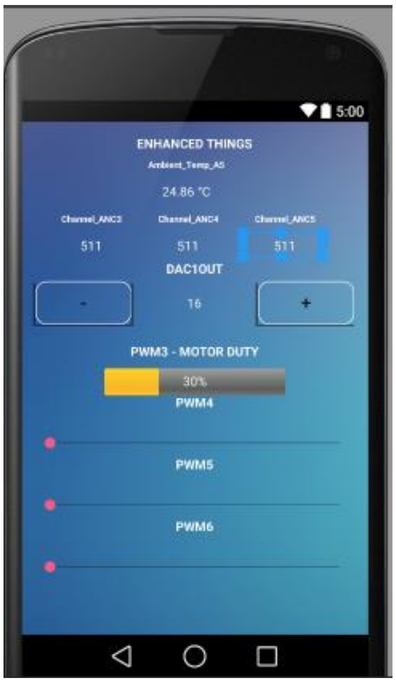
Functionality

This user application performs two major functions:

- Listens to the changes on the ADC values on the server and updates the appropriate text box fields.
- Controls child values of the PWM and DAC output using seek bars and switch buttons.

The user application has text fields, increment/decrement buttons and several seek bars in order to control the PWM and DAC outputs and monitor the changes on the ADC values. Your app can change the values of the database fields by retrieving a reference to the database and the fields and use the *setValue( )* method on the field to update the value. You can find examples of how to implement SeekBar listeners online. As you can imagine, there are many possible exceptions – please handle them appropriately.

The following screen shot shows one possible organization for the mobile app:



Manual Changes

Note: The version for Android Things has changed since March 2018 and it is likely that the current version of the firebase-core and firebase-database have changed. Change the dependencies as appropriate. It may be that later version of Android Studio already incorporate these changes.

Android Things devices expose APIs through support libraries that are not part of the Android SDK. The new project wizard automatically adds a dependency to the support library to your app-level `build.gradle` file:

```
 dependencies {
        ...
        provided 'com.google.android.things:androidthings:0.7-devpreview'
        compile 'com.google.firebase:firebase-core:11.6.0'
        compile 'com.google.firebase:firebase-database:11.6.0'
    }
    repositories {
        jcenter()

        maven {
       // Google's Maven repository
            url "https://maven.google.com"
        }
     }
```

The New Project wizard adds <uses-library> to your app's manifest file to make this prebuilt library available to the app's classpath at run time. Also add in your Firebase dependencies. We have attached build.grade (Module : app) dependencies (above).

Add the following (below) to the dependencies of your build.gradle (Project)

```
      classpath 'com.google.gms:google-services:3.1.1'
```

AndroidManifest.xml (For reference):

```
<application
    android:label=“@string/app_name">
    // Add the following
    <uses-library android:name=“com.google.android.things"/>
    <activity android:name=“.MainActivity">
        <!-- Launch activity automatically on boot -->
        <intent-filter>
            <action android:name=“android.intent.action.MAIN"/>
            <category android:name="android.intent.category.IOT_LAUNCHER"/>
            <category android:name="android.intent.category.DEFAULT"/>
        </intent-filter>

        <!-- Launch activity as default from Android Studio -->
        <intent-filter>
            <action android:name="android.intent.action.MAIN"/>
            <category android:name="android.intent.category.LAUNCHER"/>
        </intent-filter>
    </activity>
</application>
```

References

1. developer.android.com. (2018). *Android Things | Android Things*. [online] Available at: https://developer.android.com/things/index.html

2. (2018). *Raspberry Pi 3 | Android Things*. [online] developer.android.com. Available at: https://developer.android.com/things/hardware/raspberrypi.html

3. Tool, P. (2018). *PIO CLI Tool | Android Things*. [online] developer.android.com. Available at: https://developer.android.com/things/sdk/pio/pio-cli.html

4. Learn.sparkfun.com. (2018). *I2C - learn.sparkfun.com*. [online] Available at: https://learn.sparkfun.com/tutorials/i2c

5. Learn.sparkfun.com. (2018). *I2C - learn.sparkfun.com*. [online] Available at: https://learn.sparkfun.com/tutorials/pulse-width-modulation

6. Microchipdeveloper.com. (2018). Home - Developer Help. [online] Available at: http://microchipdeveloper.com

Revision History

1.0	26-March-2018 (KA)	Major changes in the problem statement. Added a micro-controller, a temperature sensor for the project. Updated to the latest AndroidThings OS 0.6.0 platform.
1.1	29-April-2018 (KA)	Automated the motor control (DC motor) based on the temperature setpoint. Updated to the latest AndroidThings OS 0.7.0 platform.
1.2	13-May-2018 (RK)	Minor changes, clean-up prior to release
1.3	03-Nov-2018 (RK)	Added warnings that the specific screen shots may have changed since Spring 2018. Will need to update the screenshots before Fall 2019.
1.4	14-May-2019 (RK)	Removed RPI3 B+ as a suitable platform, other minor changes.