ECE 558 - EMBEDDED SYSTEMS PROGRAMMING
PROJECT 3

AndroidThings - Hardware

Prepared By:

KIYASUL ARIF ABDUL MAJEETH

Revised By:

ROY KRAVITZ

TABLE OF CONTENTS

Project Overview

The main goal of this project is to control an RGB LED and monitor a temperature sensor remotely using Google's Firebase realtime database as the communication bridge between a user application running on a mobile Android device and the Raspberry Pi 3 (RPI3) running a control app. The apps will also control the speed of a low cost DC motor based on the reading of the temperature sensor. The RPI3 (master) communicates with a PIC16F (slave) series microcontroller via I^2C. The PIC contains the PWM channels (Pulse width modulation generator), DACs (digital to analog converter), and ADC's (analog to digital converter) actually connected to the devices.

We added the PIC microcontroller (cost ~ $1.00) to the project to address the limited number of hardware PWM pins (only 2) and lack of built-in ADC and DAC channels on the RPI3, and to extend the capability of this IoT platform. Distributed processing architectures such as this are becoming more prevalent as the cost of SoC-based microcontrollers and microcomputers decreases because the functionality is distributed to devices most suitable for meeting the performance and cost needs of the application.

This document describes the necessary hardware tools and knowledge required to know before starting the software development of the apps.

Bill of Materials (costs are approximate)

DESCRIPTION	QUANTITY	UNIT	COST
RASPBERRY PI3 MODEL B W/ MICRO SD CARD **(Note: Android Things has NOT be ported to the RPI3 B+ and does not work. Use a Model RPI3 B)**	1	$50	$50
Pre-programmed PIC16F15325	1	$1.07	$1.07
TMP36	1	$1.50	$1.50
DC Motor 6V 11.5krpm	1	$3.50	$3.50
270 ohm resistor	6	$0.30	$1.80
330 ohm resistor	1	$0.10	$0.10
1N4148 General Purpose Diode	1		
2N3904 NPN Transistor	1		
Common Cathode RGB LED	2	$0.50	$1
Prototyping board	1	$4	$4
M-F & M-M jumper wires	As required	$5 approx.	$5
TOTAL COST			**$70.00***
Pushbutton	Optional	Needed if you want to try the *Create a Things App* tutorial (https://developer.android.com/things/training/first-device/)	

*NOTE: The cost from the EPL may vary by a few dollars

Although the cost is not trivial (we feel your pain) our hope is that you will be able to make use of the hardware for other courses and projects.

The following is required to bring up and configure the RPI3 but are not included in the final configuration:

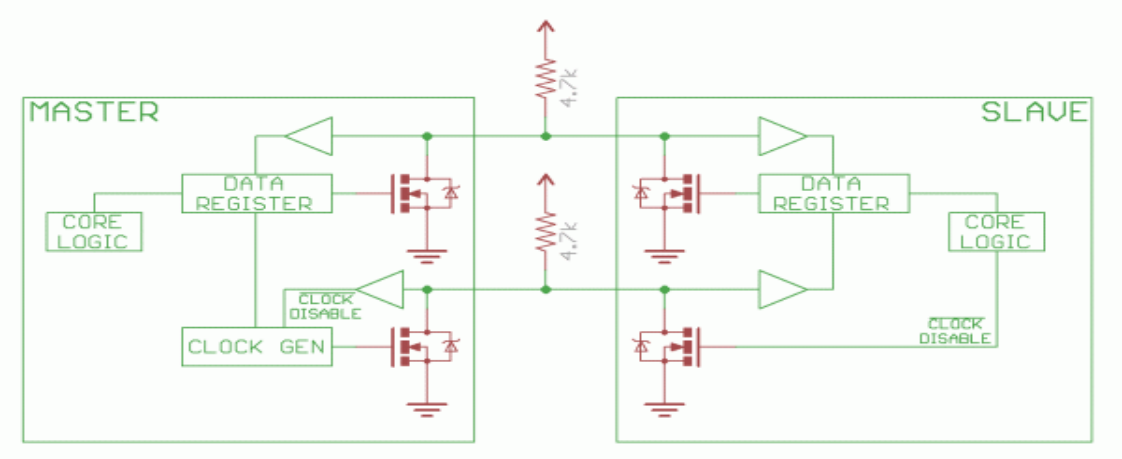
- VGA – HDMI Cable or HDMI – HDMI Cable if you have an HDMI-compatible monitor.
- Memory Card Reader (to flash the image file on to SD Card).
- USB Mouse and Keyboard to configure Wi-Fi (or) you may use `adb` to setup Wi-Fi
- (Optional) Ethernet LAN cable (to configure Wi-Fi the first time)

Brush-up Topics

Inter-integrated Circuit Protocol (I^2C):

The Inter-integrated Circuit (I^2C) Protocol is a hardware-based protocol intended to allow multiple "slave" digital integrated circuits ("chips") to communicate with one or more "master" chips. Like the Serial Peripheral Interface (SPI), I^2C is intended for short distance communications within a single printed circuit assembly. Like Asynchronous Serial Interfaces, I^2C only requires two signal wires to exchange information.

Each I^2C bus segment consists of two signals: SCL and SDA. SCL is the clock signal, and SDA is the data signal. The clock signal is always generated by the current bus master. Slave devices may force the clock low at specific times to delay the master sending more data or to take more time to prepare data before the master attempts to receive it. This is called "clock stretching" and is not supported on the RPI3. Unlike UART or SPI connections, the I^2C bus drivers are "open drain", meaning that they can pull the corresponding signal line low but cannot drive it high. Thus, there can be no bus contention where one device is trying to drive the line high while another tries to pull it low, eliminating the potential for damage to the drivers or excessive power dissipation in the system. Each signal line has a pull-up resistor on it, to restore the signal to high when no device is asserting it low.



Notice the two pull-up resistors on the two communication lines.

Resistor selection varies with devices on the bus. I^2C is a fairly robust protocol, and can be used with short runs of wire (2-3m). For long runs, or systems with lots of devices, smaller resistors are better.

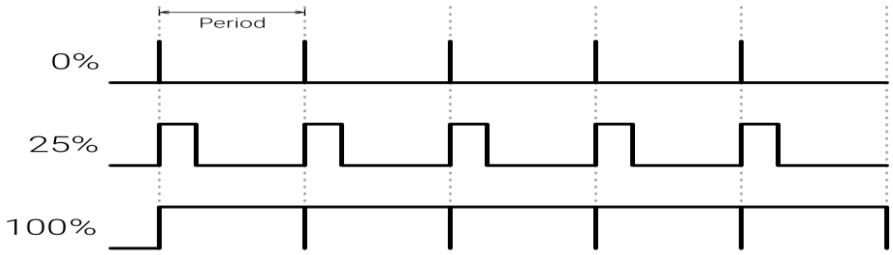
The Raspberry Pi3 Model B has an internal pull-up resistors on its SCL (I^2C CLK) and SDATA (I^2C DATA) so the external pull-up resistors are not necessary.

Pulse Width Modulation (PWM):

PWM is a technique used widely to control the average voltage being applied to a device like a DC motor or the anodes of the Red, Blue and Green LEDs in the RGB LED. The average voltage being applied to the

device is varied by changing the high time of the pulse in relation to the period of the signal. For example, an LED driven by a 3.3v signal with a 60% duty cycle receives an average voltage of about 2v. We apply PWM in this project to control the brightness of all the LEDs in the RGB LED and to control the speed of the motor.

The human eye will average the intensity of the LEDs (illuminated when the duty cycle is high, dark when the duty cycle is low) to perceive the color of the LED. As long as we continuously change the state of the LED from ON->OFF- >ON at least 60 times/second (60 Hz) the eye will not perceive any flicker.



Analog to Digital Converter (ADC):

Microcontrollers are capable of detecting binary signals: is the button pressed or not? These are digital signals. When a microcontroller is powered from five volts, it interprets zero volts (0V) as a binary 0 and a five volts (5V) as a binary 1. But what if the signal is 2.72V? Is that a 0 or a 1? We often need to measure signals that vary; these are called analog signals. A 5V analog sensor may output 0.01V or 4.99V or anything in between. Luckily, nearly all microcontrollers have a device built into them that allows us to convert these voltages into values that we can use in a program to make a decision.

An Analog to Digital Converter (ADC) converts an analog voltage on a pin to a binary number. By converting from the analog world to the digital world, we can begin to use electronics to interface to the analog world around us.

In this project, you will be using an analog temperature sensor (TMP36) to read analog values and calibrate it to display readings on the user application.

Digital to Analog Converter (DAC):

In electronics, a digital-to-analog converter (DAC, D/A, D2A, or D-to-A) converts a binary value into an equivalent analog value between 0 volts the DAC reference voltage. For example, an 8-bit DAC with a 5V reference voltage and a binary value of 0x7F would produce an analog voltage of about 2.5V. DACs are often used to convert finite-precision time series data to a continually varying physical signal.

The single DAC output on the PIC is connected to all the ADC channels on the PIC except for the one that is being used to measure the temperature as inputs. Note that the ADC value will be scaled to match the resolution of the DAC. For example, setting the DAC output value to 16 should result in a reading of 511 from the ADC. This is because the DAC on the PIC has 5-bit resolution and the ADC has 10-bit resolution.

Raspberry **PI 3 Model B** (the Models RPI3 B+ and RPI4 do 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 two fully supported platforms for Android Things. As a fairly powerful 32-bit single board computer with HDMI and audio it is overkill for this particular IoT device, but it is cost effective and well supported by the Raspberry Pi Foundation (https://www.raspberrypi.org/) and a vibrant "community." The hardware design is not open source - RPI's are targeted for the education market and are generally not considered for commercial projects of any volume. There are open source hardware platforms like the Beaglebone (https://beagleboard.org/bone) line of SBC's that are better suited for commercial applications.

Installing and configuring Android Things for the RPI3:

Android Things 1.0 has been in production for more than a year and provides a robust platform for developing IoT applications using the RPI3 and Android Studio. You can find instructions for how to install and configure Android Things on your RPI3 development board. Our advice is to use the install script for your chosen OS since that automates much of the process, but the link also provides instructions for installing and configuring Android Things manually if you are adventurous or if you run into problems.

https://developer.android.com/things/hardware/raspberrypi

While you could certainly overwrite the Raspbian image and/or NOOBS that comes on the microSD card included with many of the RPI development kits (including the kit sold by the EPL) you may find it more convenient to buy a microSD card for your Android Things development. MicroSD cards are always on sale (Best Buy currently has 32GB microSD cards on sale for ~$10.00).

NOTE: Windows may insist on trying to reformat the microSD card if the card you are using holds a Linux filesystem. That was Roy's case when he upgraded to the latest version of Android Things (1.0.5). Cancel out and ignore the demands and complaints. The etcher tools will rewrite the microSD card with the new image. BTW, Windows is correct to complain - the Linux filesystem is not a FAT or NTFS file system.

While it is not required, we believe that completing the *Create a Things App* tutorial could help you ramp up more quickly on the RPI3/Android Things portion of the project. You can find the tutorial at:

https://developer.android.com/things/training/first-device/

Starting the Application via ADB shell:

Once you have successfully configured the RPI3 you can develop, install, and launch your control app on the RPI3 using adb (the **A**ndroid **D**ebug **B**ridge).

STEP 1:- After developing the RPI3 Application on Android Studio, the signed .apk file is generated & installed on the RPI3 and the activity is launched via the ADB shell prompt.

STEP 2:- The adb is executed and connected to the RPI3 through its IP address

- `./adb connect 192.168.1.7 (IP address of my RPI3)`

STEP 3:- The .apk file is installed using the command

- `./adb install ~/THE_PATH/app-release.apk`

STEP 4:- The application is started using the command

- `./adb shell am start -n PackageName/.MainActivity`

PIO CLI Tool:

A pio shell command line tool enables verification and testing of hardware peripheral connections without having to write any code on the RPI3. This command line tool enables you to query and control the various interfaces supported by Peripheral I/O. You can use this tool to do the following:

- Easily discover the pin names for a given development board
- Ensure peripheral devices are connected to the board correctly
- Run basic functionality tests as well as scripted tests

An `adb` shell (or the Serial debugger shell if you are using a USB to Serial debug cable) should be used to access this tool. Basic I2C bus communication can be tested using the following commands.

Example :-

```
rpi3:/ $ pio i2c I2C1 0x08 write-reg-byte 0x01 0x25
rpi3:/ $ pio i2c I2C1 0X08 read-reg-byte 0x01 0x25
```

where,
i2c - I2C peripheral I/O
I2C1 - Bus Name
0x08 - Slave Address (PIC)
0x01 - Read/Write value to this register address
0x25 - Value to be written

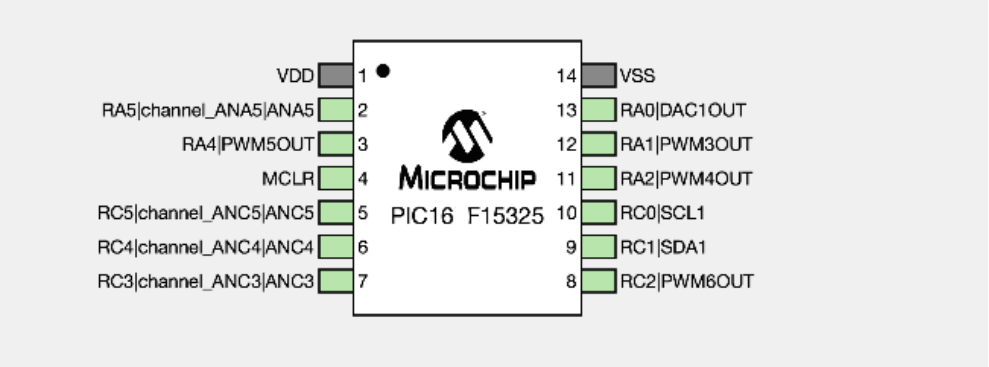
Note: The PIO CLI tool should be used when the application on RPI3 is not running in the background. An I/O error will occur if the user tries to open the (already) configured pins (from RPI3 app) from the pio cli tool. And the Android SDK tool directories are not, by default, included in the path. The default location for the platform-tools directory is:

Windows : `Users/username/AppData/Local/Android/sdk/platform-tools`

MacOS : `Users/“yourusername”/Library/Android/sdk/platform-tools`

PIC Microcontroller

The Project #3 circuit includes a Microchip PIC16F15325. This is a low cost, 8-bit microcontroller that has been a staple in the microcontroller market for many years. The microcontroller contains a number of peripheral devices and comes pre-programmed from the EPL store.



PIC16F15325 Pinout:

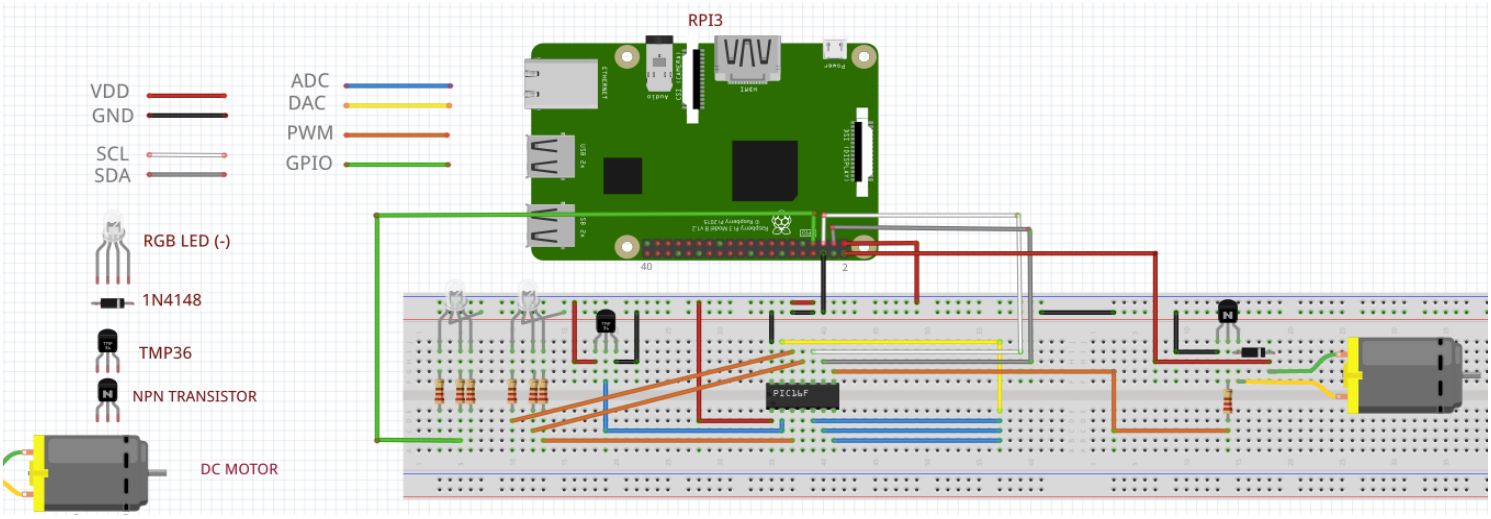
RPI3	CONNECTED TO	PIC
PIN1(3.3v)	CNT	PIN1(VDD)
PIN6(GND)	CNT	PIN14(GND)
PIN3(I2C1-SDA)	CNT	PIN9(SDA)
PIN5(I2C1-SCL)	CNT	PIN10(SCL)

Interfacing with the RPI3 via I2C:

The RPI3 communicates to the peripheral I/O of the PIC microcontroller using the I^2C protocol. Appropriate data is sent to a buffered register in the PIC to control the duty cycle of PWM pins, read values from the ADC channels and send digital values to output appropriate analog signal (DAC).

Circuit Schematic / Wiring Diagram

You will need to construct the following circuit to complete this project. The wiring diagram shown below



is for one of those white proto boards. You could also create a PCB and produce it in the EPL, although do so quickly because the EPL gets busy towards the end of the term. In fact, I'm willing to consider Extra Credit for an enterprising student who produces an easy to use PCB for the project. The EPL has the capability to cut single and double sided PCB's.

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 Android Things 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	14-May-2018 (RK)	Final updates and clean-up before project release
1.3	03-Nov-2018 (RK)	Modified for Android Things OS 1.0
1.4	12-May-2019 (RK)	Modified to remove RPI3 B+ as one of the supported development platforms.
1.5	11-Nov-2019 (RK)	Fixed minor typos. No functional changes.