CNT5517/CIS4930 Mobile Computing

Fall 2020

Professor Sumi Helal

Lab 2 - Metronome Thing Implementation Using Raspberry Pi

Due Date: 11:55am Tuesday 29 September 2020.

In this assignment you will set up and (learn how to) program your Raspberry PI 3 Model B, while simultaneously preparing the device to become a Thing in the Internet of Things. Specifically, you will create a simple metronome using push buttons and LEDs on your Pi board, and then create a web interface to interact with the device remotely.

Background

A metronome is a device that creates a visual or audible pulse at consistent intervals. A common measurement for the speed, or tempo, of a metronome is Beats per Minute (BPM). A metronome at 60 BPM will pulse once every second, and at 120 BPM, twice every second, etc.

A common way to set the speed of modern metronomes is for the user to repeatedly press a button at the desired tempo. The metronome will calculate the average time between the presses and set the BPM accordingly. You will emulate this feature using one of two push buttons needed for this assignment.

Requirements

The following will be required for this assignment:

- Raspberry Pi 3 Model B and online manuals and programming tips.
- Your laptop

- One breadboard
- Jumper wires (male to male to use on the breadboard, and female to male jumper wires to connect your IO port on your Pi to the breadboard)
- Resistors
- Tne LED: one Green and one Red.
- Two pushbuttons
- USB-x to Ethernet adaptor cable to connect your Pi to your laptop. This way you will be able to use ssh to connect to your RPi from your laptop, which avoids having to connect a monitor, keyboard, mouse etc to the RPi. Also, UF WiFi may not allow you to do SSH from your laptop to your Pi, hence the tethered connection. To more information on this specific requirement see the additional document "**Lab2-Configuring your Raspberry Pi Remote Connection.pdf"** placed in Canvases' FILE section.

Part One — Embedded Application & RESTful API

For the first part of this lab, you will create a metronome-style feature on your Pi board. The metronome should have two modes; a "**play**" mode, which will blink an LED at a given tempo, and a "**learn**" mode, in which the user will repeatedly press a button to set the tempo. The two modes will be toggled between using another dedicated push button.

In the "play" mode, the green LED will blink repeatedly at the previously specified tempo. On startup, the default mode should be at a tempo of 0, meaning the LED will always be off.

In the "learn" mode, the red LED will pulse every time the user presses the button, and the time of the tap will be recorded. Once the user is done tapping (i.e., the user presses the mode button again), the BPM should be calculated and

the new value should be reflected in the “play” mode. At least 4 taps are required to create a valid BPM measurement.

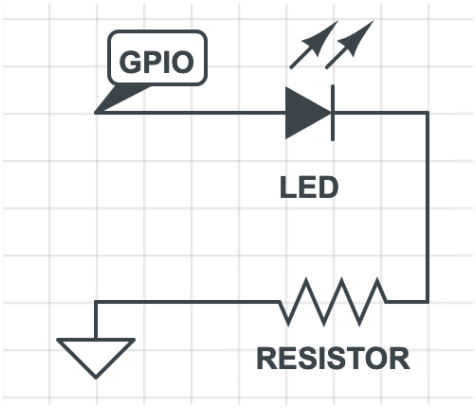
To make your metronome a Thing, your board will expose the following REST endpoints with the respective CRUD operations:

- `/bpm/` (GET, PUT): The current metronome BPM. Writing a new value here will update the speed the LED blinks at.
- `/bpm/min/` (GET, DELETE): Return the lowest BPM the user has set the metronome to (not 0). On delete, reset this value to 0 (“none”).
- `/bpm/max/` (GET, DELETE): Works the same as the `min` endpoint, but tracks the highest BPM the user has set.

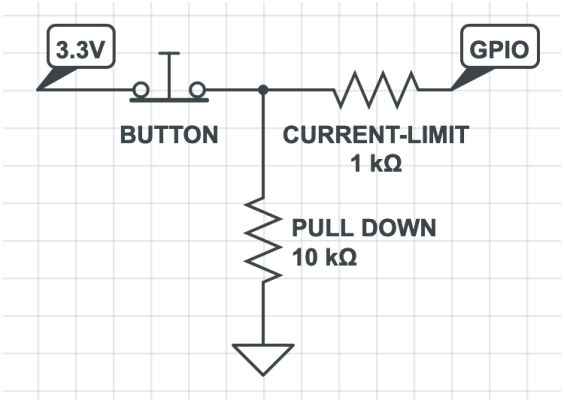
Connecting the Hardware

The Raspberry PI interacts with the push buttons and LEDs through its GPIO (General Purpose I/O) pins. Each of these pins can read and write digital (on/off) values, controlled through the command line or a library. Each pin is numbered so that it can be accessed through the software; this numbering can be found here. Note that some pins are reserved for ground and power, and that there are multiple voltages available. ONLY USE THE 3.3V POWER PINS. The RPi GPIO senses at this voltage, so using 5V may damage the device.

To connect the LED, a circuit must be created between an output GPIO pin and a GND pin. Note that if you plug the LED in directly, it will attempt to draw as much current as possible, and probably burn itself out. Therefore, an appropriately-sized resistor must be connected in series with the LED. An exemplary circuit diagram can be seen below. Pay attention to the orientation of the anode/cathode legs of the LED.



To connect the push button, a circuit must be created with 3.3V on one side, and an input GPIO pin on the other. This way, once the button is pushed, the 3.3V (“HIGH”) signal is connected to the input pin. In this configuration, however, the pin has no way to sense when it is not pressed, since it has no connection to GND (“LOW”)—this is called a “floating” input, and will return noise (random HIGH/LOW) in this state. To make sure the button always reads LOW when it is not pressed, we need a pull-down resistor connecting it to ground. The resistor makes sure the GPIO pin is always connected, but will be “overridden” by the "un-resisted” HIGH signal when pressed. A circuit diagram is shown below.



Note that we also include a current-limiting resistor between the button and the GPIO pin. Before the RPi is configured, the GPIO may be set to output—pressing the button in this state might create a direct connection between power and ground without the protective resistor. *Also note that the RPi device contains*

built-in pull-up and pull-down resistors on each GPIO pin. These can be controlled through software and used in lieu of the physical 10k resistor.

Getting Started

A skeleton project (**metronome.zip**) has been prepared to get you started with the RPi GPIO functionality. The program will continuously blink an LED as long as a push button is held down. Unzip the files included with the assignment onto your RPi. The sources are heavily commented with some insight on how to complete this lab. You will also need to install some additional packages:

```
sudo apt install wiringpi libcpprest-dev
```

Be sure to specify the LED and button pin numbers that correspond to your physical circuit setup. Once configured, you can compile the program with the following command (on your Raspberry Pi, inside the cloned folder):

```
mkdir build && cd build
cmake ..
make
```

The program can then be run using this command:

```
./main
```

Note that one subsequent builds, only `make` needs to be run. The sample also exposes a simple REST service that should be accessible through the same IP you use to SSH into your device, on port `8080`.

Implementing the Metronome

Note that the base project also contains the header `metronome.hpp`. This contains the definitions for the functionality of the metronome itself. You will need to write the implementation of this class. The metronome should `start_timing()` in the “learn” mode, and `stop_timing()` upon return to the “play” mode. While timing, each press of the button should record the current time. The `m_beats` array can be used to implement a running average—only the most recent *N* taps are recorded.

Part Two — HTML Dashboard Application

For this part of the lab, you will create a web dashboard application to interact with your embedded service. The application will allow you to easily interact with your REST endpoints.

Therefore, your web application should allow you to:

- Refresh the current BPM, along with the minimum and maximum.
- Set a new BPM on the device.
- Reset the current minimum and maximum BPM.

Getting Started

The dashboard should be implemented as a simple HTML and JavaScript project. It should present some simple styling to visualize the BPM values and allow the user to interact with the API through a set of buttons. No specific format or layout is necessary, but the user should be able to interact with all endpoints offered by the RPi through the dashboard.

Submission Details

You will submit a *.zip* file containing a folder for each part. For part one, **delete** the `build` folder containing your compilation artifacts, and rename the top level

`metronome` folder to `part1`. For part two, name the directory containing your HTML, JavaScript, and any supporting files `part2`. The final *.zip* submission should contain these two folders.