Miami University

Capstone Project

TTL Output User Manual

Authors:
John Gora

Supervisor:
Dr. Karthik Vishwanath

August 1, 2017

MIAMI UNIVERSITY
OXFORD, OH • EST. 1809

1 Introduction

The purpose of this code is to produce Transistor-transistor logic (TTL) signals that can be used to trigger devices that accept these types of serial inputs. These pulses are able to be automated in order to collect data for extended periods of time. Also, the RPi will output multiple pulses simultaneously, which allows the user to sync multiple TTL devices to collect data together! The TTL signals will be produced using a Raspberry Pi (RPi). This device was chosen because of the versatile nature of the device, the cheap cost (around $25) and the ease of programmability of voltage creating pins using Python. The RPi seems a perfect fit for creating and sending TTL signals, and this guide explains how to set-up the RPi, how to get the needed code onto the Rpi, and how to use the program to ultimately send the desired pulses to a device that accepts TTL pulses. Some examples of such devices are lasers, shuttered light sources, filter shutters, imaging devices, printers, and many other electronic devices that need to communicate with other devices.

We will be using an Oceanoptics Halogen Light Source with TTL shutter (HL-2000-FHSA) as the example device, in order to explain how to communicate with devices. Also, the RPi used in this manual is a RPi 1 Model B, but any RPi will work. The first thing that needs to be done is actually getting our RPi up and running, which we will explain in the first section.

2 Raspberry Pi Configuration

Congratulations! You now have a multipurpose tool that can be used for countless projects and uses. The RPi's 26 General Purpose Input-Output (GPIO) pins (or 40 pins, depending on the model of RPi you bought) are now waiting to do your bidding. Lets get right into how you can program them to automate the sending of TTL signals.

2.1 What You Need

What you will need to set-up your Raspberry Pi is a RPi, a RPi power cable, a clean, formatted 8GB+ SD card (if you have a used SD card and you want to wipe it, use this link on SD card reformatting), a laptop with Wi-fi, an ethernet cable or a Wi-Fi dongle, a HDMI cord, a mouse, a keyboard, and a device that supports HDMI inputs (such as a TV or monitor).

2.2 Setting Up Your Raspberry Pi

So you have all your equipment, but how do we power on the RPi and get started with the code? The first thing you have to do is install the operating system on the RPi. This is done through a package provided by the RPi manufacturers, called New Out Of Box Software, or "NOOBS." This is downloaded for free from this website. After downloading the ZIP file, unzip it, and copy all the individual files (not the folder itself) onto your formatted SD card. After these files transfer, you are ready to plug the SD card in and power on your RPi. However, you wont be able to see the video output. That's where the next step comes in.

2.3 VNC Viewer/Secure Shell (SSH)

Now in order to install the operating system (OS), we will need a monitor with an HDMI input, but we don't want to be reliant on a monitor at all times. This is where using Secure Shell and VNC viewer comes in. These programs will allow us to view and manipulate the programs on the RPi using a laptop or other computer that is connected to a network. This allows us to remotely control our RPi, which has many beneficial uses.

So, to begin, plug your mouse, keyboard, SD card and HDMI cable into the RPi. Then, plug this into your HDMI-supported monitor. Power on the RPi, and you should see the OS download screen appear. Choose to download Raspbian, and begin the install.

After the OS has been downloaded, the desktop will pop up. In order to begin using SSH and VNC, you must first find a network, either via a Wi-fi dongle or an ethernet cable, and enabale SSH/VNC viewing. This is done by going to the command line on your RPi, and typing the command `sudo raspi-config`. This will pull up a menu. Navigate to "5 Interfacing Options," and then enable "SSH" and "VNC" options. This will allow you to remotely control your RPi!

Next, install PuTTY and VNC-Viewer on your laptop/desktop computer you want to use to control your RPi. Make sure your RPi is connected to a network, launch PuTTY, and type "raspberrypi.local" into the host name box. If all goes according to plan, you will be prompted to login. The default login/password for a new RPi is login: pi with password: raspberry. Login to your pi, and change the password if desired, using the command `passwd`. In order to see the interface for the desktop of the RPi, use the VNC-Viewer. Accessing the VNC is similar to PuTTY; just enter "raspberrypi.local" into the server address and login with your user name and password. You can now access all parts of your RPi remotely, without need of an extra mouse, keyboard, or monitor!

2.4 Downloading the Code

Now we can view our RPi desktop, and access the command line via SSH. Now it's time to download our code and get our program up and running. This step is pretty straightforward. The Python code is contained in an online Github repository. To access and download this package, just type the command

```
git clone https://github.com/goraj41/TTLoutput.git.
```

into the command window and hit "Enter." This should download a folder on your RPi titled "TTLoutput." This folder contains the .py file in which the code is contained. If you are unfamiliar with Linux commands, to navigate into this folder, type `cd TTLoutput` into the command line. Use the command `ls` to see what is contained in the current folder. You should be able to see the TTLoutput.py file.

3 Device Setup

Now you have the code, are in the correct folder, and are ready to set up the devices that you want to send signals to. Keep in mind that the RPi can send signals to multiple devices!

3.1 What You Need

All you need to deliver your signals from the RPi to your device(s) is a few female-to-male jumper cables (that will be able to reach your devices input), a few male-to-male jumper cables, and a small breadboard.

3.2 Wiring

The wiring section is pretty straightforward. The first thing that you will need to know is where your input is on each device. Using the user manual provided with the device, you should be able to locate the pinout diagram. This diagram should show where the TTL input is accepted. For instance, for our example device, the Halogen Light, the manual (linked here) contains a diagram on page 14 that shows the TTL input pin as pin 13 and the ground pin as pin 10. Next, you will need to know the output pins for each device on the RPi. It is currently programmed so that the **light-emitting device output is pin 12** and the **detector output is pin 7**. Also, the **ground pin is pin 6**, which needs to be connected to the ground input on each device. If you wish to change these output pins for any reason, see Section 3.3: Changing GPIO Output Pins.

To set up the wiring needed, use a female-to-male cable from pins 7, 12, and ground on the RPi to a designated spot on the breadboard. Next, wire the ground from the breadboard to each ground input on your device(s), using male-to-male jumper cables. Do the same for each TTL input (see Figure below). Now your devices are wired and ready to go!

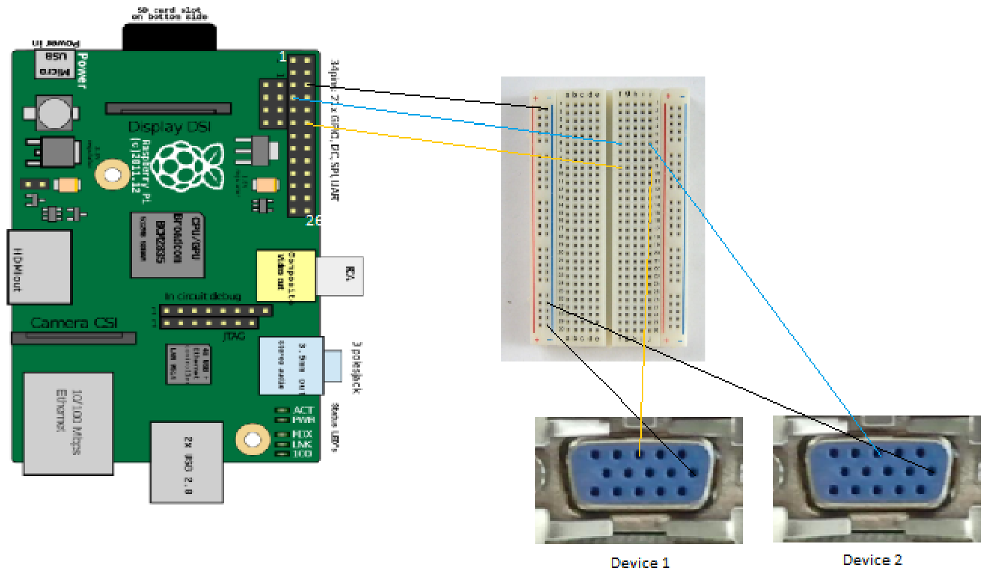

Figure 1: Schematic showing the wiring needed for two devices. The black wires indicate the ground wires, blue wires show the detector wires from GPIO pin 7, and orange show the lamp wires from pin 12. The input pins for the female serial input on each device are found using the user manual for each device.

3.3 Changing GPIO Output Pins

If, for any reason, you wish to change the output pins designated for the detector and lamp from 7 and 12 to something else, follow the steps outlined here.

First, navigate to the folder that contains TTLoutput.py. To do this, see Section 4.1.1: Navigating to the Correct Folder. Once in the correct directory, use the command

```
nano TTLoutput.py
```

This will allow you to edit the code contained in the .py file. Navigate to the `GPIO.setup` section of code, and change the numbers 7 and 12 to the desired pins. Continue changing each 7 and 12 in the `GPIO.output` section of the code to the desired pin numbers throughout the code. This should successfully alter the pins that output the TTL signals.

4 Code Implementation

The last thing that needs to be done is to supply the program with inputs and actually run the code. This section will explain how to deliver the desired number and length of pulses to your device(s), as well as different options there are for collecting data.

4.1 Delivering the Pulse

4.1.1 Navigating to the Correct Folder

Make sure that you are in the TTLoutput folder. You can tell what folder you are in by looking at the current line of the command line. If it has a `/TTLoutput $` then you are in the correct folder. If not, use the command `cd` to navigate to the correct folder (or the command “`cd ..`” to navigate back a folder). To check what programs are contained in the current directory, use the command `ls`. After navigating to the correct folder, it is time to run the program.

4.1.2 User Inputs

To begin the program, use the command

```
python3 TTLoutput.py
```

This will prompt a user input, asking for the number of times to output pulses, call it N. Enter the desired number and hit enter. Next, the program asks for the desired length of cycle, or T. This is the total time of each cycle, including the time the device is on and off (see Figure 1). Next is the time in the cycle in which the device turns on, or T_{on}. The desired time to turn off in the cycle, of T_{off}, is similar, except that the device turns off at this time in the cycle. For example, if you wanted to create a series of 1 second pulses with 1 second in between each pulse, you would enter $T = 2$, $T_{on} = 0$, and $T_{off} = 1$. This would turn on the pulse at the beginning of the cycle, leave it on until T_{off} at 1 second, and then leave the pulse off until the end of the 2 second cycle. If you've done everything else correctly, this should deliver your desired pulses to each of your devices! The last input is whether you want to collect background for each collection. If you have a detector that you wish to take background before you capture your data, then type "y" when prompted. This is explained further in the next section.

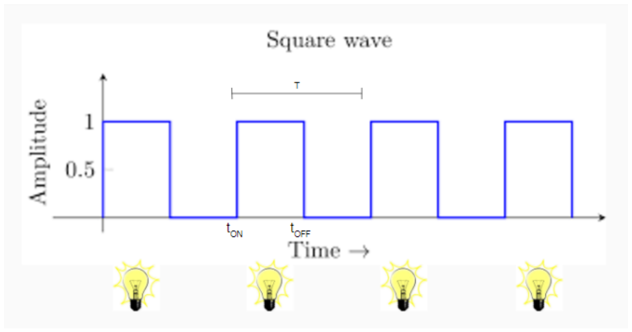

Figure 2: Voltage output of TTLoutput program. The light bulbs represent when the detector/lamp is on. This example shows $T_{on} = 0$ and T_{off} coming halfway through the total pulse time, T.

4.2 Background Collection

This last section covers background collection and how to use it. If you have a device that measures things such as light, and you are not working in a dark room, you may want to take a sample of the background light in order to subtract it from your desired data sample. If this is the case, type "y" when prompted. This will result in your collection device taking background data before each sample is collected.

5 Conclusion

If you need a program that uses a cheap and easy device in order to control your TTL devices, then this is the program for you. Using a Raspberry Pi to deliver your signal is an automated way to collect more accurate data faster, and in a more controlled environment. This program will work for any programmable TTL device, such as lasers, lamps, spectrometers, and other types of imaging devices. The cheap cost of the RPi is another positive, as it is easily accessible and versatile. This guide explains how to use your RPi in order to create a simple, fast, and accurate TTL machine. I hope you found this guide useful and informative! Thanks for reading.

6 Contact Me

If you have any questions or suggestions for improving the program, code, or device used in this guide, contact me at gorajt@miamioh.edu.