**Programming the Physical World**

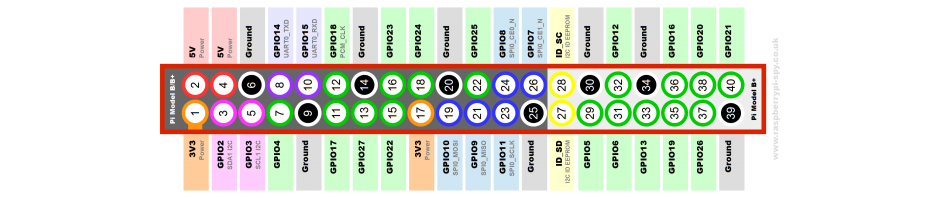
**Morse Code Translator**

In this lab we are going to write a program that interfaces with computer hardware to make something physical occur, specifically blink Morse code on a Light Emitting Diode (LED).

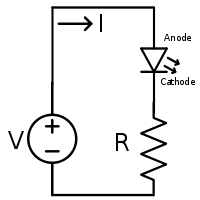
For this activity we will require the following:

1. Raspberry Pi
2. Two male/female jumper wires
3. Solderless breadboard
4. LED
5. 330 Ω resistor
6. HDMI to DVI cable

The hardware we will be using will be the Raspberry Pi which has 18 GPIO (General Purpose Input Output) pins which can be easily programmed to send output signals to or receive input signals.



For this activity, we will be using pins 6 and 12. These correspond to Ground and GPIO Pin 18 respectively. Here is a diagram of an LED setup. To get the LED to light we need approximately 2.1 volts across it. The purpose of the resistor is to limit the current through the LED. Pin 18 is capable of supplying 5 volts. If we do not use a current limiting resistor, the LED will light brightly for a while and then go dark. For a nominal brightness we want about 10 mA to flow through the LED. This means a 330 Ω resistor should do. Actually we should be OK with a resistor or 180 Ω to 550 Ω.



We will be using a solderless breadboard to hook up the components. Pin 6 (Ground) should hook to one end of the resistor. Pin 12 (GPIO 18) should hook to the Anode of the LED (the longer lead).

**Hardware Assembly Procedure**

1. Add 330 Ω resistor to the breadboard
   1. Measure the resistor and record its value \_\_\_\_\_\_\_\_\_ Ω
2. Add LED to breadboard so Cathode of LED (shorter lead) is connected to resistor
3. Connect Pin 6 on Raspberry Pi GPIO header to unconnected lead of resistor
4. Connect Pin 12 on Raspberry Pi GPIO header to Anode of LED (longer lead)

Note: If you have the LED hooked up backwards it will not light and you can correct this by reversing its connections.

Now we are ready to program the GPIO pins. In order to do this we will write a short Python script. The RPi.GPIO library is the software that allows us easy access to the hardware. The Operating System running on the Raspberry Pi is a variation of Debian Linux. What this means to you is to directly access the hardware, you must have root. This means we must run IDLE as root. Linux has a mechanism to do this called sudo which allows you to run programs with the security privileges of another user (typically the superuser). This is a concatenation for superuser do. Start IDLE as follows:

1. Open the Run Dialog (Start|Run)
2. Type sudo idle
3. Hit Enter

Once you have the IDLE shell opened, open a new window (File|New Window)

Note: Make a folder on your desktop called MyPython for storing your programs.

First Experiment

Type in and run the following script:

#Turn LED on

importRPi.GPIO as GPIO

GPIO.setwarnings(False) #Disable warnings for any not reset pins

GPIO.setmode(GPIO.BCM) #Allow ARM processor to access GPIO

GPIO.setup(18, GPIO.OUT) #set pin 18 to output

GPIO.output(18, True) #Turn LED on

With the LED on, measure the voltage across the resistor and the LED

Vresistor = \_\_\_\_\_\_\_ volts

VLED = \_\_\_\_\_\_\_\_\_ volts

Vtotal = \_\_\_\_\_\_\_\_\_ volts

What was the value of the resistor.

R = \_\_\_\_\_\_\_\_\_ Ω

Write a Python program to report the current and number of electrons/second flowing through the LED.

Note 1: Current = Voltage / Resistance

Note 2: 1 Ampere of current = 6.241 x 1018 electrons/second

Current = \_\_\_\_\_\_\_\_\_\_ mA = \_\_\_\_\_\_\_\_\_\_ electrons/second

Now let's try some other experiments:

Type in the following script:

#Simple Blink

importRPi.GPIO as GPIO

import time

GPIO.setwarnings(False) #Disable warnings for any not reset pins

GPIO.setmode(GPIO.BCM) #Allow ARM processor to access GPIO

GPIO.setup(18, GPIO.OUT) #set pin 18 to output

#Make LED blink

while True:

GPIO.output(18, True)

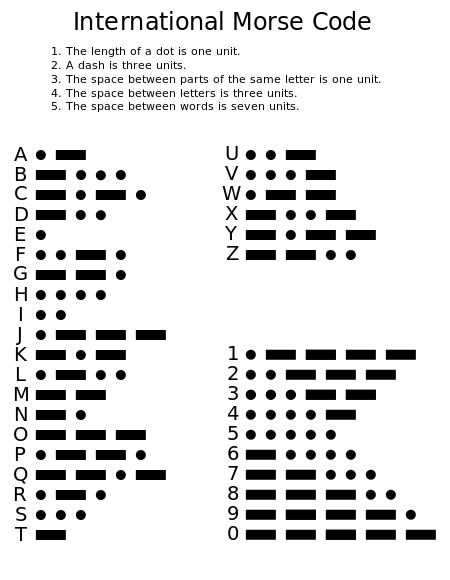
time.sleep(1)

GPIO.output(18, False)

time.sleep(.1)

Run you program and you should see the LED blinking.

Now let’s do something more interesting. We are going to use the International Morse Code to make our program send an SOS.



Here is the code to make this happen.

#SOS code generator

importRPi.GPIO as GPIO

import time

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BCM)

GPIO.setup(18, GPIO.OUT)

while True:

#S

fori in range(3):

GPIO.output(18, True)

time.sleep(.1)

GPIO.output(18, False)

time.sleep(.1)

time.sleep(0.5)

#O

fori in range(3):

GPIO.output(18, True)

time.sleep(1)

GPIO.output(18, False)

time.sleep(.1)

time.sleep(0.5)

#S

fori in range(3):

GPIO.output(18, True)

time.sleep(.1)

GPIO.output(18, False)

time.sleep(.1)

time.sleep(1.5)

Finally, we are going to create a program that will transmit any message we type in as Morse Code.

importRPi.GPIO as GPIO

import time

GPIO.setwarnings(False)

GPIO.setmode(GPIO.BCM)

GPIO.setup(18, GPIO.OUT)

defcodeGen(s):

output = ''

s = s.lower() #put string in lower case to reduce dictionary size

fori in range(len(s)):

if s[i] == '\n':

output += '\n'

continue

output += convert(s[i])

output += ' '

print output

fori in range(len(output)):

if output[i] == '.':

dot()

if output[i] == '-':

dash()

if output[i] == ' ':

time.sleep(0.1)

def convert(letter):

#International Morse Code dictionary

table = {'a':'.-', 'b':'-...', 'c':'-.-.', 'd':'-..', 'e':'.',

'f':'..-.', 'g':'--.', 'h':'....', 'i':'..', 'j':'.---',

'k':'-.-', 'l':'.-..', 'm':'--', 'n':'-.', 'o':'---',

'p':'.--.', 'q':'--.-', 'r':'.-.', 's':'...', 't':'-',

'u':'..-', 'v':'...-', 'w':'.--', 'x':'-..-', 'y':'-.--',

'z':'--..', ' ':' ', '1':'.----', '2':'..---',

'3':'...--', '4':'....-', '5':'.....', '6':'-....',

'7':'--...', '8':'---..', '9':'----.', '0':'-----'}

if letter in table.keys():

return table[letter]

else: #handle other characters where code does not exist in dictionary

return '' # 2 single quotes

def dot():

GPIO.output(18, True)

time.sleep(0.1)

GPIO.output(18, False)

time.sleep(0.1)

def dash():

GPIO.output(18, True)

time.sleep(1)

GPIO.output(18, False)

time.sleep(0.1)

phrase = raw\_input("Enter a phrase: ")

codeGen(phrase)

Notes on codeGen function:

Build the output string sending each character in s to the convert function and getting the Morse code for that character. Put three spaces between each character to be consistent with the International Morse Code Standard. When s has been converted, print the result to the screen.

Display each character by flashing LED in the Morse Code string. If the character is '.', call dot(). If the character is a '-', call dash(). If the character is a space ' ', sleep(0.1)