

Lab Manual for IECE 553/453  
Cyber-Physical Systems  
Fall 2020

**Prof. Dola Saha**  
Assistant Professor  
Department of Electrical & Computer Engineering  
University at Albany, SUNY

# Chapter 1

## Setup Headless Raspberry Pi

This tutorial is to setup the Raspberry Pi without requirement of any keyboard, mouse or monitor. You should be able to login to the Pi from your laptop or desktop using your home Wi-Fi.

### 1.1 Boot up Raspberry Pi for the first time

1. Insert a microSD card in your laptop or desktop.
2. Download and install Raspberry Pi Imager (<https://www.raspberrypi.org/downloads/>).
3. Once installed, run the Raspberry Pi Imager program. Choose 'Raspberry Pi OS (32 bit)' from the OS menu. Click 'Choose SD card' and select your card from the menu.
4. Click Write. This process will take several minutes as Raspberry Pi Imager downloads Raspberry Pi OS and burns it to your microSD card.
5. Create an empty file in boot partition of the Micro SD card and name it `ssh` without any extension. Use commands `touch` in Linux/OSX or `fsutil` in Windows. This enables `ssh` in Raspberry Pi.
6. To setup a Wi-Fi connection on your headless Raspberry Pi, create a text file in the same folder of the microSD card called `wpa_supplicant.conf`. The content of the file is below. Change the text below based on the SSID and password of your home Wi-Fi router.

```
country=US
ctrl_interface=DIR=/var/run/wpa_supplicant
GROUP=netdev
update_config=1
network={
scan_ssid=1
ssid="my_wifi_ssid"
psk="my_wifi_password"
}
```

7. Unmount/eject the Micro SD card from your computer.
8. Insert the Micro SD card in the Raspberry Pi and power it. This will boot up in Raspberry Pi OS with SSH being enabled.

### 1.2 Login to your Raspberry Pi and setup hostname

1. Once booted up, your Raspberry Pi will connect to your Home Router using Wi-Fi connection.
2. From your laptop (also connected to your Home Router by wireless connection), use `ssh` to connect to the Raspberry Pi. The default values are:

```
hostname: raspberrypi
username: pi
password: raspberry
```

If you are using Linux or OSX, you can use the following command to ssh in with X-forwarding enabled:

```
ssh -Y pi@raspberrypi.local
```

If you are using Windows, choose PuTTY to ssh in with the same credentials.

3. Expand the Filesystem: Use the following command: `sudo raspi-config`. Choose Option 7, Advanced Options, then choose A1 Expand Filesystem. Save the changes and reboot when prompted.
4. At this point, you may choose to create a new user with unique ID and password and delete the default one, just like you would do in a Linux system.

## 1.3 Initial setup

1. Make sure that the Raspberry Pi is up to date with the latest versions of Raspberry Pi OS: (this is a good idea to do regularly, anyway).

```
sudo apt-get update
```

```
sudo apt-get upgrade
```

2. Setup RPi libraries:

The RPi.GPIO module is installed by default in Raspberry Pi OS Desktop image. To make sure that it is at the latest version:

```
sudo apt-get install python-rpi.gpio python3-rpi.gpio
```

3. Setting up GPIO Zero libraries,

GPIO Zero is installed by default in the Raspberry Pi OS image, and the Raspberry Pi Desktop image. However it can be installed or updated using,

```
sudo apt install python3-gpiozero
```

4. Setting up Exploring Raspberry pi libraries,

```
sudo apt-get install git
```

```
git clone https://github.com/derekmolloy/exploringrpi.git
```

5. WiringPi is pre-installed with standard Raspberry Pi OS systems. However it can be installed or updated using,

```
sudo apt-get install wiringpi.
```

## 1.4 Shutdown and Restart

After every use, make sure to properly shutdown the Pi using the command:

```
sudo shutdown -h now.
```

To reboot the Pi, use the following command:

```
sudo reboot
```

# Chapter 2

## Basic Input and Output Using Pseudo Filesystem

### 2.1 The First Circuit

We will use GPIO to Breadboard Interface Board with GPIO Ribbon Cable to connect the Raspberry PI for ease of use. Figure 2.1 shows the connection. Make sure the red line of the breadboard is aligned with positive voltage in the interface board, as shown in Figure 2.2.

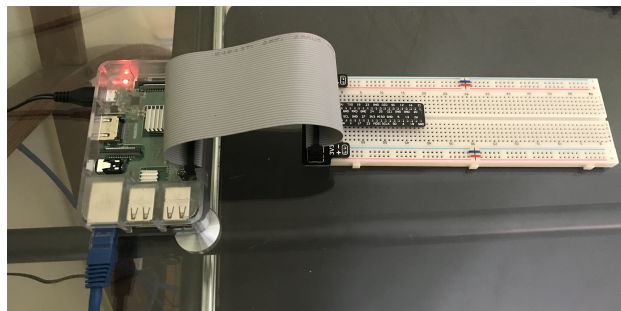
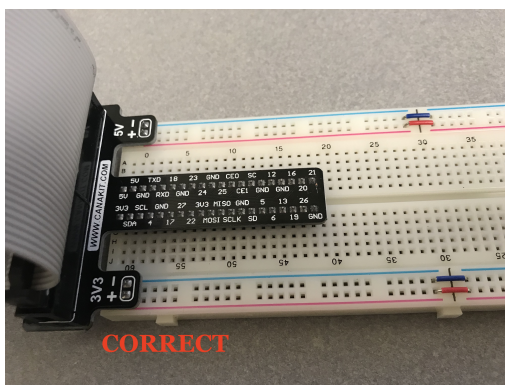
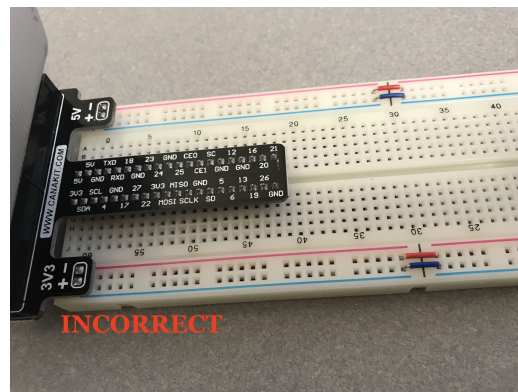


Figure 2.1: GPIO to Breadboard Interface Board and Ribbon Cable.



(a) Red line matching positive.

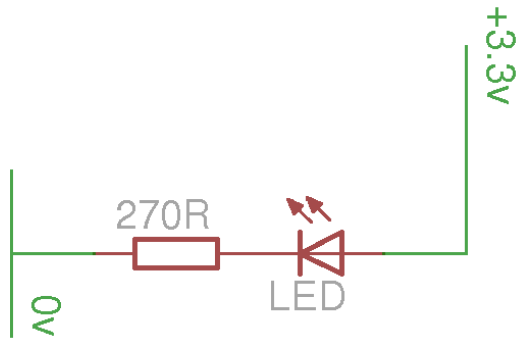


(b) Red line not matching positive.

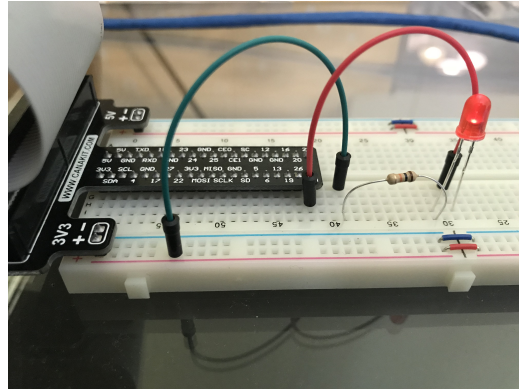
Figure 2.2: Breadboard.

The first circuit that we will work on is shown in Figure 2.3. It is the simplest circuit, where the circuit is always connected to +3.3V line. This will keep the LED turned on until the circuit is disconnected.

Once you complete the circuit and LED is turned on, you have completed the first circuit. Congratulations!



(a) Schematic Diagram.



(b) Circuit on breadboard.

Figure 2.3: LED Resistor Circuit.

## 2.2 Programming The First Circuit Using `sysfs`

We will use the GPIO pins to program the controlling of the LED. Our Raspberry Pi uses Raspbian based on Debian and optimized for the Raspberry Pi hardware. We will use `sysfs` to access the kernel space for controlling the GPIO pins. Change the circuit to get power from GPIO pin 4 instead of 3.3V as shown in figure 2.4.

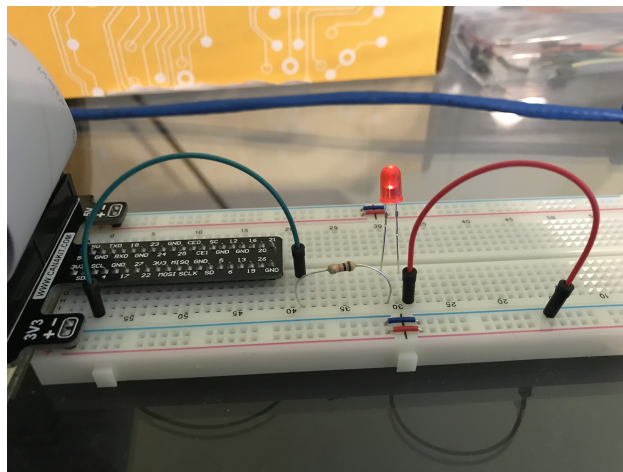


Figure 2.4: LED Resistor Circuit connected to GPIO Pin 4.

The steps for controlling the GPIO pin using `sysfs` is given below.

1. Become the Sudo user to access `sysfs`.

```
dsaha@sahaPi:~ $ sudo su
```

2. Go to the GPIO folder and list the contents

```
root@sahaPi:/home/dsaha# cd /sys/class/gpio/
root@sahaPi:/sys/class/gpio# ls
export gpiochip0 gpiochip128 unexport
```

3. Export gpio 4

```
root@sahaPi:/sys/class/gpio# echo 4 > export
root@sahaPi:/sys/class/gpio# ls
export gpio4 gpiochip0 gpiochip504 unexport
```

#### 4. Go to the gpio4 folder and list contents

```
root@sahaPi:/sys/class/gpio# cd gpio4/
root@sahaPi:/sys/class/gpio/gpio4# ls
active_low device direction edge power subsystem uevent value
```

#### 5. Set direction (in or out) of pin

```
root@sahaPi:/sys/class/gpio/gpio4# echo out > direction
```

#### 6. Set value to be 1 to turn on the LED

```
root@sahaPi:/sys/class/gpio/gpio4# echo 1 > value
```

#### 7. Set value to be 0 to turn off the LED

```
root@sahaPi:/sys/class/gpio/gpio4# echo 0 > value
```

#### 8. Check the status (direction and value) of the pin

```
root@sahaPi:/sys/class/gpio/gpio4# cat direction
out
root@sahaPi:/sys/class/gpio/gpio4# cat value
0
```

#### 9. Ready to give up the control? Get out of gpio4 folder and list contents, which shows gpio4 folder

```
root@sahaPi:/sys/class/gpio/gpio4# cd ../
root@sahaPi:/sys/class/gpio# ls
export gpio4 gpiochip0 gpiochip128 unexport
```

#### 10. Unexport gpio 4 and list contents showing removal of gpio4 folder

```
root@sahaPi:/sys/class/gpio# echo 4 > unexport
root@sahaPi:/sys/class/gpio# ls
export gpiochip0 gpiochip504 unexport
```

## 2.3 The First Circuit Using Bash, Python and C

Listing 2.1<sup>1</sup> shows the bash script to turn on or off the LED using GPIO pins. Note that each step in the script is same as shown in §2.2. Listings 2.2 and 2.3 show similar procedure in Python and C languages respectively.

Listing 2.1: LED code in Bash script

```
#!/bin/bash
# A small Bash script to turn on and off an LED that is attached to GPIO 4
# using Linux sysfs. Written by Derek Molloy (www.derekmolloy.ie) for the
# book Exploring Raspberry PI
LED_GPIO=4    # Use a variable -- easy to change GPIO number

# An example Bash functions
function setLED
{ # $1 is the first argument that is passed to this function
  echo $1 >> "/sys/class/gpio/gpio$LED_GPIO/value"
}

# Start of the program -- start reading from here
if [ $# -ne 1 ]; then # if there is not exactly one argument
  echo "No_command_was_passed._Usage_is:_bashLED_command,"
  echo "where_command_is_one_of:_setup,_on,_off,_status_and_close"
  echo -e "e.g.,_bashLED_setup,_followed_by_bashLED_on"
  exit 2 # error that indicates an invalid number of arguments
fi
```

<sup>1</sup>This part of the lab follows the book "Exploring Raspberry Pi: Interfacing to the Real World with Embedded Linux", by Derek Molloy, Wiley, ISBN 978-1-119-18868-1, 2016.

```

echo "The_LED_command_that_was_passed_is:_$1"
if [ "$1" == "setup" ]; then
    echo "Exporting_GPIO_number_$1"
    echo $LED_GPIO >> "/sys/class/gpio/export"
    sleep 1    # to ensure gpio has been exported before next step
    echo "out" >> "/sys/class/gpio/gpio$LED_GPIO/direction"
elif [ "$1" == "on" ]; then
    echo "Turning_the_LED_on"
    setLED 1   # 1 is received as $1 in the setLED function
elif [ "$1" == "off" ]; then
    echo "Turning_the_LED_off"
    setLED 0   # 0 is received as $1 in the setLED function
elif [ "$1" == "status" ]; then
    state=$(cat "/sys/class/gpio/gpio$LED_GPIO/value")
    echo "The_LED_state_is:_$state"
elif [ "$1" == "close" ]; then
    echo "Unexporting_GPIO_number_$LED_GPIO"
    echo $LED_GPIO >> "/sys/class/gpio/unexport"
fi

```

Listing 2.2: LED code in Python

```

#!/usr/bin/python2
# A small Python program to set up GPIO4 as an LED that can be
# turned on or off from the Linux console.
# Written by Derek Molloy for the book "Exploring Raspberry Pi"

import sys
from time import sleep
LED4_PATH = "/sys/class/gpio/gpio4/"
SYSFS_DIR = "/sys/class/gpio/"
LED_NUMBER = "4"

def writeLED ( filename, value, path=LED4_PATH ):
    "This_function_writes_the_value_passed_to_the_file_in_the_path"
    fo = open( path + filename, "w" )
    fo.write(value)
    fo.close()
    return

print "Starting_the_GPIO_LED4_Python_script"
if len(sys.argv) != 2:
    print "There_is_an_incorrect_number_of_arguments"
    print "__usage_is:__pythonLED.py_command"
    print "__where_command_is_one_of_setup,_on,_off,_status,_or_close"
    sys.exit(2)
if sys.argv[1]=="on":
    print "Turning_the_LED_on"
    writeLED (filename="value", value="1")
elif sys.argv[1]=="off":
    print "Turning_the_LED_off"
    writeLED (filename="value", value="0")
elif sys.argv[1]=="setup":
    print "Setting_up_the_LED_GPIO"
    writeLED (filename="export", value=LED_NUMBER, path=SYSFS_DIR)
    sleep(0.1);
    writeLED (filename="direction", value="out")
elif sys.argv[1]=="close":
    print "Closing_down_the_LED_GPIO"

```

```

writeLED (filename="unexport", value=LED_NUMBER, path=SYSFS_DIR)
elif sys.argv[1]=="status":
    print "Getting_the_LED_state_value"
    fo = open( LED4_PATH + "value", "r")
    print fo.read()
    fo.close()
else:
    print "Invalid_Command!"
print "End_of_Python_script"

```

Listing 2.3: LED code in C

```

/** Simple On-board LED flashing program - written in C by Derek Molloy
 *   simple functional struture for the Exploring Raspberry Pi book
 *
 *   This program uses GPIO4 with a connected LED and can be executed:
 *       makeLED setup
 *       makeLED on
 *       makeLED off
 *       makeLED status
 *       makeLED close
 */

#include<stdio.h>
#include<stdlib.h>
#include<string.h>

#define GPIO_NUMBER "4"
#define GPIO4_PATH "/sys/class/gpio/gpio4/"
#define GPIO_SYSFS "/sys/class/gpio/"

void writeGPIO(char filename[], char value[]){
    FILE* fp; // create a file pointer fp
    fp = fopen(filename, "w+"); // open file for writing
    fprintf(fp, "%s", value); // send the value to the file
    fclose(fp); // close the file using fp
}

int main(int argc, char* argv[]){
    if(argc!=2){ // program name is argument 1
        printf("Usage_is_makeLEDC_and_one_of:\n");
        printf("___setup,_on,_off,_status,_or_close\n");
        printf("_e.g._makeLEDC_on\n");
        return 2; // invalid number of arguments
    }
    printf("Starting_the_makeLED_program\n");
    if(strcmp(argv[1], "setup")==0){
        printf("Setting_up_the_LED_on_the_GPIO\n");
        writeGPIO(GPIO_SYSFS "export", GPIO_NUMBER);
        usleep(100000); // sleep for 100ms
        writeGPIO(GPIO4_PATH "direction", "out");
    }
    else if(strcmp(argv[1], "close")==0){
        printf("Closing_the_LED_on_the_GPIO\n");
        writeGPIO(GPIO_SYSFS "unexport", GPIO_NUMBER);
    }
    else if(strcmp(argv[1], "on")==0){
        printf("Turning_the_LED_on\n");
        writeGPIO(GPIO4_PATH "value", "1");
    }
}

```



```

}
else if (strcmp(argv[1], "off")==0){
    printf("Turning_the_LED_off\n");
    writeGPIO(GPIO4_PATH "value", "0");
}
else if (strcmp(argv[1], "status")==0){
    FILE* fp; // see writeGPIO function above for description
    char line[80], fullFilename[100];
    sprintf(fullFilename, GPIO4_PATH "/value");
    fp = fopen(fullFilename, "rt"); // reading text this time
    while (fgets(line, 80, fp) != NULL){
        printf("The_state_of_the_LED_is_%s", line);
    }
    fclose(fp);
}
else{
    printf("Invalid_command!\n");
}
printf("Finished_the_makeLED_Program\n");
return 0;
}

```

## 2.4 Simple Circuit Using Python Libraries

### 2.4.1 RPi Library

In this section, we will use RPi Python library to demonstrate similar LED switching functions. Listing 2.4 shows a simple code in Python for GPIO pin manipulation. Check the website [<https://sourceforge.net/projects/raspberry-gpio-python/>] to learn other functions available through this library. Note that the current release does not support SPI, I2C, 1-wire or serial functionality on the RPi yet.

Listing 2.4: LED code using RPi Library in Python

```

import RPi.GPIO as GPIO
from time import sleep

ledPin = 4 # GPIO Pin Number, where LED is connected

GPIO.setmode(GPIO.BCM) # Broadcom pin-numbering scheme
GPIO.setup(ledPin, GPIO.OUT) # LED pin set as output

GPIO.output(ledPin, GPIO.HIGH) # Turn the LED on
sleep(1) # Sleep for 1 sec
GPIO.output(ledPin, GPIO.LOW) # Turn the LED off
GPIO.cleanup() # Clean up at the end of the program

```

### 2.4.2 GPIOZero Library

Another efficient Python library is gpiozero [<https://gpiozero.readthedocs.io/en/stable/>]. It provides a simple interface to GPIO devices with Raspberry Pi. It was created by Ben Nuttall of the Raspberry Pi Foundation, Dave Jones, and other contributors. Listing 2.5 shows a simple code for switching on and off the LED.

Listing 2.5: LED code using GPIOZero Library in Python

```

from gpiozero import LED
from time import sleep

led = LED(4) # GPIO Pin Number
led.on() # Turn on LED
sleep(1) # Sleep for 1 sec
led.off() # Turn off LED

```

## 2.5 Use GPIO Pins for Input

In the next experiment, we will use GPIO Pin as an input. The circuit is shown as in figure 2.5. A button switch, when pressed, will be detected by the program and a message will be printed accordingly. In this case, the process continuously polls the status of the input pin. Listings 2.6 and 2.7 shows the code using RPi and gpiozero libraries.

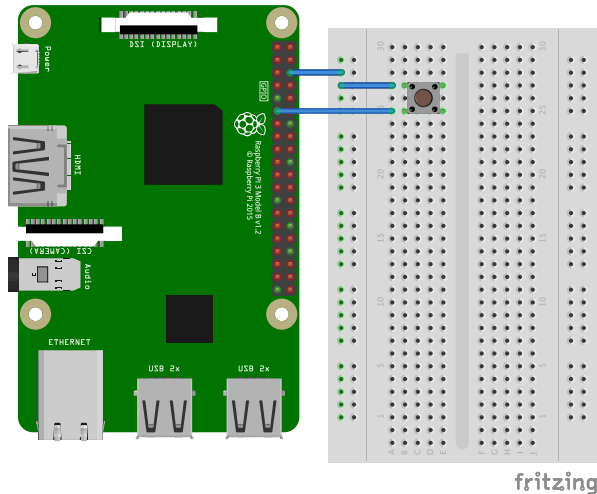


Figure 2.5: Button Switch connected to GPIO Pin.

Listing 2.6: Button Press detection code using RPi Library in Python

```
import RPi.GPIO as GPIO
import time

buttonPin=17    # GPIO Pin Number where Button Switch is connected

GPIO.setmode(GPIO.BCM)        # Broadcom pin-numbering scheme
GPIO.setup(buttonPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
# Button pin set as input

while True:
    input_state = GPIO.input(buttonPin) # Monitor continuously
    # Get the input state
    if input_state == False:           # Check status
        # Print
        print('Button_Pressed')
        # Sleep before checking again
        time.sleep(0.2)
```

Listing 2.7: Button Press detection code using GPIOZero Library in Python

```
from gpiozero import Button
import time

button = Button(17) # GPIO Pin Number where Button Switch is connected

while True:
    # Monitor continuously
    if button.is_pressed:
        # Check Status
        # Print
        print("Button_Pressed")
        # Sleep before checking again
        time.sleep(0.2)
```

# Chapter 3

## Basic Input and Output Using Address Map

In this lab, we will use memory mapped I/O to a) change the GPIO pin output and b) control the pull-up/down resistor configuration.

### 3.1 Memory Mapped Input and Output Access

All the instructions below **must be executed with root privilege**.

1. Get the file describing the *Address Mapping* of the system's memory for each physical device, using the command:

```
cat /proc/iomem
```

The output would be an *Address Mapping* of the form

```
3f200000-3f2000b3 : gpio@7e200000
```

The first column displays the memory registers used by each of the different types of memory. The second column lists the kind of memory located within those registers and displays which memory registers are used by the kernel within the system RAM or, if the network interface card has multiple Ethernet ports, the memory registers assigned for each port.

2. `/dev/mem` provides access to the system's physical memory. It is primarily used to access IO memory addresses related to peripheral hardware.

Get the file for direct memory access using the command.

```
wget http://www.lartmaker.nl/lartware/port/devmem2.c
```

If you do not have access to internet on the Raspberry Pi, download the file from URL above to your machine, and copy the contents to the Raspberry Pi.

Compile the code using

```
gcc devmem2.c -o devmem2
```

Usage: `./devmem2 address [ type [ data ] ] address` : memory address to act upon  
type: access operation type :  
[b]yte, [h]alfword, [w]ord data: data to be written

### 3.2 GPIO Pull-up/down Register Control

Pull-down and Pull-up Resistors are used to ensure that the switches do not create floating inputs. We will configure the Internal pull-up/pull-down Resistors using memory based GPIO control. The GPIO Pull-up/down Register controls the actuation of the internal pull-up/down control line to ALL the GPIO pins. This register must be used in conjunction with the 2 GPPUDCLKn registers.

We will change the Pull-up/ pull-down configurations of GPIO pin 4. For this we set bit 4 on the GPPUDCLK0 register, clear the GPPUD register, and then remove the clock control signal from GPPUDCLK0. GPIO4 is bit 4, which is '10000' in binary (0x10<sub>16</sub>).

Get the value of GPIO pin 4 using the `sysfs` file system.

### 3.2.1 Pull-down Configuration

1. GPPUD Enable Pull-down  

```
sudo /home/dsaha/myCode/devmem2 0x3F200094 w 0x01
```
2. GPPUDCLK0 enable GPIO 4  

```
sudo /home/dsaha/myCode/devmem2 0x3F200098 w 0x10
```
3. GPPUD Disable Pull-down  

```
sudo /home/dsaha/myCode/devmem2 0x3F200094 w 0x00
```
4. GPPUDCLK0 disable Clk signal  

```
sudo /home/dsaha/myCode/devmem2 0x3F200098 w 0x00
```
5. cat value  

```
value: 0
```

### 3.2.2 Pull-up Configuration

1. GPPUD Enable Pull-up  

```
sudo /home/dsaha/myCode/devmem2 0x3F200094 w 0x02
```
2. GPPUDCLK0 enable GPIO 4  

```
sudo /home/dsaha/myCode/devmem2 0x3F200098 w 0x10
```
3. GPPUD Disable Pull-up  

```
sudo /home/dsaha/myCode/devmem2 0x3F200094 w 0x00
```
4. GPPUDCLK0 disable Clk signal  

```
sudo /home/dsaha/myCode/devmem2 0x3F200098 w 0x00
```
5. cat value  

```
value: 1
```

## 3.3 GPIO Control

### 3.3.1 GPIO Output Example

In this part of the lab, we will turn on an LED using the Address map. All of the GPIOs can be set to read or write mode, or they can be set to an ALT mode. The mode is set using a 3-bit value. For example, if the 3-bit value is set to 001 then the pin will act as an output. The location at which it should be written is described in the BCM 2837 Manual (Table 6-1). For example, to set GPIO17 to be an output, we can write 001 to the FSEL17 value, which is bits 21, 22, and 23 in the GPFSEL1 register (Table 6-3). Importantly, we need to ensure that we only manipulate those specific 3 bits when we write to GPFSEL1 register. Otherwise, change in any other bits will impact on GPIO10-GPIO19 and likely will change their pin modes.

First, set up the circuit similar to figure 2.3 so that an LED can be turned on via GPIO17. Rest of the values below are specified for GPIO 17. So, it is important to check the pin connection at this point.

1. Read the status of the Function Select Registry  

```
sudo /home/dsaha/myCode/devmem2 0x3F200004
```
2. Set GPIO17 to be an output (Bits 23,22,21 are set to 001 in GPFSEL1 registry, detailed in Tables 6-2 to 6-7)  

```
sudo /home/dsaha/myCode/devmem2 0x3F200004 w 0x200000
```

3. Check the value/status of GPIO17 pi (using the GPVL0 registry, detailed in Table 6-12 and 6-13)  
`sudo /home/dsaha/myCode/devmem2 0x3F200034`
4. Clear the output of GPIO17 (Bit 17 is set to 1 in GPCLR0 registry, detailed in Table 6-12 and 6-13)  
`sudo /home/dsaha/myCode/devmem2 0x3F200028 w 0x20000`
5. Set the output of GPIO17 to 1 (Bit 17 is set to 1 in GPSET0 registry, detailed in Table 6-8 and 6-9)  
`sudo /home/dsaha/myCode/devmem2 0x3F20001C w 0x20000`
6. Set GPIO17 to be an input (Bits 23,22,21 are set to 000 in GPFSEL1 registry)  
`sudo /home/dsaha/myCode/devmem2 0x3F200004 w 0x00`

# Chapter 4

## Analog Output: PWM (Pulse Width Modulation)

The RPi has pulse-width modulation (PWM) capabilities that can provide digital-to-analog conversion (DAC), or generate control signals for motors and certain types of servos. The number of PWM outputs is very limited on the RPi boards. The RPi B+ model has two PWMs (PWM0 & PWM1) output at Pins 12 and 33 (GPIO18, GPIO13).

$$PWM \text{ frequency} = \frac{19.2MHz}{(divisor \times range)} \quad (4.1)$$

The PWM device on the RPi is clocked at a fixed base-clock frequency of 19.2 MHz, and therefore integer divisor and range values are used to tailor the PWM frequency for your application according to the following expression: where the range is subsequently used to adjust the duty cycle of the PWM signal. RPi PWMs share the same frequency but have independent duty cycles. The default PWM mode of operation on the RPi is to use balanced PWM. Balanced PWM means that the frequency will change as the duty cycle is adjusted, therefore to control the frequency you need to use the call `pwmSetMode (PWM_MODE_MS)` to change the mode to mark-space.

### 4.1 Hard PWM

Listing 4.1 shows a PWM example, which uses both PWMs on the RPi to generate two signals with different duty cycles. The script can be accessed by navigating to

`exploringPi/chp06/wiringPi/pwm.cpp` Implement the circuit corresponding to the script to turn on two LEDs using an Analog output. LEDs are current-controlled devices, so PWM is typically employed to provide brightness-level control. This is achieved by flashing the LED faster than can be perceived by a human, where the amount of time that the LED remains on, versus off (i.e., the duty cycle) affects the human-perceived brightness level.

**Note:** To compile this code use: `g++ pwm.cpp -o pwm -lwiringPi`.

Listing 4.1: PWM code in C++

```
#include <iostream>
#include <wiringPi.h>
using namespace std;

#define PWM0      12          // this is physical pin 12
#define PWM1      33          // only on the RPi B+/A+/2

int main() {                 // must be run as root
    wiringPiSetupPhys();     // use the physical pin numbers
    pinMode (PWM0, PWM_OUTPUT); // use the RPi PWM output
    pinMode (PWM1, PWM_OUTPUT); // only on recent RPis

    // Setting PWM frequency to be 10kHz with a full range of 128 steps
    // PWM frequency = 19.2 MHz / (divisor * range)
    // 10000 = 19200000 / (divisor * 128) => divisor = 15.0 = 15
    pwmSetMode (PWM_MODE_MS); // use a fixed frequency
    pwmSetRange (128);        // range is 0-128
    pwmSetClock (15);         // gives a precise 10kHz signal
```

```

cout << "The_PWM_Output_is_enabled" << endl;
pwmWrite(PWM0, 32);           // duty cycle of 25% (32/128)
pwmWrite(PWM1, 64);           // duty cycle of 50% (64/128)
return 0;                     // PWM output stays on after exit
}

```

### 4.1.1 PWM Application: Fading an LED

Listing 4.2 provides a code example for slowly fading an LED on and off using PWM. Implement the circuit corresponding to the script to fade in and out and LED.

Listing 4.2: PWM code in C++

```

#include <iostream>
#include <wiringPi.h>
#include <unistd.h>
using namespace std;
#define PWM_LED 18           // this is PWM0, Pin 12
bool running = true;        // fade in/out until button pressed
int main() {                // must be run as root
    wiringPiSetupGpio();     // use the GPIO numbering
    pinMode(PWM_LED, PWM_OUTPUT); // the PWM LED - PWM0
    cout << "Fading_the_LED_in/out" << endl;
    for(int i=1; i<=1023; i++) { // Fade fully on
        pwmWrite(PWM_LED, i);
        usleep(2000);
    }
    for(int i=1022; i>=0; i--) { // Fade fully off
        pwmWrite(PWM_LED, i);
        usleep(2000);
    }

    pwmWrite(PWM_LED, 1023);
    cout << "LED_Off:_Program_has_finished_gracefully!" << endl;
    return 0;
}

```

## 4.2 Soft PWM

It is possible to use software PWM on the other GPIO pins by toggling the GPIO, but this approach has a high CPU cost and is only suitable for low-frequency PWM signals. Alternatively, additional circuitry can be used to add hardware PWMs to each I<sup>2</sup>C bus. WiringPi includes a software-driven PWM handler capable of outputting a PWM signal on any of the Raspberry Pi's GPIO pins. An example of Software PWM is shown in Listing 4.3. Implement the circuit corresponding to the above script to turn on two LEDs using a Software PWM output.

Listing 4.3: PWM code in C

```

#include <wiringPi.h>
#include <softPwm.h>
#include <unistd.h>

#define GPIO1 4
#define GPIO2 17

int main(int argc, char *argv[])
{
    if (wiringPiSetupGpio() < 0) return 1;

    pinMode(GPIO1, OUTPUT);
    digitalWrite(GPIO1, LOW);
    //int softPwmCreate (int pin, int initialValue, int pwmRange) ;
}

```

```
softPwmCreate(GPIO1, 0, 200);  
// void softPwmWrite (int pin, int value) ;  
softPwmWrite(GPIO1, 15);  
  
pinMode(GPIO2, OUTPUT);  
digitalWrite(GPIO2, LOW);  
softPwmCreate(GPIO2, 0, 500);  
softPwmWrite(GPIO2, 15);  
sleep(10);  
}
```

This can also be done using the gpiozero using the following scripts.

Listing 4.4: LED code in Python

```
from gpiozero import PWMLED  
from time import sleep  
  
led = PWMLED(17)  
  
while True:  
    led.value = 0 # off  
    sleep(1)  
    led.value = 0.5 # half brightness  
    sleep(1)  
    led.value = 1 # full brightness  
    sleep(1)
```

Listing 4.5: LED code in Python

```
from gpiozero import PWMLED  
from signal import pause  
  
led = PWMLED(17)  
  
led.pulse()  
  
pause()
```



# Chapter 5

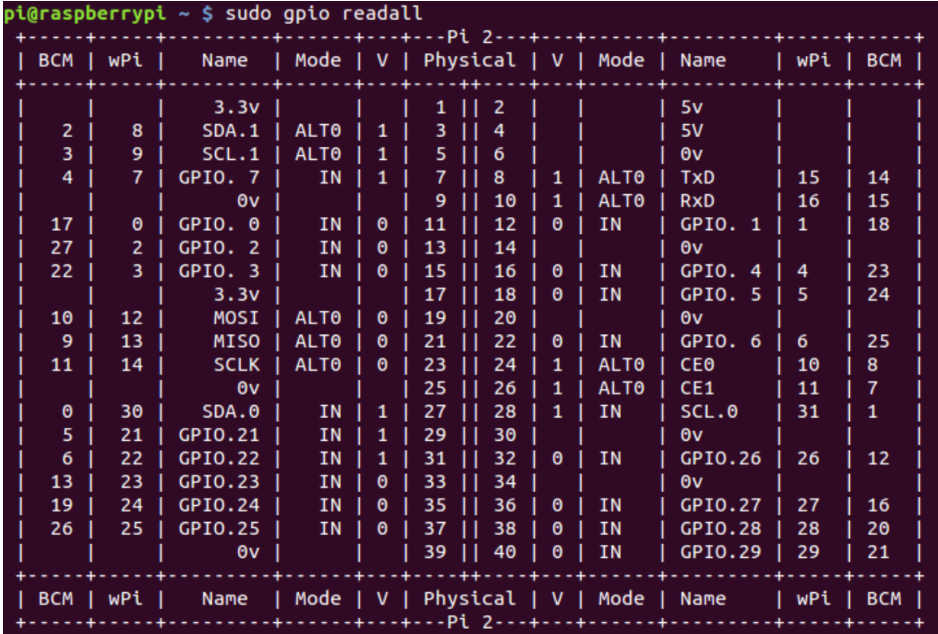
## Analog Input

In this lab we will work learn how to use the Raspberry pi for Analog input.

The default configurations of the RPi GPIO pins can be viewed by querying the following in terminal:

```
sudo gpio readall
```

This will display the default configurations as shown in Figure 5.1. Make sure to use the correct pin as initialized in the initial setup of wiringPi (<http://wiringpi.com/reference/setup/>).



```
pi@raspberrypi ~ $ sudo gpio readall
-----Pi 2-----
| BCM | wPi |   Name   | Mode | V | Physical | V | Mode | Name   | wPi | BCM |
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 2   | 8   | SDA.1    | ALT0 | 1 | 3   | 4   |      | 5V     |     |     | |
| 3   | 9   | SCL.1    | ALT0 | 1 | 5   | 6   |      | 5V     |     |     |
| 4   | 7   | GPIO. 7  | IN    | 1 | 7   | 8   | 1   | ALT0  | Tx0  | 15  | 14  |
|     |     | 0v       |      |   |     |     |     | ALT0  | Rx0  | 16  | 15  |
| 17  | 0   | GPIO. 0  | IN    | 0 | 11  | 12  | 0   | IN    | GPIO. 1 | 1  | 18  |
| 27  | 2   | GPIO. 2  | IN    | 0 | 13  | 14  |     |      | 0v     |     |     |
| 22  | 3   | GPIO. 3  | IN    | 0 | 15  | 16  | 0   | IN    | GPIO. 4 | 4  | 23  |
|     |     | 3.3v     |      |   |     |     |     |      | 0v     |     |     |
| 10  | 12  | MOSI     | ALT0 | 0 | 17  | 18  | 0   | IN    | GPIO. 5 | 5  | 24  |
| 9   | 13  | MISO     | ALT0 | 0 | 19  | 20  |     | IN    | 0v     |     |     |
| 11  | 14  | SCLK     | ALT0 | 0 | 21  | 22  | 0   | IN    | GPIO. 6 | 6  | 25  |
|     |     | 0v       |      |   |     |     |     |      | CE0    | 10  | 8   |
| 0   | 30  | SDA.0    | IN    | 1 | 23  | 24  | 1   | ALT0  | CE1    | 11  | 7   |
| 5   | 21  | GPIO.21  | IN    | 1 | 25  | 26  | 1   | ALT0  | SCL.0  | 31  | 1   |
| 6   | 22  | GPIO.22  | IN    | 1 | 27  | 28  | 0   | IN    | 0v     |     |     |
| 13  | 23  | GPIO.23  | IN    | 0 | 29  | 30  |     | IN    | GPIO.26 | 26  | 12  |
| 19  | 24  | GPIO.24  | IN    | 0 | 31  | 32  | 0   | IN    | 0v     |     |     |
| 26  | 25  | GPIO.25  | IN    | 0 | 33  | 34  | 0   | IN    | GPIO.27 | 27  | 16  |
|     |     | 0v       |      |   |     |     |     | IN    | GPIO.28 | 28  | 20  |
|     |     | 0v       |      |   |     |     |     | IN    | GPIO.29 | 29  | 21  |
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| BCM | wPi |   Name   | Mode | V | Physical | V | Mode | Name   | wPi | BCM |
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
-----Pi 2-----
```

Figure 5.1: Default GPIO configurations.

### 5.1 Temperature & Humidity Sensor

**Sensor:** In this section we will learn how to use DHT-11 to read temperature and humidity data and display it on the terminal. DHT-11 features a temperature and humidity sensor module with a calibrated digital signal output. This sensor includes a resistive humidity measurement component and an NTC (Negative Temperature Coefficient) temperature measurement component, and connects to an 8-bit microcontroller to communicate with the RPi. Complete the circuit as shown in Figure 5.2.

**Communicating to One-Wire Sensors:** DHT11 can digitally communicate with the RPi using a single GPIO. The GPIO can be set high and low with respect to time to send data bits to the sensor to initiate communication. The same

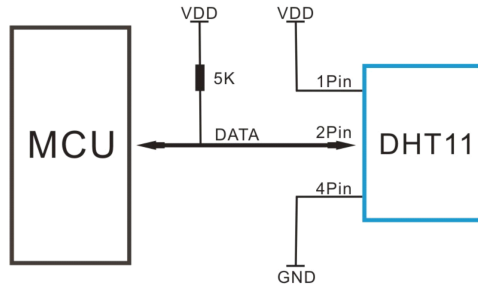


Figure 5.2: Temperature & Humidity Sensor circuit.

GPIO can then be sampled over time to read the sensor's response. Communication takes place when the RPi pulls the GPIO low for  $18ms$  and then releases the line high for a further  $20-40\mu s$ . The GPIO switches to read mode and ignores the  $80\mu s$  low level and the  $80\mu s$  high pulse that follows. The sensor then returns 5 bytes of data (i.e., 40-bits) in most-significant bit (MSB) first form, where the first 2 bytes represent the humidity value, the following 2 bytes represent the temperature, and the final byte is a parity-sum, which can be used to verify that the received data is valid (it is the 8-bit bounded sum of the preceding 4 bytes). The bits are sent by varying the duration of high pulses. When DHT is sending data to MCU, every bit of data begins with the  $50\mu s$  low-voltage-level and the length of the following high-voltage-level signal determines whether data bit is "0" or "1". A high for  $26\mu s-28\mu s$  signifies a binary 0, and a high for  $70\mu s$  signifies a binary 1. Figure 5.3 illustrates an actual oscilloscope data capture and worked calculations to explain the process for the DHT11.

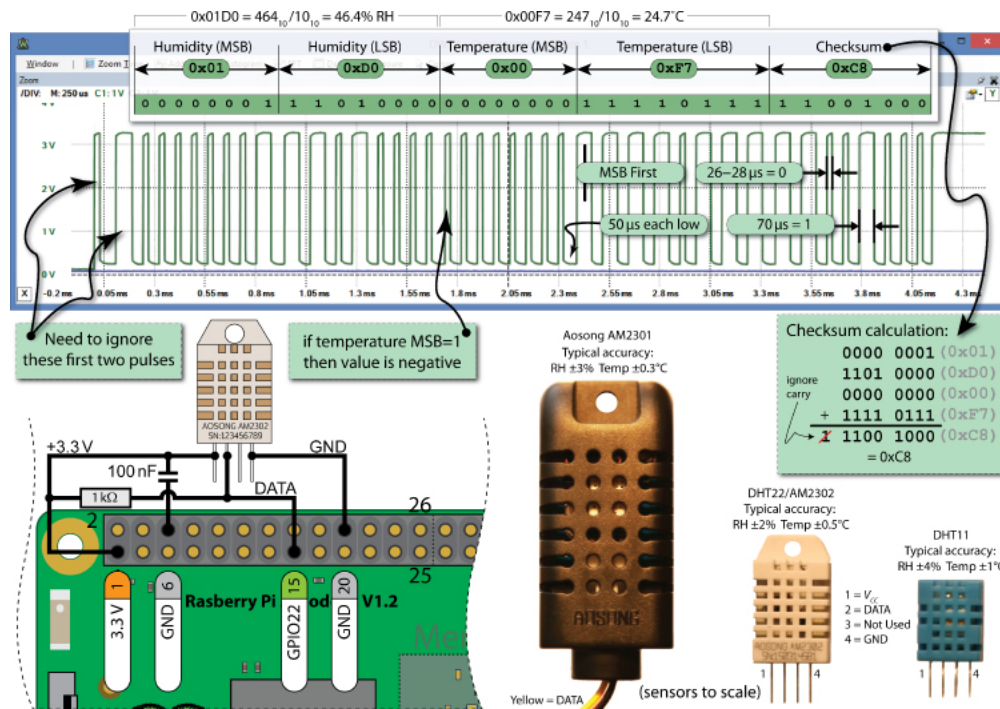


Figure 5.3: Data captured from DHT11.

The script to read the temperature and humidity is available at </exploringrpi/chp06/dht/dht.cpp>.

Listing 5.1: DHT code in C++

```
#include<iostream>
```

```

#include<unistd.h>
#include<wiringPi.h>
#include<iomanip>
using namespace std;

#define USING_DHT11      false    // The DHT11 uses only 8 bits
#define DHT_GPIO        22       // Using GPIO 22 for this example
#define LH_THRESHOLD    26       // Low=~14, High=~38 - pick avg.

int main(){
    int humid = 0, temp = 0;
    cout << "Starting_the_one-wire_sensor_program" << endl;
    wiringPiSetupGpio();
    piHiPri(99);
TRYAGAIN:                                // If checksum fails (come back here)
    unsigned char data[5] = {0,0,0,0,0};
    pinMode(DHT_GPIO, OUTPUT);           // gpio starts as output
    digitalWrite(DHT_GPIO, LOW);        // pull the line low
    usleep(18000);                       // wait for 18ms
    digitalWrite(DHT_GPIO, HIGH);       // set the line high
    pinMode(DHT_GPIO, INPUT);           // now gpio is an input

    // need to ignore the first and second high after going low
    do { delayMicroseconds(1); } while(digitalRead(DHT_GPIO)==HIGH);
    do { delayMicroseconds(1); } while(digitalRead(DHT_GPIO)==LOW);
    do { delayMicroseconds(1); } while(digitalRead(DHT_GPIO)==HIGH);
    // Remember the highs, ignore the lows -- a good philosophy!
    for(int d=0; d<5; d++) {            // for each data byte
        // read 8 bits
        for(int i=0; i<8; i++) {        // for each bit of data
            do { delayMicroseconds(1); } while(digitalRead(DHT_GPIO)==LOW);
            int width = 0;              // measure width of each high
            do {
                width++;
                delayMicroseconds(1);
                if(width>1000) break; // missed a pulse -- data invalid!
            } while(digitalRead(DHT_GPIO)==HIGH); // time it!
            // shift in the data, msb first if width > the threshold
            data[d] = data[d] | ((width > LH_THRESHOLD) << (7-i));
        }
    }
    if (USING_DHT11){
        humid = data[0] * 10;           // one byte - no fractional part
        temp = data[2] * 10;           // multiplying to keep code concise
    }
    else {                               // for DHT22 (AM2302/AM2301)
        humid = (data[0]<<8 | data[1]); // shift MSBs 8 bits left and OR LSBs
        temp = (data[2]<<8 | data[3]); // same again for temperature
    }
    unsigned char chk = 0; // the checksum will overflow automatically
    for(int i=0; i<4; i++){ chk+= data[i]; }
    if(chk==data[4]){
        cout << "The_checksum_is_good" << endl;
        cout << "The_temperature_is_" << (float)temp/10 << "C" << endl;
        cout << "The_humidity_is_" << (float)humid/10 << "%" << endl;
    }
    else {
        cout << "Checksum_bad_-_data_error_-_trying_again!" << endl;
        usleep(2000000); // have to delay for 1-2 seconds between readings
    }
}

```

```
    goto TRYAGAIN;    // a GOTO!!! call yourself a C/C++ programmer!  
  }  
  return 0;  
}
```

# Chapter 6

## More Input and Output

In this lab, we will work on another input and output device. We will use the code that came with your Sensor kit. For that, you can use the following command in your terminal to get all the code.

```
$git clone  
https://github.com/adeept/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi.git
```

### 6.1 7 Segment Display

The seven-segment display is a form of electronic display device for displaying decimal numerals that is an alternative to the more complex dot matrix displays. It is an 8-shaped LED display device composed of eight LEDs (including a decimal point). These segments are termed a, b, c, d, e, f, g, dp respectively as shown in 6.1. It can be either common anode or common cathode segment display through internal connections, as you have experienced with RGB LED. When using a common anode LED, the common anode should be connected to the power supply (VCC); when using a common cathode LED, the common cathode should be connected to the ground (GND). Each segment of a segment display is composed of LED, so a resistor is needed for protecting the LED. A 7-segment display has seven segments for displaying a figure and a segment for displaying a decimal point. If you want to display a number '1', you should only light the segment b and c.

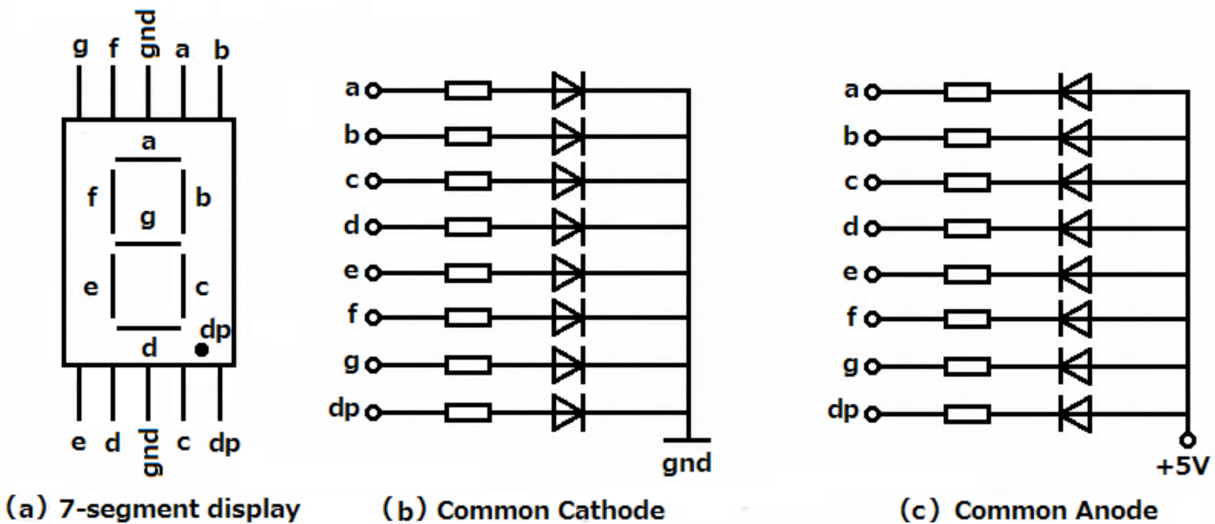


Figure 6.1: 7 segment display.

Figure 6.2 shows the circuit diagram for a common cathode 7 segment display. We will use the code at `Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/09_segment/segment.c` to determine the wiring.

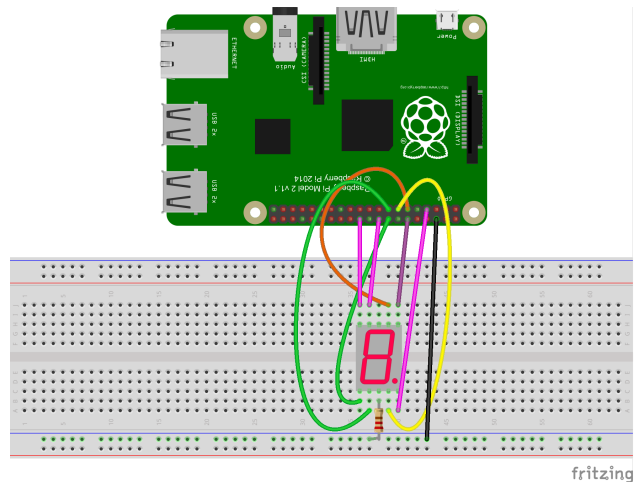


Figure 6.2: Circuit for 7 segment display with common cathode.

The code is shown in Listing 6.1.

Listing 6.1: C code for 7 Segment Display

```

/*
 * File name      : segment.c
 * Description    : display 0~9, A~F on 7-segment display
 * Website       : www.aadept.com
 * E-mail        : support@aadept.com
 * Author        : Jason
 * Date          : 2015/05/26
 */
#include <stdio.h>
#include <wiringPi.h>

typedef unsigned char uchar;

const uchar SegCode[17] = {0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c
    ,0x39,0x5e,0x79,0x71,0x80};

int main(void)
{
    int i;

    if(wiringPiSetup() < 0){ // setup wiringPi
        printf("wiringPi_setup_failed!\n");
        return -1;
    }

    for(i = 0; i < 8; i++){ // set pin mode as output(GPIO0~GPIO7)
        pinMode(i, OUTPUT);
    }

    while(1){
        for(i = 0; i < sizeof(SegCode)/sizeof(uchar); i++){ // display 0~9,A~F
            digitalWriteByte(SegCode[i]);
            delay(500);
        }
        digitalWriteByte(0x00); //segment off
        delay(1000);
    }
}

```

```

}

return 0;
}

```

You may notice that the leads are addressed using pins 0-7. Also, notice that `wiringPiSetup()` is used for setting up, which uses `wiringPi` pin numbers. Make sure that you are connecting to the correct pin number by checking the mapping using the command in the terminal: `$gpio readall`.

The output of `$gpio readall` is shown in Figure 6.3.

Pi 3B+											
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	
		3.3v			1	2		5v			
2	8	SDA.1	OUT	0	3	4		5v			
3	9	SCL.1	OUT	0	5	6		0v			
4	7	GPIO.7	OUT	0	7	8	1	TxD	15	14	
		0v			9	10	1	RxD	16	15	
17	0	GPIO.0	OUT	1	11	12	1	GPIO.1	1	18	
27	2	GPIO.2	OUT	1	13	14		0v			
22	3	GPIO.3	OUT	1	15	16	0	GPIO.4	4	23	
		3.3v			17	18	0	GPIO.5	5	24	
10	12	MOSI	IN	1	19	20		0v			
9	13	MISO	IN	1	21	22	1	GPIO.6	6	25	
11	14	SCLK	IN	1	23	24	0	CE0	10	8	
		0v			25	26	0	CE1	11	7	
0	30	SDA.0	IN	1	27	28	1	SCL.0	31	1	
5	21	GPIO.21	IN	1	29	30		0v			
6	22	GPIO.22	IN	1	31	32	0	GPIO.26	26	12	
13	23	GPIO.23	IN	0	33	34		0v			
19	24	GPIO.24	IN	0	35	36	0	GPIO.27	27	16	
26	25	GPIO.25	IN	0	37	38	0	GPIO.28	28	20	
		0v			39	40	0	GPIO.29	29	21	
BCM	wPi	Name	Mode	V	Physical	V	Mode	Name	wPi	BCM	
Pi 3B+											

Figure 6.3: GPIO Pin Map.

## 6.2 4x4 Matrix Keyboard/Keypad

A 4x4 keypad module and its pin connections are shown in figure 6.4. Internally, it has 16 buttons arranged in matrix formation, as shown in Figure 6.5.

Listing 6.2 shows the code available at

`Adept_Ultimate_Starter_Kit_C_Code_for_RPi/13_matrixKeyboard/matrixKeyboard.c`. Pay extra attention to the row and column pin numbers to make the connection.

Listing 6.2: C code for Matrix Keyboard

```

#include <wiringPi.h>
#include <stdio.h>

const int ROW[] = {0, 1, 2, 3};
const int COLUMN[] = {4, 5, 6, 7};

int getKey(void)
{
    int i;
    int tmpRead;
    int rowVal = -1;
    int colVal = -1;
    char keyVal;

    for(i = 0; i < 4; i++){

```

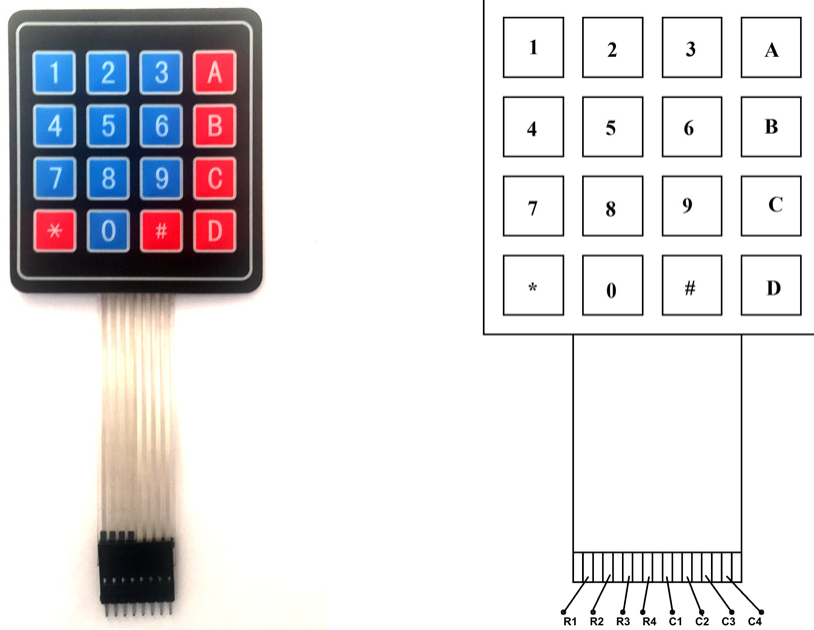


Figure 6.4: 4x4 Matrix Keyboard.

```

        pinMode(COLUMN[i], OUTPUT);
        digitalWrite(COLUMN[i], LOW);
    }

    for(i = 0; i < 4; i++){
        pinMode(ROW[i], INPUT);
        pullUpDnControl(ROW[i], PUD_UP);
    }

    for(i = 0; i < 4; i++){
        tmpRead = digitalRead(ROW[i]);
        if(tmpRead == 0){
            rowVal = i;
        }
    }

    if(rowVal < 0 || rowVal > 3){
        return -1;
    }

    for(i = 0; i < 4; i++){
        pinMode(COLUMN[i], INPUT);
        pullUpDnControl(COLUMN[i], PUD_UP);
    }

    pinMode(ROW[rowVal], OUTPUT);
    digitalWrite(ROW[rowVal], LOW);

    for(i = 0; i < 4; i++){
        tmpRead = digitalRead(COLUMN[i]);
        if(tmpRead == 0){
            colVal = i;

```



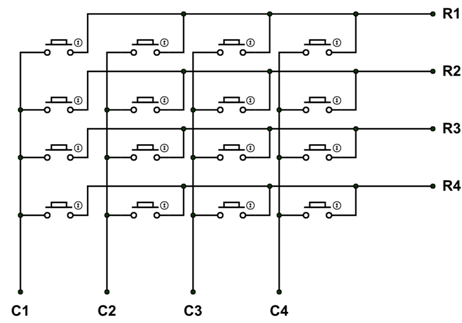


Figure 6.5: 4x4 Matrix Keyboard internal connections.

```

    }
}

if(colVal < 0 || colVal > 3){
    return -1;
}

//printf("%d, %d\n", rowVal, colVal);
switch(rowVal){
    case 0:
        switch(colVal){
            case 0: keyVal = 0; break;
            case 1: keyVal = 1; break;
            case 2: keyVal = 2; break;
            case 3: keyVal = 3; break;
            default:
                break;
        }
        break;
    case 1:
        switch(colVal){
            case 0: keyVal = 4; break;
            case 1: keyVal = 5; break;
            case 2: keyVal = 6; break;
            case 3: keyVal = 7; break;
            default:
                break;
        }
        break;
    case 2:
        switch(colVal){
            case 0: keyVal = 8; break;
            case 1: keyVal = 9; break;
            case 2: keyVal = 10; break;
            case 3: keyVal = 11; break;
            default:
                break;
        }
        break;
    case 3:
        switch(colVal){
            case 0: keyVal = 12; break;

```

```

        case 1: keyVal = 13; break;
        case 2: keyVal = 14; break;
        case 3: keyVal = 15; break;
        default:
            break;
    }
    break;
default:
    break;
}

return keyVal;
}

int main(void)
{
    int i;
    int key = -1;

    if(wiringPiSetup() == -1){
        printf("setup_wiringPi_failed!\n");
        return -1;
    }

    while(1){
        key = getKey();
        if(key != -1){
            switch(key) {
                case 0: printf("1\n"); break;
                case 1: printf("2\n"); break;
                case 2: printf("3\n"); break;
                case 3: printf("A\n"); break;
                case 4: printf("4\n"); break;
                case 5: printf("5\n"); break;
                case 6: printf("6\n"); break;
                case 7: printf("B\n"); break;
                case 8: printf("7\n"); break;
                case 9: printf("8\n"); break;
                case 10: printf("9\n"); break;
                case 11: printf("C\n"); break;
                case 12: printf("*\n"); break;
                case 13: printf("0\n"); break;
                case 14: printf("#\n"); break;
                case 15: printf("D\n"); break;
                default:
                    break;
            }
        }
        delay(200);
    }

    return 0;
}

```

# Chapter 7

## I2C Communication

In this chapter, we will study I2C. I2C on the RPi is implemented using the Broadcom Serial Controller (BSC), which supports 7-bit/10-bit addressing and bus frequencies of up to 400 kHz.

We will use ADXL345 or MPU-6050 Accelerometer in this example. The level of acceleration supported by a sensor's output signal specifications is specified in  $\pm g$ . This is the greatest amount of acceleration the part can measure and accurately represent as an output. For example, the output of a  $\pm 2g$  accelerometer is linear with acceleration up to  $\pm 2g$ . If it is accelerated at  $4g$ , the output may saturate.

1. We begin by enabling the I2C interface on the RPi by typing the following in terminal:

```
$sudo raspi-config  
Choose Interfacing Options → I2C → Yes.
```

2. Check whether I2C is enabled. `/dev` is the location of device files. If I2C was enabled correctly, it will show up in `/dev/`. Type the following in terminal.

```
$sudo ls /dev/i2c*
```

Your output will be similar to:

```
/dev/i2c-1
```

Then check if the kernel module is loaded by issuing `lsmod` command, which outputs information for each loaded kernel module on a new line:

```
$lsmod | grep i2c
```

Your output will be similar to:

```
i2c_bcm2835 16384 0
```

```
i2c_dev 16384 0
```

If those two modules are not loaded, we can use `modprobe` to load the modules in Linux Kernel.

```
$modprobe i2c_dev
```

```
$modprobe i2c_bcm2835
```

3. In the next step, we need to create the circuit as below:

- (a) Connect SCL on the RPi to SCL on the ADXL345
- (b) Connect SDA on the RPi to SDA in the ADXL345
- (c) Connect GND on the RPi to GND on the ADXL345
- (d) Connect 3.3V on the RPi to VIN on the ADXL345

4. The I<sup>2</sup>C bus can be probed to detect connected devices by using the following command:

```
sudo i2cdetect -y -r 1
```

The output shows the I2C addresses of the attached devices. For example, when ADXL345 (0x53), MPU-6050 (0x68) and PCF8591T (0x48) are attached to the I2C bus, we get the following output.

5. Refer to the manual (ADXL345 or MPU6050) for understanding the functionality and registers.

```

      0  1  2  3  4  5  6  7  8  9  a  b  c  d  e  f
00:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
10:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
20:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
30:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --
40:  --  --  --  --  --  --  --  --  48  --  --  --  --  --  --  --
50:  --  --  --  53  --  --  --  --  --  --  --  --  --  --  --  --
60:  --  --  --  --  --  --  --  --  68  --  --  --  --  --  --  --
70:  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --  --

```

Figure 7.1: I2C Detect Output.

6. We will use the code that came with your Sensor kit. For ADXL345, the code resides in folder `Adept_Ultimate_Starter_Kit_C_Code_for_RPi/25_ADXL345/`.  
For MPU6050, clone the repository as below:  

```
$git clone https://github.com/adeept/Adept_Ultimate_Starter_Kit_for_RPi.git
```

The code for MPU6050 is available at  
`Adept_Ultimate_Starter_Kit_for_RPi/Lesson05_MPU6050/code/c/Lesson05_MPU6050.c`
7. Both the codes use `wiringPi`, for which you need to compile with `lwiringPi`. MPU6050 code also uses `math` library, for which you need to compile it with `-lm`.
8. `WiringPi` implementation for I2C is available at <https://github.com/WiringPi/WiringPi/blob/master/wiringPi/wiringPiI2C.c>
9. Note that calibration should be performed for correct values, which can be set in the offset registers.  
<https://learn.adafruit.com/adxl345-digital-accelerometer?view=all#programming>

# Chapter 8

## Servo Motor

In this chapter, we will discuss Servo Motors and learn how to control it with the Raspberry Pi.

The Servo motor is a type of geared motor that can rotate 180 degrees. It is controlled by sending pulses signal from your microcontroller. These pulses tell the servo what position it should move to. The Servo motor consists of a shell, circuit board, non-core motor, gear and location detection modules. Its working principle is as follow:

1. The Raspberry Pi sends a PWM signal to the servo motor.
2. This signal is processed by an IC on circuit board to calculate the rotation direction to drive the motor, and then this driving power is transferred to the swing arm by a reduction gear.
3. The position detector returns the location signal to gauge whether the set location is reached or not.

The relationship between the rotation angle of the servo and pulse width is shown in Figure 8.1.

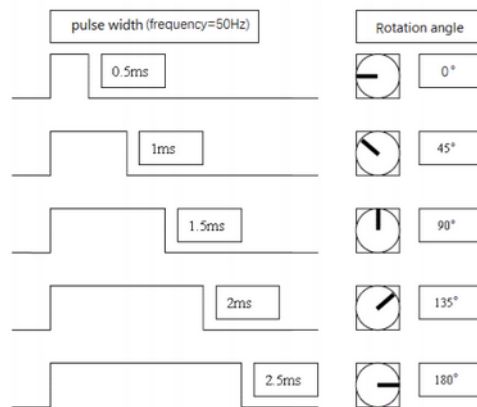
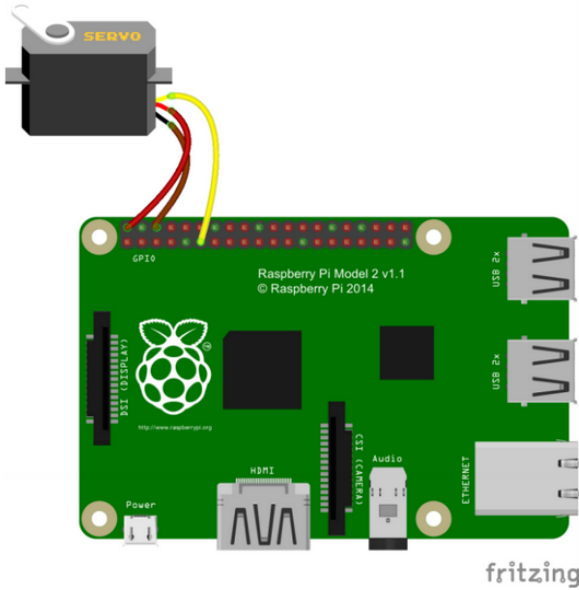


Figure 8.1: Servo Motor Principle.

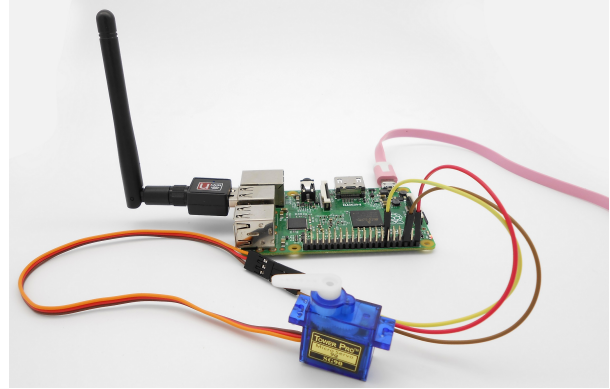
There are three pins in a Servo motor, where you should use 5V for power supply. The color coding of wiring of the servo can be of three different types as shown in Table 8.1.

Pin Number	Signal Name	(Futaba)	(JR)	(Hitec)
1	Ground	Black	Brown	Black
2	Power Supply	Red	Red	Red or Brown
3	Control Signal	White	Orange	Yellow or White

Table 8.1: Color coding for the wiring of Servo



(a) Circuit Connection.



(b) Servo Connected with Raspberry Pi.

Figure 8.2: Servo Connection with Raspberry Pi.

In the next step, you should connect the Servo motor with the Raspberry Pi as shown in Figure 8.2, using the color code in Table 8.1.

The script, as shown in Listing 8.1, is available at:

`/home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/23_servo/servo.c.`

It generates software PWM signals for moving the Servo clockwise and anticlockwise.

Listing 8.1: Servo Motor Code

```
#include <stdio.h>
#include <wiringPi.h>

#define Servo 0

void servo(int angle) //500~2500
{
    digitalWrite(Servo, 1);
    delayMicroseconds (angle);
    digitalWrite(Servo, 0);
    delayMicroseconds (20000-angle);
}

int main(void)
{
    int i, j;

    if(wiringPiSetup() < 0){
        printf("wiringPi_setup_error!\n");
        return -1;
    }

    pinMode(Servo, OUTPUT);

    while(1){
```

```
servo(500);
delay(500);
for(i=500; i <=2500; i=i+500){
    servo(i);
    printf("i=%d\n", i);
    delay(500);
}
servo(2500);
delay(500);
for(i=2500; i >=500; i=i-500){
    servo(i);
    printf(".....i=%d\n", i);
    delay(500);
}
}

return 0;
}
```

# Chapter 9

## Stepper Motor

In this lab, we will learn the principles of Stepper Motor and how to control it with Raspberry Pi.

**Stepper Motor:** A stepper motor is a motor that converts electrical pulse signals into corresponding angular or linear displacements. Unlike DC motors, which rotate continuously when a DC voltage is applied, stepper motors normally rotate in discrete fixed-angle steps. Each time a pulse signal is input, the rotor rotates by an angle or a step forward. The output angular displacement or linear displacement is proportional to the number of pulses input, and the rotation speed is proportional to the pulse frequency. Therefore, stepper motors are also called pulse motors. Stepper motors can be positioned very accurately, because they typically have a positioning error of less than 5% of a step (i.e., typically  $0.1^\circ$ ). The error does not accumulate over multiple steps, so stepper motors can be controlled in an *open-loop form*, without the need for feedback. The motor from your Sensor kit is shown in Figure 9.1.

**ULN2003 Driver Module:** The Raspberry Pi's GPIO cannot directly drive a stepper motor due to the weak current. Therefore, a driver circuit is necessary to control the stepper motor. This experiment uses the ULN2003 driver module. There are four LEDs on the module to indicate stepping state. The white socket in the middle is for connecting a stepper motor. IN1, IN2, IN3, IN4 are used to connect with the microcontroller. The power supply [5V to 12V DC] and an On/Off jumper also resides on the board as shown in Figure 9.1.

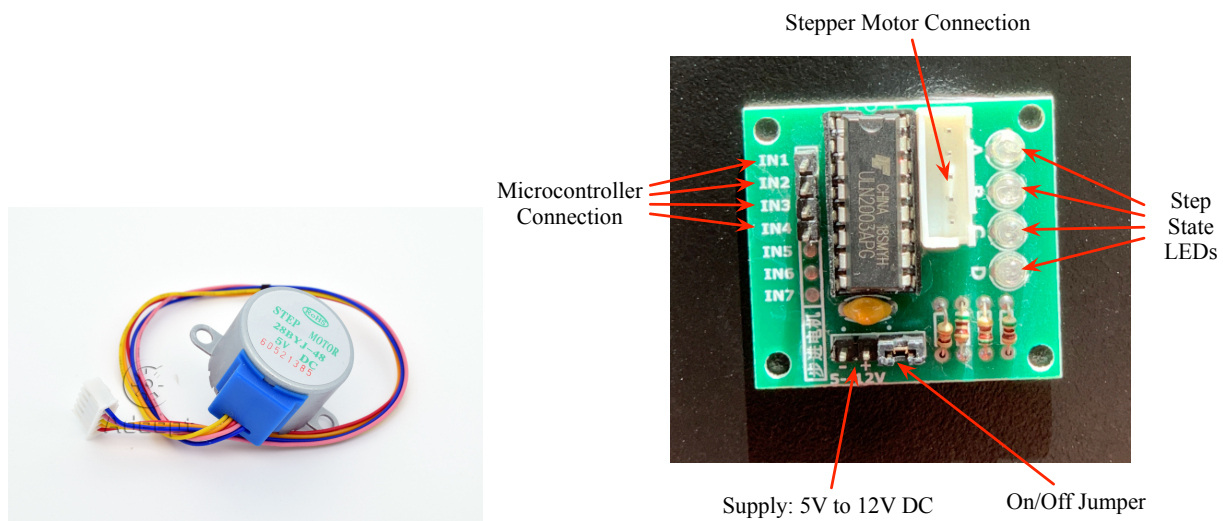


Figure 9.1: Stepper Motor and ULN 2003A.



**Circuit:** Figure 9.2 shows the circuit connection for the stepper motor with the driver and the Microcontroller. The script to control the Stepper Motor can be accessed at:

`/home/Adeept_Ultimate_Starter_Kit_C_Code_for_RPi/24_stepperMotor.c`. It is also shown in Listing 9.1. Based on the code that you will be using, connect 4 GPIO pins from your Raspberry Pi to the four inputs of the ULN2003A. When connecting the circuit, we should pay attention to the difference between positive and negative poles.

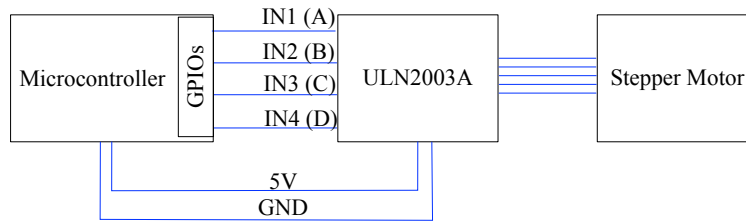


Figure 9.2: Stepper Motor Circuit.

### Understanding the Code:

1. The GPIO pins should be output.
2. The four pins should be set at high level for the four pins in sequence to control the clockwise rotation of the stepping motor.
3. The four pins should be set at high level for the four pins in sequence *in reverse order* to control the *anti-clockwise* rotation of the stepping motor.
4. When all output are 0, the driver stops.
5. 512 steps are needed to turn 360°.

Listing 9.1: Stepper Motor Code

```
/*
 * File name   : stepperMotor.c
 * Description : control a stepper motor.
 * Website    : www.adeept.com
 * E-mail     : support@adeept.com
 * Author     : Jason
 * Date      : 2015/06/21
 */
#include <wiringPi.h>
#include <stdio.h>

#define IN1 0 // wiringPi GPIO0 (pin11)
#define IN2 1 // pin12
#define IN3 2 // pin13
#define IN4 3 // pin15

void setStep(int a, int b, int c, int d)
{
    digitalWrite(IN1, a);
    digitalWrite(IN2, b);
    digitalWrite(IN3, c);
    digitalWrite(IN4, d);
}

void stop(void)
{
    setStep(0, 0, 0, 0);
}
```

```

void forward(int t, int steps)
{
    int i;

    for(i = 0; i < steps; i++){
        setStep(1, 0, 0, 0);
        delay(t);
        setStep(0, 1, 0, 0);
        delay(t);
        setStep(0, 0, 1, 0);
        delay(t);
        setStep(0, 0, 0, 1);
        delay(t);
    }
}

void backward(int t, int steps)
{
    int i;

    for(i = 0; i < steps; i++){
        setStep(0, 0, 0, 1);
        delay(t);
        setStep(0, 0, 1, 0);
        delay(t);
        setStep(0, 1, 0, 0);
        delay(t);
        setStep(1, 0, 0, 0);
        delay(t);
    }
}

int main(void)
{
    if (wiringPiSetup() < 0) {
        printf("Setup_wiringPi_failed!\n");
        return -1;
    }

    /* set pins mode as output */
    pinMode(IN1, OUTPUT);
    pinMode(IN2, OUTPUT);
    pinMode(IN3, OUTPUT);
    pinMode(IN4, OUTPUT);

    while (1){
        printf("forward...\n");
        forward(3, 512);

        printf("stop...\n");
        stop();
        delay(2000);          // 2s

        printf("backward...\n");
        backward(3, 256);    // 512 steps ---- 360 angle

        printf("stop...\n");
        stop();
    }
}

```

```
        delay(2000);      // 2s
    }
    return 0;
}
```