VMP502

BASIC LEARNING KIT FOR RASPBERRY PI®

USER MANUAL

USER MANUAL

1. Introduction

To all residents of the European Union

Important environmental information about this product

This symbol on the device or the package indicates that disposal of the device after its lifecycle could harm the environment. Do not dispose of the unit (or batteries) as unsorted municipal waste; it should be taken to a specialized company for recycling. This device should be returned to your distributor or to a local recycling service. Respect the local environmental rules.

If in doubt, contact your local waste disposal authorities.

Thank you for choosing Velleman®! Please read the manual thoroughly before bringing this device into service. If the device was damaged in transit, do not install or use it and contact your dealer.

2. Safety Instructions

- This device can be used by children aged from 8 years and above, and persons with reduced physical, sensory or mental capabilities or lack of experience and knowledge if they have been given supervision or instruction concerning the use of the device in a safe way and understand the hazards involved. Children shall not play with the device. Cleaning and user maintenance shall not be made by children without supervision.

- Indoor use only.
  Keep away from rain, moisture, splashing and dripping liquids.

3. General Guidelines

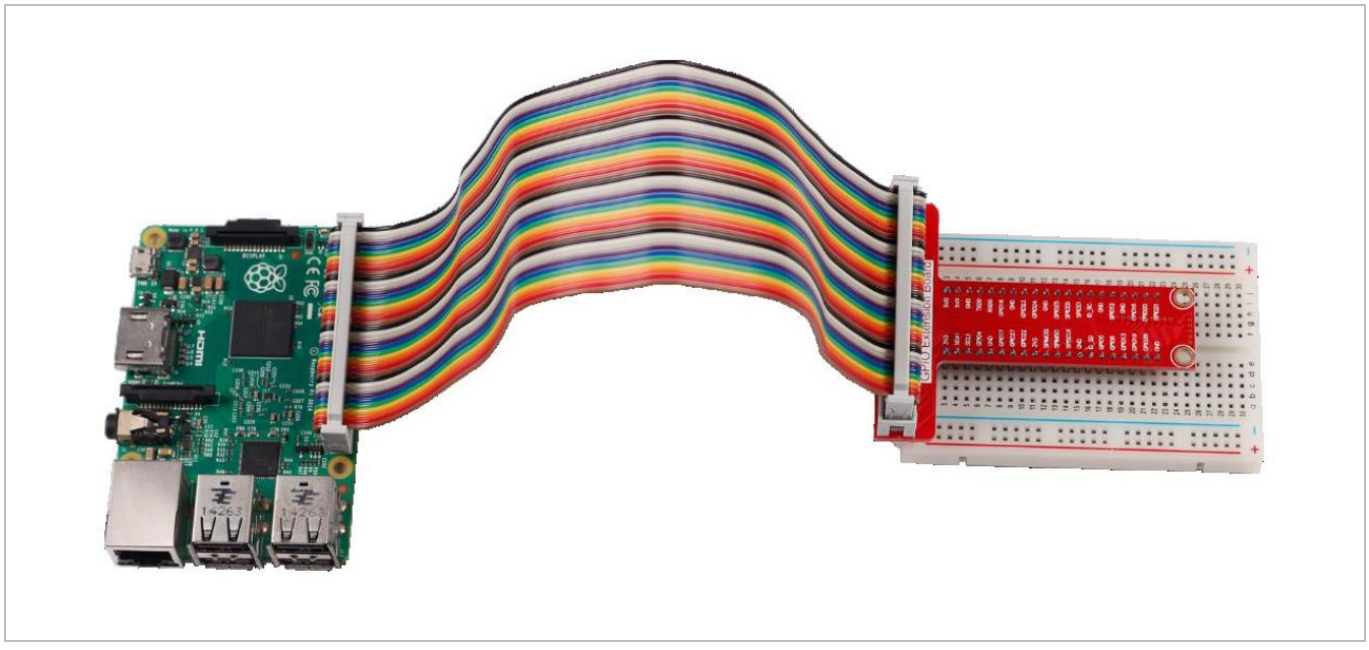
- Refer to the Velleman® Service and Quality Warranty on the last pages of this manual.
- Familiarise yourself with the functions of the device before actually using it.
- All modifications of the device are forbidden for safety reasons. Damage caused by user modifications to the device is not covered by the warranty.
- Only use the device for its intended purpose. Using the device in an unauthorised way will void the warranty.
- Damage caused by disregard of certain guidelines in this manual is not covered by the warranty and the dealer will not accept responsibility for any ensuing defects or problems.
- Nor Velleman nv nor its dealers can be held responsible for any damage (extraordinary, incidental or indirect) – of any nature (financial, physical…) arising from the possession, use or failure of this product.
- Due to constant product improvements, the actual product appearance might differ from the shown images.
- Product images are for illustrative purposes only.
- Do not switch the device on immediately after it has been exposed to changes in temperature. Protect the device against damage by leaving it switched off until it has reached room temperature.
- Keep this manual for future reference.

4. Contents

- 830 points solderless breadbord
- 5 x 10K resistors (RA10K0)
- 5 x 2K resistors
- 5 x 220R resistors (RA220E0)
- 1 x 1602 LCD module (LCD1602BLC)
- 1 x T-shape GPIO expansion board
- 1 x ribbon cable for GPIO board
- 1 x infrared receiver VS1838
- 1 x mini remote control for VMA317
- 1 x 50K potentiometer (K047AM)
- 4 x button with round cap 4-pin 12 x 12 mm
- 1 x 40 pin 2.54 mm single row male pin header
- 3 x photo sensitive transistor (SGPT5053C)
- 1 x LM35 temperature sensor (LM35DZ)
- 1 x active buzzer 5 V
- 1 x passive buzzer 5 V
- 30 x breadboard jumper wire M-M different length
- 1 x 20P / 20 cm male to female jumper
- 1 x micro servo 9 g (VMA600)
- 1 x MAX7219 + 1088AS matrix 8 x 8 LED module
- 1 x PCF8591 analog to digital converter module + 3p jumper
- 1 x 5 mm RGB LED
- 1 x flame sensor YG1006 (IR photodiode)
- 2 x tilting sensor (MERS4)
- 1 x shift register 74HC595N
- 1 x 1 digit 7 segment display SMA42056
- 1 x 4 digit 7 segment display SMA420564
- 1 x plastic clear box 200 x 140 x 48 mm

5. How to Use the GPIO Extension Board

Connect the Raspberry Pi® and the extension board as follows:



6. Operation

6.1 A Blinking LED

Learn how to programme the Raspberry Pi® to make an LED burn.

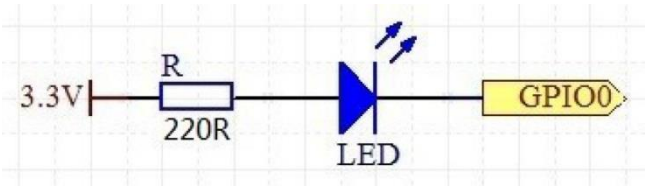
Required Hardware

- 1 x Raspberry Pi®
- 1 x breadboard
- 1 x LED
- 1 x 220 Ω resistor
- jumper wires as needed

A semiconductor LED is a type of component, which can turn electric energy into light energy via PN junctions. By wavelength, it can be categorized into a laser diode, an IR LED and a visible LED.

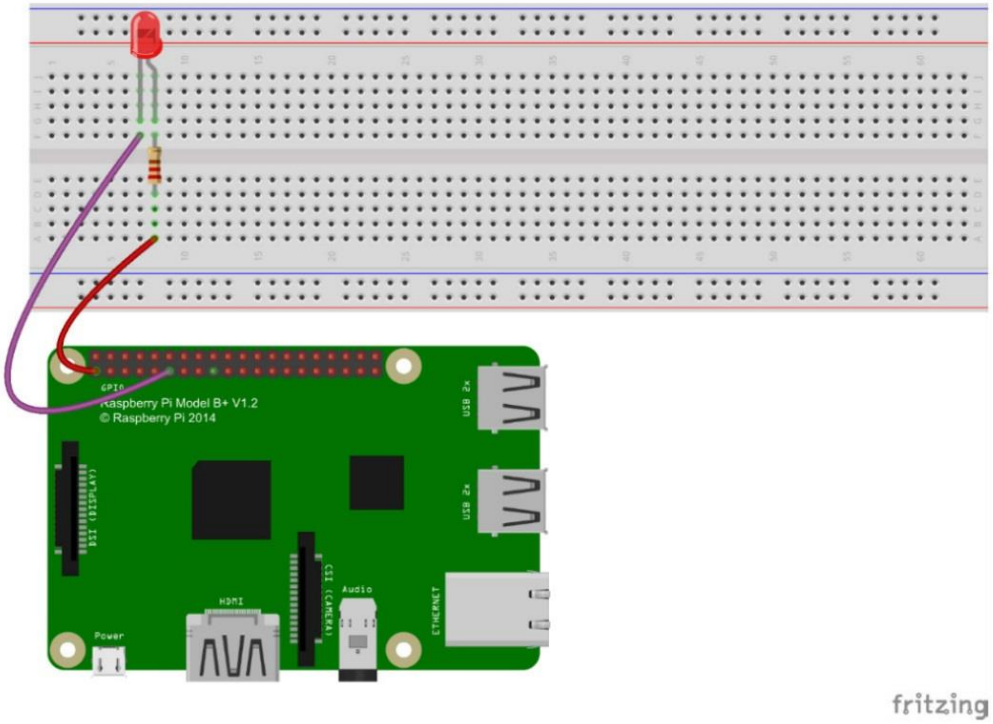
When a 2 V to 3 V forward voltage is supplied to an LED, it will blink only if the forward currents flow through the LED. Usually, there are red, yellow, green, blue and colour-changing LEDs. LEDs are widely used due to their low operating voltage, low current, luminescent stability and small size.

LEDs are diodes. Hence, they have a voltage drop, which varies from 1 V to 3 V depending on their types. Likewise, LEDs usually emit light if supplied with a 5 mA tot 30 mA current, and generally 10 ma to 20 mA is used. When an LED is used, it is necessary to connect a current-limiting resistor to protect the LED from over-burning.



In this experiment, connect a 220 Ω resistor to the anode of the LED, connect the resistor to a 3.3 V power source, connect the cathode of the LED to the GPIO. Write 0 to the GPIO and the LED will blink.

Experiment



C Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_C_code_for_RaspberryPi/01_LED
2. Compile:
   gcc led.c –o led –lwiringPi
3. Run:
   sudo ./led

Python Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_Python_code_for_RaspberryPi/
2. Run:
   sudo python 01_led.py

The LED should be blinking. Change the delay time if you want the LED to blink faster.

Programming

C Programming

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

#define LedPin 0
int main(void)
{
if(wiringPiSetup()  ==  -1){  //when  initialize  wiring  failed,print
messageto screen
printf("setup wiringPi failed !");
return 1;
}
printf("linker  LedPin  :  GPIO  %d(wiringPi  pin)\n",LedPin);  //when
initialize wiring successfully,print message to screen

pinMode(LedPin, OUTPUT);
while(1){
digitalWrite(LedPin, LOW); //led on
printf("led on...\n");
delay(500);
digitalWrite(LedPin, HIGH); //led off
printf("...led off\n");
delay(500);
}
return 0;
}
```

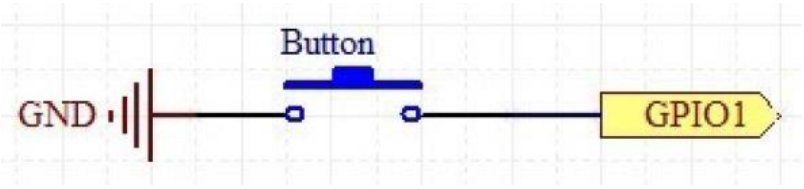
Python Programming

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time
LedPin = 11 # pin11
def setup():
GPIO.setmode(GPIO.BOARD) # Numbers GPIOs by physical location
GPIO.setup(LedPin, GPIO.OUT) # Set LedPin's mode is output
GPIO.output(LedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to off led
def loop():
while True:
print '...led on'
GPIO.output(LedPin, GPIO.LOW) # led on
time.sleep(0.5)
print 'led off...'
GPIO.output(LedPin, GPIO.HIGH) # led off

time.sleep(0.5)
def destroy():
GPIO.output(LedPin, GPIO.HIGH) # led off
GPIO.cleanup() # Release resource
if __name__ == '__main__': # Program start from here
setup()
try:
loop()
except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the child
program destroy() will be executed.
destroy()
```

6.2 Controlling an LED by a Button

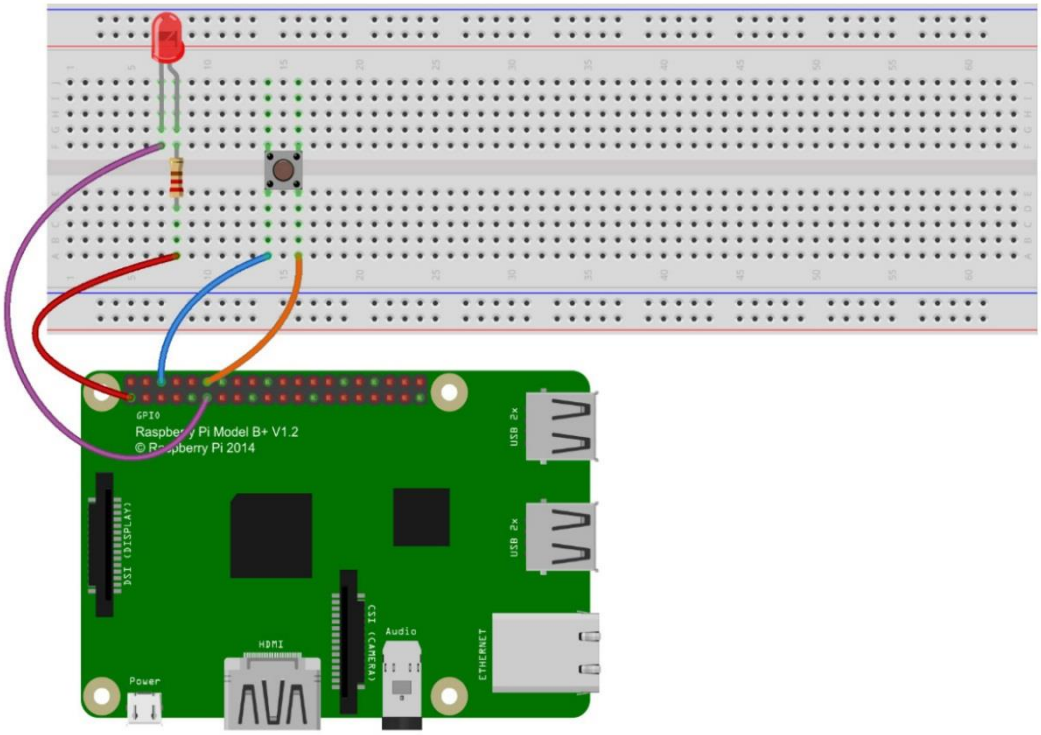
Required Hardware

- 1 x Raspberry Pi®
- 1 x breadboard
- 1 x LED
- 1 x 220 Ω resistor
- 1 x button
- jumper wires as needed



Use a NO button as the Raspberry Pi® input. When the button is pressed, the GPIO connected to the button will turn into low level (0 V). We can detect the state of the GPIO connected to the button through programming. You can run the corresponding code when the button is pressed, and the LED will light.

Experiment



C Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_C_code_for_RaspberryPi/02_BtnAndLED/
2. Compile:
   gcc BtnAndLed.c –o BtnAndLed –lwiringPi
3. Run:
   sudo ./BtnAndLed

Python Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_Python_code_for_RaspberryPi/
2. Run:
   sudo python 01_btnAndLed.py

Hold the button pressed and the LED will light; release the button to switch off the LED.

Programming

C Programming

```
#include <wiringPi.h>
#include <stdio.h>
#define LedPin 0
#define ButtonPin 1
int main(void)
{
if(wiringPiSetup()  ==  -1){  //when  initialize  wiring  failed,print
messageto screen
printf("setup wiringPi failed !");
return 1;
}

pinMode(LedPin, OUTPUT);

pinMode(ButtonPin, INPUT);
pullUpDnControl(ButtonPin, PUD_UP); //pull up to 3.3V,make GPIO1 a stable
level
while(1){
digitalWrite(LedPin, HIGH);
if(digitalRead(ButtonPin) == 0){ //indicate that button has pressed down
digitalWrite(LedPin, LOW); //led on
}
}
return 0;
}
```

Python Programming

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
LedPin = 11 # pin11 --- led
BtnPin = 12 # pin12 --- button
Led_status = 1
def setup():
GPIO.setmode(GPIO.BOARD) # Numbers GPIOs by physical location
GPIO.setup(LedPin, GPIO.OUT) # Set LedPin's mode is output
GPIO.setup(BtnPin, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Set BtnPin's mode
is input, and pull up to high level(3.3V)
GPIO.output(LedPin, GPIO.HIGH) # Set LedPin high(+3.3V) to off led
def swLed(ev=None):
global Led_status
```

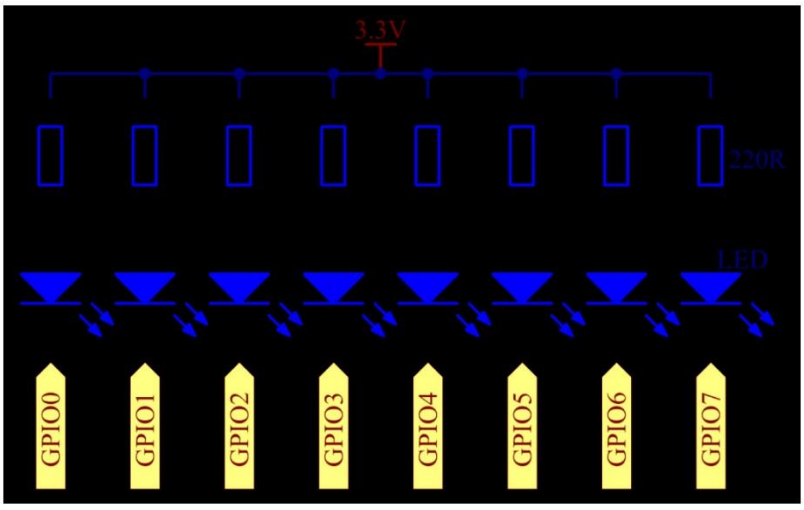
```
Led_status = not Led_status
GPIO.output(LedPin, Led_status) # switch led status(on-->off; off-->on)
if Led_status == 1:
print 'led off...'
else:
print '...led on'
def loop():
GPIO.add_event_detect(BtnPin, GPIO.FALLING, callback=swLed) # wait for
falling
while True:
pass # Don't do anything
def destroy():
GPIO.output(LedPin, GPIO.HIGH) # led off

GPIO.cleanup() # Release resource
if __name__ == '__main__': # Program start from here
setup()
try:
loop()
except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the child program
destroy() will be executed.
destroy()
```

6.3 Flowing LEDs

We will see how to make eight LEDs blink in various effects.

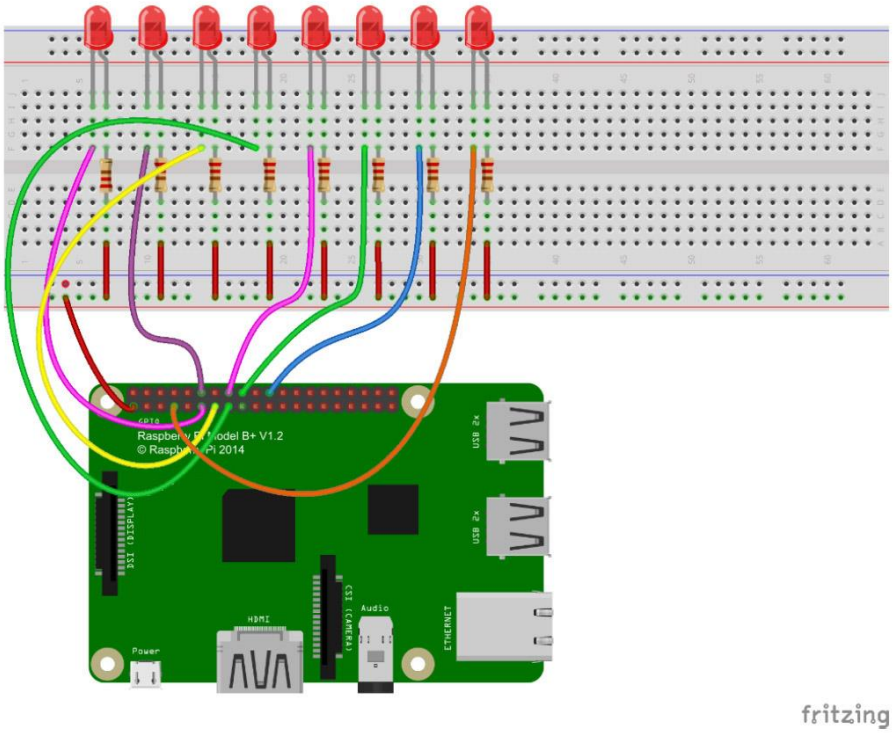
Required Hardware

- 1 x Raspberry Pi®
- 1 x breadboard
- 8 x LED
- 8 x 220 Ω resistor
- jumper wires as needed



Set the GPIO to a low level in turn by programming and LEDs 0 to 7 will light in turn. Control the delay and order to make the LEDs blink in different effects.

Experiment



C Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_C_code_for_RaspberryPi/03_8Led/
2. Compile:
   gcc 8Led.c –o 8Led –lwiringPi
3. Run:
   sudo ./8Led

Python Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_Python_code_for_RaspberryPi/
2. Run:
   sudo python 03_8Led.py

You will see eight LEDs light circularly and render different effects.

You can write the blinking effects of the LEDs in an array. If you want to use one of these effects, you can call it in the *main()* function directly.

Programming

C Programming

```
#include <wiringPi.h>
#include <stdio.h>
//make led_n on
void led_on(int n)
{
digitalWrite(n, LOW);
}
//make led_n off
void led_off(int n)
{
digitalWrite(n, HIGH);
}
int main(void)
{
int i;
if(wiringPiSetup()  ==  -1){  //when  initialize  wiring  failed,print
messageto screen
printf("setup wiringPi failed !");
return 1;
}
for(i=0;i<8;i++){
printf("linker LedPin : GPIO %d(wiringPi pin)\n",i); //when initialize
wiring successfully,print message to screen
}
for(i=0;i<8;i++){ //make 8 pins' mode is output
pinMode(i, OUTPUT);
}
while(1){
for(i=0;i<8;i++){ //make led on from left to right
led_on(i);
delay(100);
led_off(i);
}
// delay(500);
for(i=8;i>=0;i--){ //make led off from right to left
led_on(i);
delay(100);
led_off(i);
}
}
return 0;
}
```

Python Programming

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time
pins = [11, 12, 13, 15, 16, 18, 22, 7]
def setup():

GPIO.setmode(GPIO.BOARD) # Numbers GPIOs by physical location
for pin in pins:
GPIO.setup(pin, GPIO.OUT) # Set all pins' mode is output
GPIO.output(pin, GPIO.HIGH) # Set all pins to high(+3.3V) to off led
def loop():
while True:
for pin in pins:
GPIO.output(pin, GPIO.LOW)
time.sleep(0.5)
GPIO.output(pin, GPIO.HIGH)
def destroy():
for pin in pins:
GPIO.output(pin, GPIO.HIGH) # turn off all leds
GPIO.cleanup() # Release resource
if __name__ == '__main__': # Program start from here
setup()
try:
loop()
except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the child program
destroy() will be executed.
destroy()
```

6.4 Breathing LED

We will gradually increase and decrease the luminance of an LED with PWM.

Required Hardware

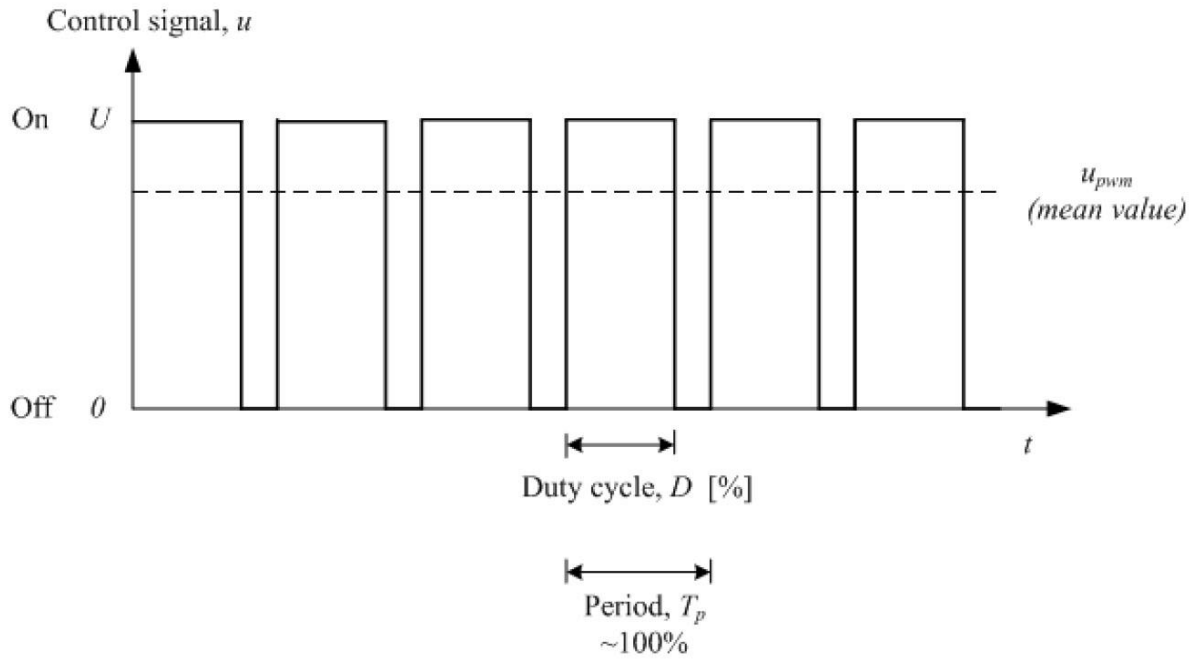
- 1 x Raspberry Pi®
- 1 x breadboard
- 1 x LED
- 1 x 220 Ω resistor
- jumper wires as needed

Pulse Width Modulation – or PWM – is a technique for getting analogue results with digital means. Digital control is used to create a square wave, a signal switched between on and off. This on-off pattern can simulate voltages between full on (3.3 V) and off (0 V) by changing the portion of the time the signal spends on versus the time that the signal spends off. The duration of the "on time" is called the pulse width. To get varying analogue values, you change – or modulate – that pulse width. If you repeat this on-off pattern fast enough with an LED, the result is as if the signal is a steady voltage between 0 and 3.3 V controlling the brightness of the LED.

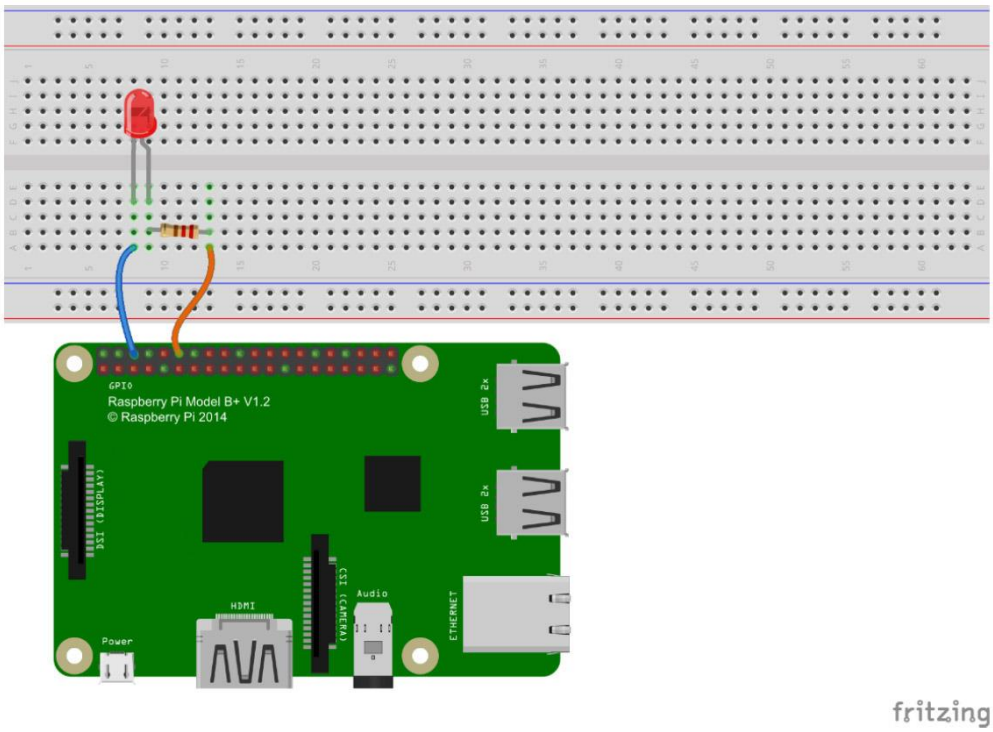
A duty cycle is the percentage of one period in which a signal is active. A period is a time it takes for a signal to complete an on-off cycle. As a formula, a duty cycle may be expressed as:

$$D = \frac{T}{P} \times 100\%$$

A 60 % duty cycle means the signal is on 60 % of the time but off for 40 % of the time. The "on time" for a 60 % duty cycle could be a fraction of a second, a day, or even week, depending on the length of the period.



Experiment



C Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_C_code_for_RaspberryPi/04_PwmLed
2. Compile:
   gcc PwmLed.c –o PwmLed –lwiringPi
3. Run:
   sudo ./PwmLed

Python Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_Python_code_for_RaspberryPi/
2. Run:
   sudo python 04_PwmLed.py

Press ENTER and you will see a gradual change of the LED luminance.

Through this experiment, you should have mastered the principle of PWM and how to programme the Raspberry Pi® with PWM. You can apply this technology to DC motor speed regulation in the future.

Programming

C Programming

```
#include <wiringPi.h>
#include <stdio.h>
#define LedPin 1
int main(void)
{
int i;
if(wiringPiSetup()  ==  -1){  //when  initialize  wiring  failed,print
messageto screen
printf("setup wiringPi failed !");
return 1;
}

pinMode(LedPin, PWM_OUTPUT);//pwm output mode
while(1){
for(i=0;i<1024;i++){

pwmWrite(LedPin, i);
delay(2);
}
delay(1000);
for(i=1023;i>=0;i--){
pwmWrite(LedPin, i);
delay(2);
}
}
return 0;
}
```

Python Programming

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time
LedPin = 12
def setup():
global p
GPIO.setmode(GPIO.BOARD) # Numbers GPIOs by physical location
GPIO.setup(LedPin, GPIO.OUT) # Set LedPin's mode is output
GPIO.output(LedPin, GPIO.LOW) # Set LedPin to low(0V)
p = GPIO.PWM(LedPin, 1000) # set Frequece to 1KHz
p.start(0) # Duty Cycle = 0
def loop():
while True:
for dc in range(0, 101, 4): # Increase duty cycle: 0~100
p.ChangeDutyCycle(dc) # Change duty cycle
time.sleep(0.05)
time.sleep(1)

for dc in range(100, -1, -4): # Decrease duty cycle: 100~0
p.ChangeDutyCycle(dc)
time.sleep(0.05)
time.sleep(1)
def destroy():
p.stop()
GPIO.output(LedPin, GPIO.HIGH) # turn off all leds

GPIO.cleanup()
if __name__ == '__main__': # Program start from here
setup()
try:
loop()
except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the child program
destroy() will be executed.
destroy()
```

6.5 RGB LED

We will gradually increase and decrease the luminance of an LED with PWM.

Required Hardware

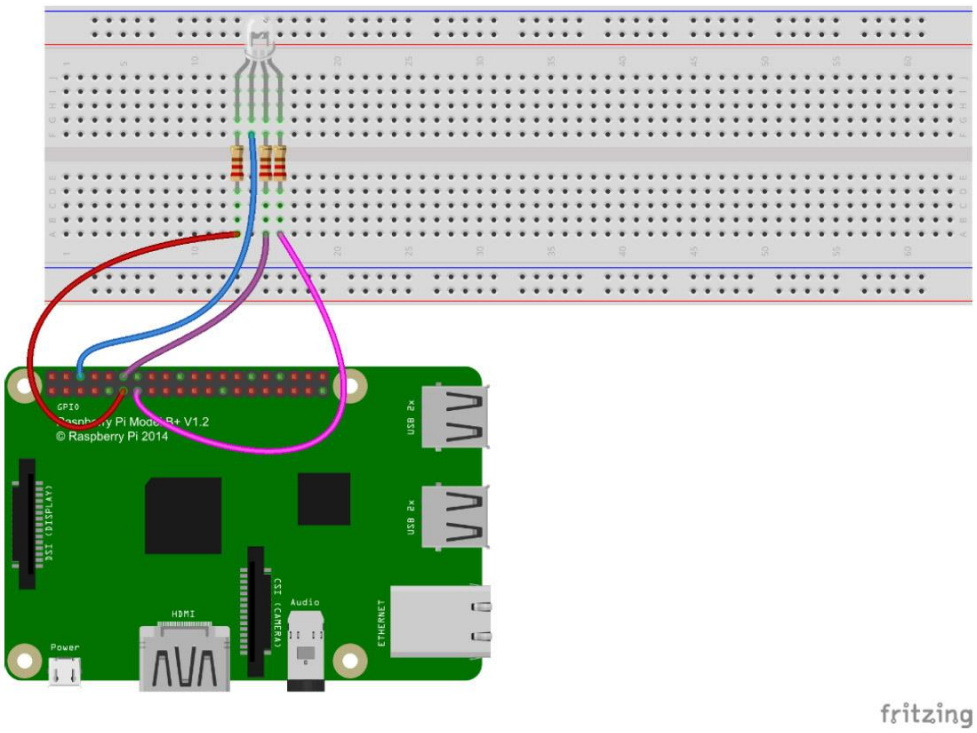
- 1 x Raspberry Pi®
- 1 x breadboard
- 1 x RGB LED
- 3 x 220 Ω resistor
- jumper wires as needed

RGB LEDs emit light in various colours. They package three LEDs of red, green and blue into a transparent or semi-transparent plastic shell, and have four pins. The three primary colours can be mixed into various colours by brightness. The LED brightness can be adjusted with PWM. The Raspberry Pi® has only one channel for

hardware PWM output, but it needs three channels to control the RGB LED, which means it is diffeicult to control the RGB LED with the hardware PWM of the Raspberry Pi®. Fortunately, the softPwm library simulates PWM (softPwm) by programming.

RGB LEDs can be categorized into common anode type and common cathode type. In this experiment, the latter is used.

Experiment



C Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_C_code_for_RaspberryPi/05_RGB
2. Compile:
   gcc rgb.c –o rgb –lwiringPi –lpthread
3. Run:
   sudo ./rgb

Python Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_Python_code_for_RaspberryPi/
2. Run:
   sudo python 05_rgb.py

You should see the LED emit light of different colours.

You can also modify the parameters of the function *ledColorSet()* by yourself, then compile and run the code to see the colour changes of the RGB LED.

Programming

C Programming

```
#include <wiringPi.h>
#include <softPwm.h>
#include <stdio.h>
#define uchar unsigned char
#define LedPinRed 0
#define LedPinGreen 1
#define LedPinBlue 2
void ledInit(void)
{
softPwmCreate(LedPinRed, 0, 100);
softPwmCreate(LedPinGreen,0, 100);
softPwmCreate(LedPinBlue, 0, 100);
}
void ledColorSet(uchar r_val, uchar g_val, uchar b_val)
{
softPwmWrite(LedPinRed, r_val);
softPwmWrite(LedPinGreen, g_val);
softPwmWrite(LedPinBlue, b_val);
}
int main(void)
{
int i;
if(wiringPiSetup() == -1){ //when initialize wiring failed,print
messageto screen
printf("setup wiringPi failed !");
return 1;
}
//printf("linker LedPin : GPIO %d(wiringPi pin)\n",LedPin); //when
initialize wiring successfully,print message to screen
ledInit();
while(1){
ledColorSet(0xff,0x00,0x00); //red
delay(500);
ledColorSet(0x00,0xff,0x00); //green
delay(500);
ledColorSet(0x00,0x00,0xff); //blue
delay(500);
ledColorSet(0xff,0xff,0x00); //yellow
delay(500);
ledColorSet(0xff,0x00,0xff); //pick
delay(500);
ledColorSet(0xc0,0xff,0x3e);
delay(500);
ledColorSet(0x94,0x00,0xd3);
```

```
delay(500);
ledColorSet(0x76,0xee,0x00);
delay(500);
ledColorSet(0x00,0xc5,0xcd);
delay(500);
}
return 0;
}
```

Python Programming

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time
colors = [0xFF0000, 0x00FF00, 0x0000FF, 0xFFFF00, 0xFF00FF, 0x00FFFF]
pins = {'pin_R':11, 'pin_G':12, 'pin_B':13} # pins is a dict
GPIO.setmode(GPIO.BOARD) # Numbers GPIOs by physical location
for i in pins:
GPIO.setup(pins[i], GPIO.OUT) # Set pins' mode is output
GPIO.output(pins[i], GPIO.HIGH) # Set pins to high(+3.3V) to off led
p_R = GPIO.PWM(pins['pin_R'], 2000) # set Frequece to 2KHz
p_G = GPIO.PWM(pins['pin_G'], 2000)
p_B = GPIO.PWM(pins['pin_B'], 5000)
p_R.start(0) # Initial duty Cycle = 0(leds off)
p_G.start(0)
p_B.start(0)
def map(x, in_min, in_max, out_min, out_max):
return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min
def setColor(col): # For example : col = 0x112233
R_val = (col & 0x110000) >> 16
G_val = (col & 0x001100) >> 8
B_val = (col & 0x000011) >> 0

R_val = map(R_val, 0, 255, 0, 100)
G_val = map(G_val, 0, 255, 0, 100)
B_val = map(B_val, 0, 255, 0, 100)

p_R.ChangeDutyCycle(R_val) # Change duty cycle
p_G.ChangeDutyCycle(G_val)
p_B.ChangeDutyCycle(B_val)
```

```
try:
while True:
for col in colors:
setColor(col)
time.sleep(0.5)
except KeyboardInterrupt:
p_R.stop()
p_G.stop()
p_B.stop()

for i in pins:
GPIO.output(pins[i], GPIO.HIGH) # Turn off all leds
GPIO.cleanup()
```

6.6 Buzzer

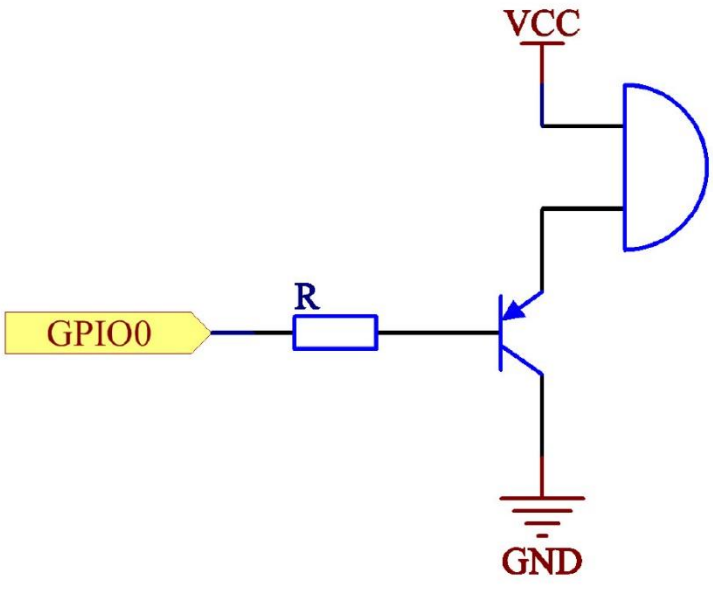
Required Hardware

- 1 x Raspberry Pi®
- 1 x breadboard
- 1 x buzzer (active)
- 1 x PNP transistor (8550)
- 3 x 1 kΩ resistor
- jumper wires as needed

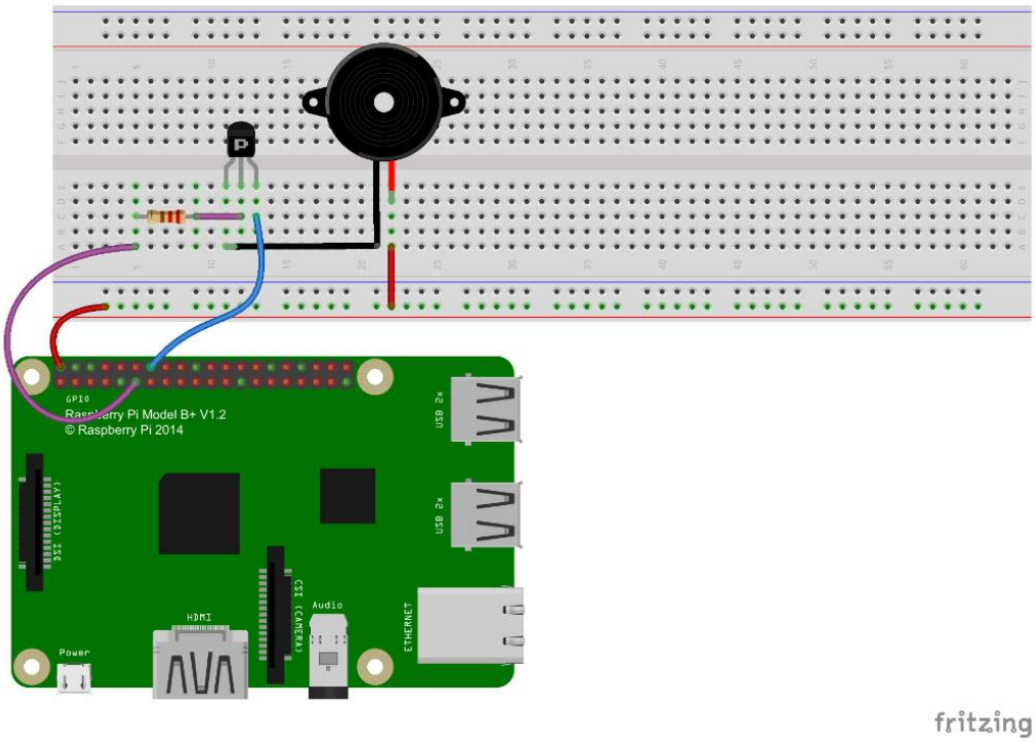
An electronic buzzer is widely used in computers, alarms, electronic toys, etc.

Buzzers can be categorized as active or passive. An active buzzer has a built-in oscillating source, so it will make sounds when electrified. A passive buzzer does not have an oscillating source and will not buzz if DC signals are used. Instead, you need to use square waves whose frequency is between 2K and 5K to drive it. The active buzzer is often more expensive than the passive one because of multiple built-in oscillating circuits.

Here, an active buzzer is used. When the GPIO output is supplied with low level (0 V) by programming, the transistor will conduct because of current saturation and the buzzer will make a sound. When a high level is supplied to the IO of the Raspberry Pi®, the transistor will be cut off and the buzzer will not make a sound.



Experiment



C Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_C_code_for_RaspberryPi/06_Beep/
2. Compile:
   gcc beep.c –o beep –lwiringPi
3. Run:
   sudo ./beep

Python Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_Python_code_for_RaspberryPi/
2. Run:
   sudo python 06_beep.py

You should the buzzer make a sound.

Using a passive buzzer, you can make it sound "do re mi fa sol la si do" with some basic programming knowledge.

Programming

C Programming

```
#include <wiringPi.h>
#include <stdio.h>
#define BeepPin 0
int main(void)
{
if(wiringPiSetup()  ==  -1){  //when  initialize  wiring  failed,print
messageto screen
printf("setup wiringPi failed !");
return 1;
}
pinMode(BeepPin, OUTPUT); //set GPIO0 output
while(1){
digitalWrite(BeepPin, LOW); //beep on
delay(100); //delay
digitalWrite(BeepPin, HIGH); //beep off
delay(100); //delay
}
return 0;
}
```

Python Programming

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time
BeepPin = 11 # pin11
def setup():
GPIO.setmode(GPIO.BOARD) # Numbers GPIOs by physical location
GPIO.setup(BeepPin, GPIO.OUT) # Set BeepPin's mode is output
GPIO.output(BeepPin, GPIO.HIGH) # Set BeepPin high(+3.3V) to off beep
def loop():
while True:
GPIO.output(BeepPin, GPIO.LOW)
time.sleep(0.1)
GPIO.output(BeepPin, GPIO.HIGH)
time.sleep(0.1)
def destroy():
GPIO.output(BeepPin, GPIO.HIGH) # beep off
GPIO.cleanup() # Release resource
```

```
if __name__ == '__main__': # Program start from here
print 'Press Ctrl+C to end the program...'
setup()
try:
loop()
except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the child program
destroy() will be executed.
destroy()
```

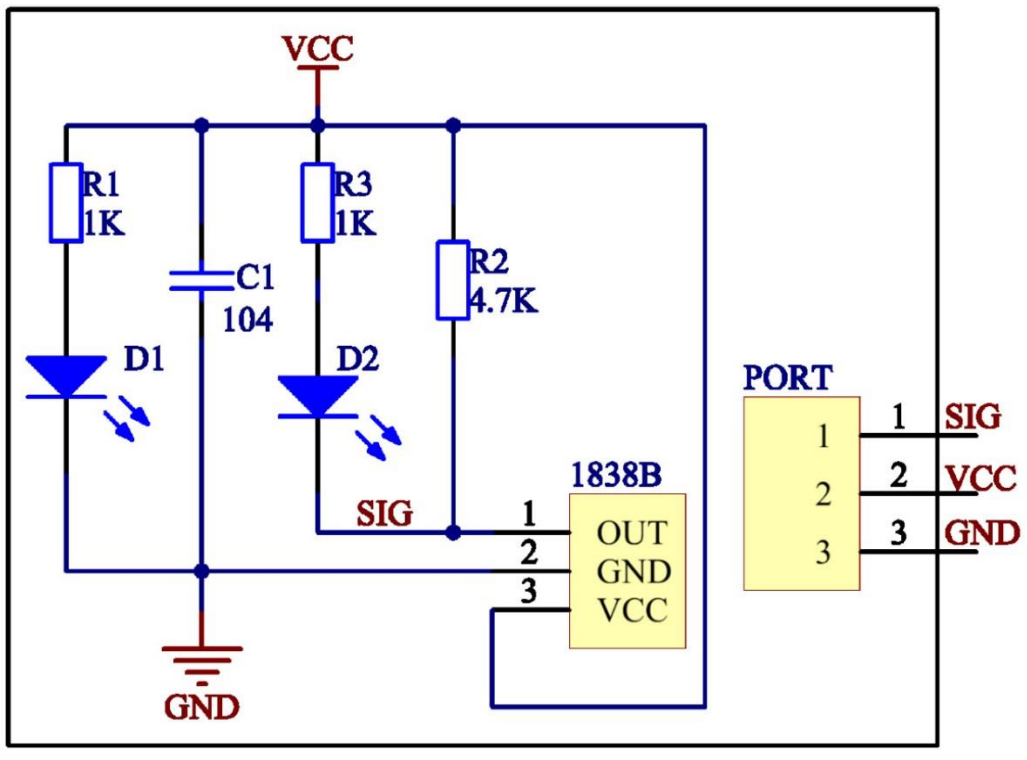
6.7 IR Receiver Module

An IR receiver is a component which receives IR signals and can independently receive IR rays and output signals compatible with TLL level.

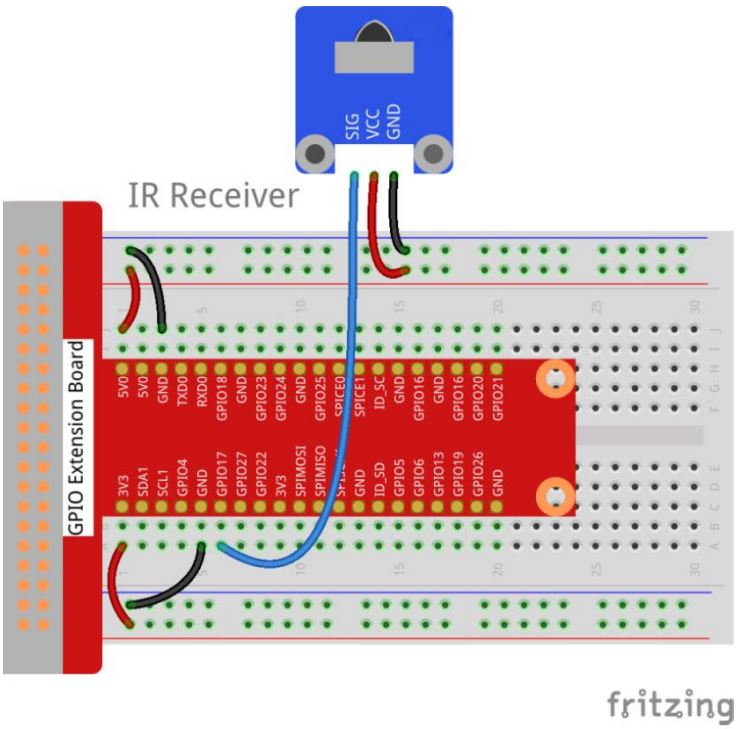
Required Hardware

- 1 x Raspberry Pi®
- 1 x breadboard
- 4 x jumper wire (male to male, 2 x red and 2 x black)
- 1 x network cable (or USB wireless network adapter)
- 1 x IR receiver module
- 1 x IR remote controller
- 1 x 3-pin anti-reverse cable

Here, we will send signals to the IR receiver by pressing buttons on the IR remote controller. The counter will add 1 every time it receives signals, i.e. the increased number indicates the received IR signals.



Experiment



The LED on the module will be blinking.

Programming

C Programming

```
#include <wiringPi.h>
#include <stdio.h>
#define IR 0
int cnt = 0;
void myISR(void)
{
printf("Received infrared. cnt = %d\n", ++cnt);
}
int main(void)
{
if(wiringPiSetup()  ==  -1){  //when  initialize  wiring  failed,print
messageto screen
printf("setup wiringPi failed !");
return 1;
}

if(wiringPiISR(IR, INT_EDGE_FALLING, &myISR) == -1){
printf("setup ISR failed !");
return 1;
```

```
}
//pinMode(IR, INPUT);
while(1);

return 0;
}
```

Python Programming

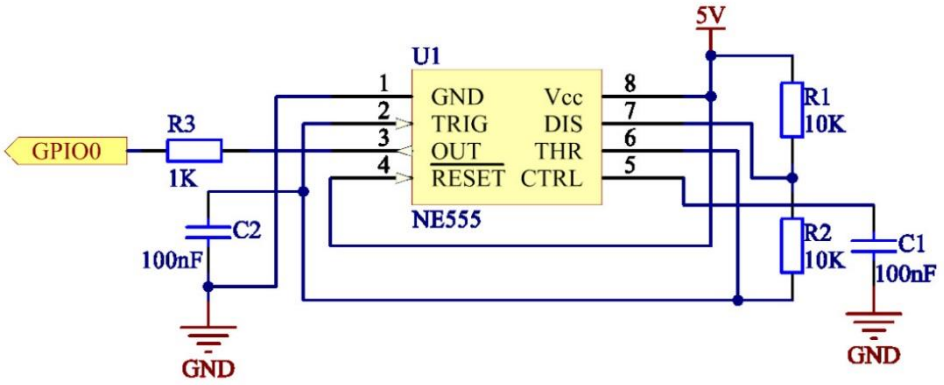
```
#!/usr/bin/env python
import RPi.GPIO as GPIO
IrPin = 11
count = 0
def setup():
GPIO.setmode(GPIO.BOARD) # Numbers GPIOs by physical location
GPIO.setup(IrPin, GPIO.IN, pull_up_down=GPIO.PUD_UP)
def cnt(ev=None):
global count
count += 1
print 'Received infrared. cnt = ', count
def loop():
GPIO.add_event_detect(IrPin,  GPIO.FALLING,  callback=cnt)  #  wait  for
falling
while True:
pass # Don't do anything
def destroy():
GPIO.cleanup() # Release resource
if __name__ == '__main__': # Program start from here
setup()
try:
loop()
except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the child program
destroy() will be executed.
destroy()
```

6.8 555 Timer

Required Hardware

- 1 x Raspberry Pi®
- 1 x breadboard
- 1 x NE555
- 3 x resistor (1 x 1 kΩ, 2 x 10 kΩ)
- 2 x 100 nF capacitor
- jumper wires as needed

A 555 timer is a medium-sized IC device which combines analogue and digital functions. The 555 timer can work under three modes. Here, the astable mode is used to generate square waves.



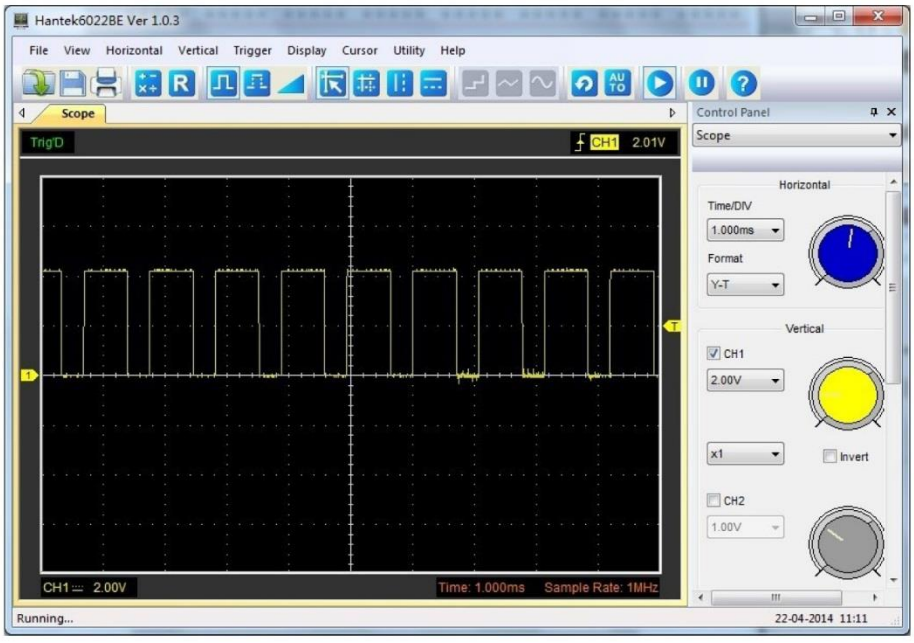
Under the astable mode, the frequency of the output waveform of the 555 timer is defined by R_1, R_2 and:

$$f = \frac{1}{\ln 2 * C_2 * (R_1 + 2R_2)}$$

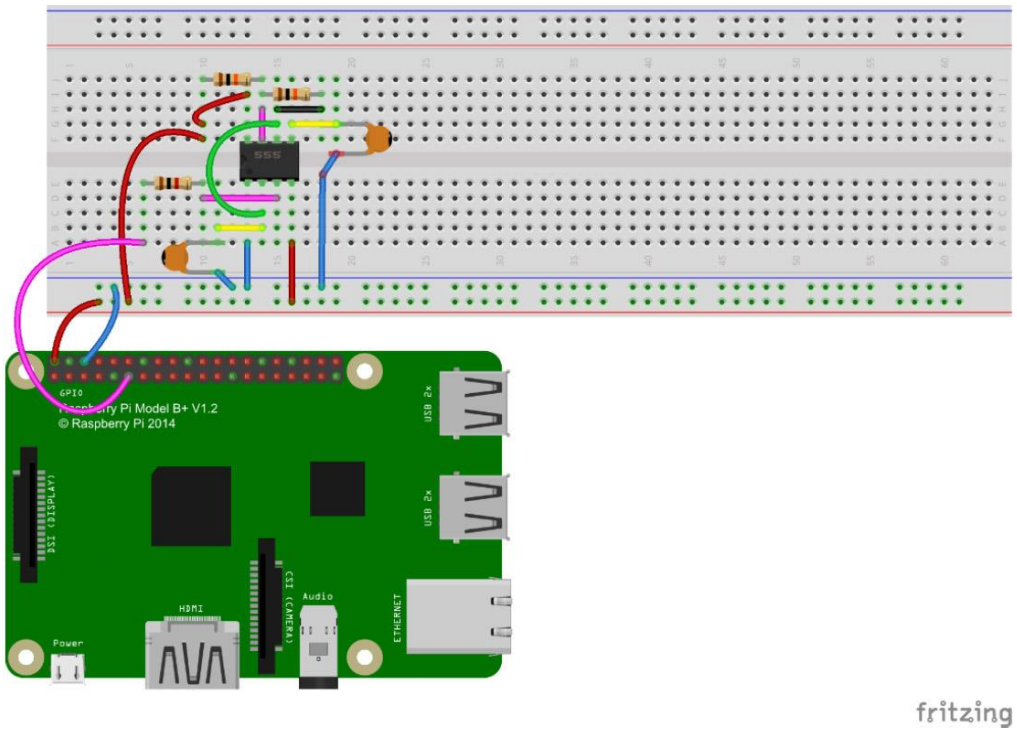
In the above circuit, R_1=R_2=10 kΩ=Ω; =100 nF=F. Frequency:

$$f = \frac{1}{\ln 2 * 10^{-7} * (10^4 + 2 * 10^4)} \approx 481\text{Hz}$$

After connecting the circuit, use an oscilloscope to observe the frequency of the output waveform. It is consistent with the above calculated result.



Experiment



C Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_C_code_for_RaspberryPi/09_Timer555/
2. Compile:
   gcc timer555.c –o timer555 –lwiringPi
3. Run:
   sudo ./timer555

Python Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_Python_code_for_RaspberryPi/
2. Run:
   sudo python 09_timer555.py

You should see data on the display, which are square waves generated by the 555 timer. The programme counts pulses by interrupt as we have learned previously.

The above circuits outputs square waves with constant frequency and duty cycle. You can simple change this circuit to adjust the frequency and duty cycle.

Programming

C Programming

```
#include <stdio.h>
#include <string.h>
#include <errno.h>
#include <stdlib.h>
#include <wiringPi.h>
#define Pin0 0
static volatile int globalCounter = 0 ;
void exInt0_ISR(void) //GPIO0 interrupt service routine
{
++globalCounter;
}
int main (void)
{
if(wiringPiSetup() < 0){
fprintf(stderr, "Unable to setup wiringPi:%s\n",strerror(errno));
return 1;
}
wiringPiISR(Pin0, INT_EDGE_FALLING, &exInt0_ISR);
while(1){
printf("Current pluse number is : %d\n", globalCounter);
}
return 0;
}
```

Python Programming

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
SigPin = 11 # pin11
g_count = 0
def count(ev=None):
global g_count
g_count += 1
def setup():
GPIO.setmode(GPIO.BOARD) # Numbers GPIOs by physical location
GPIO.setup(SigPin, GPIO.IN, pull_up_down=GPIO.PUD_UP) # Set Pin's mode
is input, and pull up to high level(3.3V)
GPIO.add_event_detect(SigPin, GPIO.RISING, callback=count) # wait for
rasing
def loop():
while True:
```

```
print 'g_count = %d' % g_count
def destroy():
GPIO.cleanup() # Release resource
if __name__ == '__main__': # Program start from here
setup()
try:
loop()
except KeyboardInterrupt: # When 'Ctrl+C' is pressed, the child program
destroy() will be executed.
destroy()
```

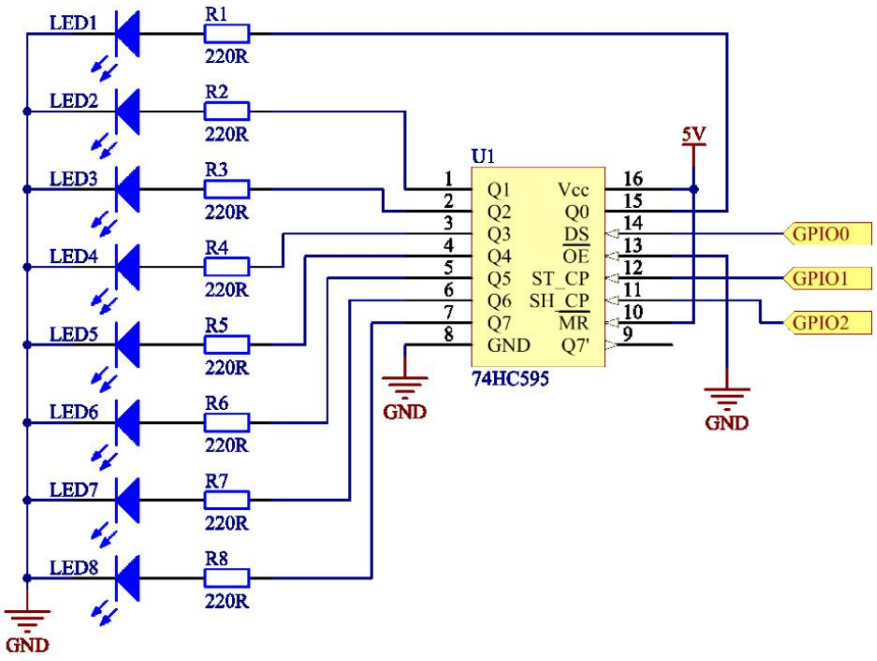
6.9 Driving LEDs by 74HC595

We will use the 74HC595 to make eight LEDs blink regularly.

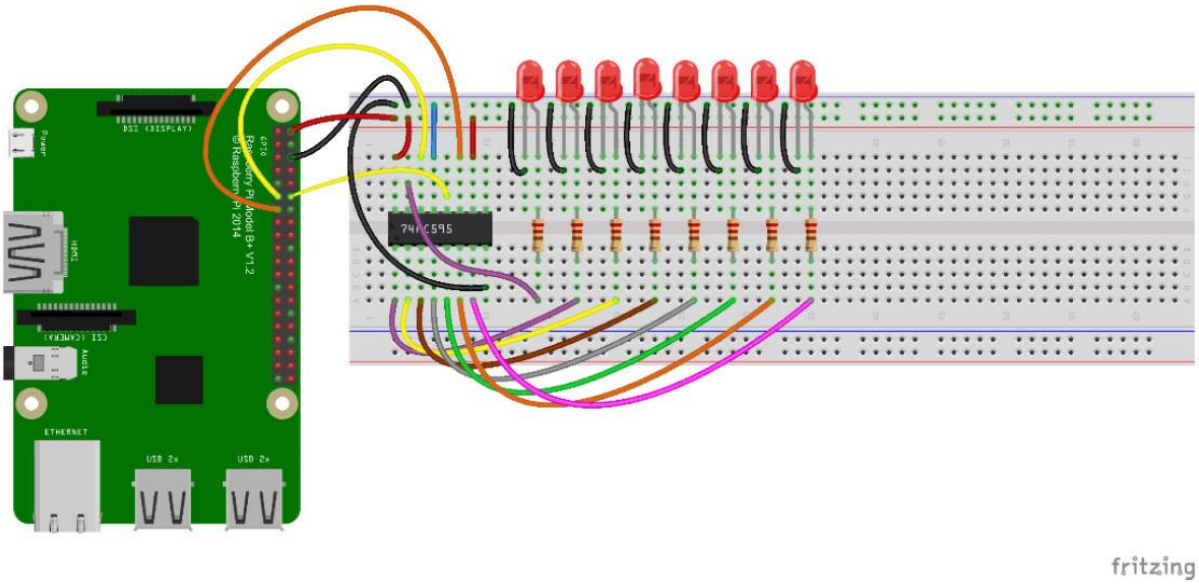
Required Hardware

- 1 x Raspberry Pi®
- 1 x breadboard
- 1 x 74HC595
- 8 x LED
- 3 x 220 Ω resistor
- jumper wires as needed

The 74HC595 is a silicon CMOS device, which has an 8-bit shift register and a memory with three-state output function. Compatible with low voltage TTL circuit, the 74HC595 can transform the serial input of 8-bit data into parallel output of 8-bit data. It is often used to extend the GPIO for embedded system and drive low power devices.



Experiment



C Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_C_code_for_RaspberryPi/10_74HC595_LED/
2. Compile:
   gcc 74HC595_LED.c –o 74HC595_LED –lwiringPi
3. Run:
   sudo ./74HC595_LED

Python Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_Python_code_for_RaspberryPi/
2. Run:
   sudo python 10_74HC595_LED.py

You should see eight LEDs blinking regularly.

Here, three Raspberry Pi® GPIOs are used to separately control eight LEDs based on the 74HC595. The 74HC595 has another powerful function: cascade. With cascade, you can use a microprocessor like three Raspberry Pi® IOs to control more peripherals.

Programming

C Programming

```
#include <wiringPi.h>
#include <stdio.h>
#define SDI 0 //serial data input
#define RCLK 1 //memory clock input(STCP)
#define SRCLK 2 //shift register clock input(SHCP)
unsigned char LED[8] = {0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80};
```

```
void pulse(int pin)
{
digitalWrite(pin, 0);
digitalWrite(pin, 1);
}
void SIPO(unsigned char byte)
{
int i;
for(i=0;i<8;i++){
digitalWrite(SDI, ((byte & (0x80 >> i)) > 0));
pulse(SRCLK);
}
}
void init(void)
{
pinMode(SDI, OUTPUT); //make P0 output
pinMode(RCLK, OUTPUT); //make P0 output
pinMode(SRCLK, OUTPUT); //make P0 output
digitalWrite(SDI, 0);
digitalWrite(RCLK, 0);
digitalWrite(SRCLK, 0);
}
int main(void)
{
int i;
if(wiringPiSetup()  ==  -1){  //when  initialize  wiring  failed,print
messageto screen
printf("setup wiringPi failed !");
return 1;
}
init();
while(1){
for(i=0;i<8;i++){
SIPO(LED[i]);
pulse(RCLK);
delay(150);
//printf("i = %d\n",i);
}
```

```
delay(500);
for(i=0;i<3;i++){
SIPO(0xff);
pulse(RCLK);
delay(100);
SIPO(0x00);
pulse(RCLK);
delay(100);
}
delay(500);
// digitalWrite(RCLK,0);
for(i=0;i<8;i++){
SIPO(LED[8-i-1]);
pulse(RCLK);
delay(150);
}
delay(500);
for(i=0;i<3;i++){
SIPO(0xff);
pulse(RCLK);
delay(100);
SIPO(0x00);
pulse(RCLK);
delay(100);
}
delay(500);
}
return 0;
}
```

Python Programming

```
import RPi.GPIO as GPIO
import time
SDI = 11
RCLK = 12
SRCLK = 13
#=============== LED Mode Defne ================
# You can define yourself, in binay, and convert it to Hex
# 8 bits a group, 0 means off, 1 means on
# like : 0101 0101, means LED1, 3, 5, 7 are on.(from left to right)
# and convert to 0x55.
```

```
LED0 = [0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80] #original mode
LED1 = [0x01,0x03,0x07,0x0f,0x1f,0x3f,0x7f,0xff] #blink mode 1
LED2 = [0x01,0x05,0x15,0x55,0xb5,0xf5,0xfb,0xff] #blink mode 2
LED3 = [0x02,0x03,0x0b,0x0f,0x2f,0x3f,0xbf,0xff] #blink mode 3
#================================================
def print_msg():
print 'Program is running...'
print 'Please press Ctrl+C to end the program...'
def setup():
GPIO.setmode(GPIO.BOARD) # Number GPIOs by its physical location
GPIO.setup(SDI, GPIO.OUT)
GPIO.setup(RCLK, GPIO.OUT)
GPIO.setup(SRCLK, GPIO.OUT)
GPIO.output(SDI, GPIO.LOW)
GPIO.output(RCLK, GPIO.LOW)
GPIO.output(SRCLK, GPIO.LOW)
def hc595_in(dat):
for bit in range(0, 8):
GPIO.output(SDI, 0x80 & (dat << bit))
GPIO.output(SRCLK, GPIO.HIGH)
time.sleep(0.001)
GPIO.output(SRCLK, GPIO.LOW)
def hc595_out():
GPIO.output(RCLK, GPIO.HIGH)
time.sleep(0.001)
GPIO.output(RCLK, GPIO.LOW)
def loop():
WhichLeds = LED0 # Change Mode, modes from LED0 to LED3
sleeptime = 0.1 # Change speed, lower value, faster speed
while True:
for i in range(0, len(WhichLeds)):
hc595_in(WhichLeds[i])
hc595_out()
time.sleep(sleeptime)

for i in range(len(WhichLeds)-1, -1, -1):
hc595_in(WhichLeds[i])
hc595_out()
time.sleep(sleeptime)
def destroy(): # When program ending, the function is executed.
GPIO.cleanup()
```

```
if __name__ == '__main__': # Program starting from here
print_msg()
setup()
try:
loop()
except KeyboardInterrupt:
destroy()
```

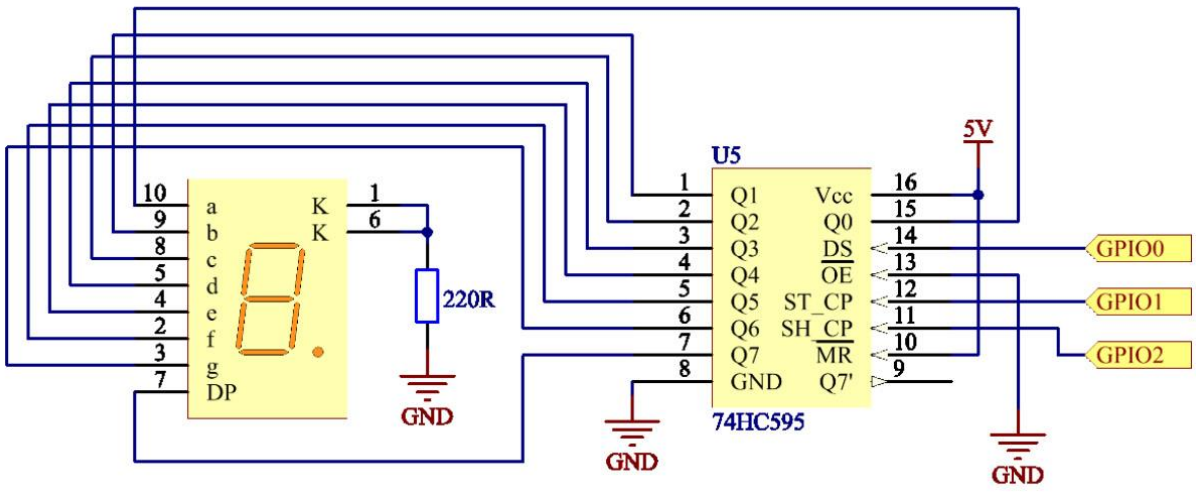
6.10 Driving a 7-Segment Display by 74HC595

We will learn to use the 74HC595 to drive a 7-segment display to cycle a figure from 0 to 9.

Required Hardware

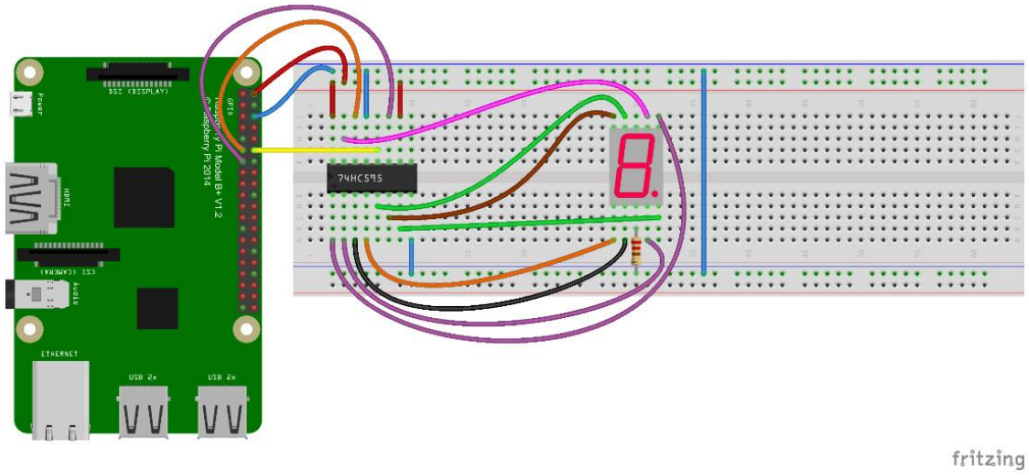
- 1 x Raspberry Pi®
- 1 x breadboard
- 1 x 74HC595
- 1 x 7-segment display
- 3 x 1 kΩ resistor
- jumper wires as needed

7-segment displays can be categorized into two types: a common cathode and a common anode, depending on the different light-emitting diode connections. They are widely used in electronic appliances, especially home appliances such as air conditionings, water heaters, etc.



We use a common cathode 7-segment display, which connects all the LED cathodes to form a common cathode (COM) electrode. It should be connected to ground. When the anode of an LED in a certain segment is at high level, the corresponding segment will light. When it is at low, the segment will not light.

Experiment



C Programming

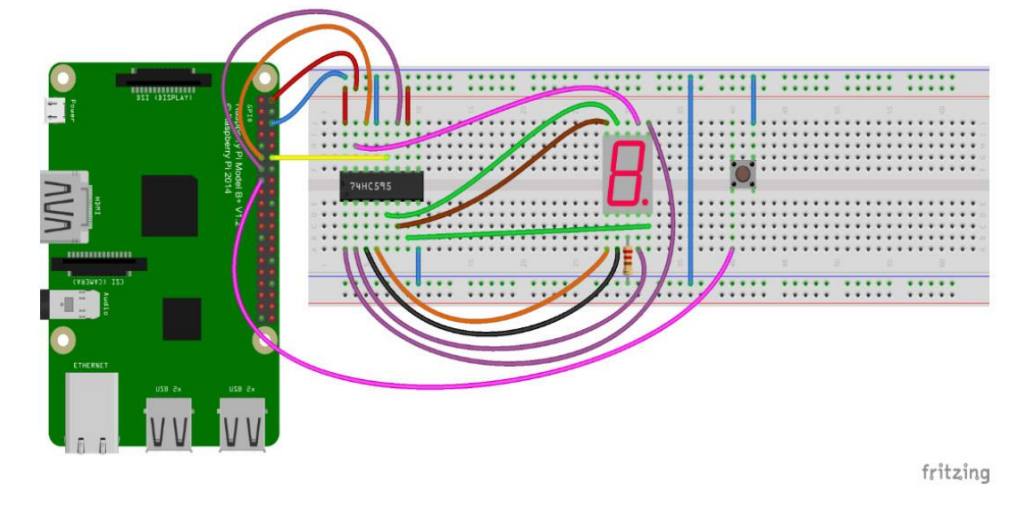
1. Change directory:
   cd/home/pi/IDUINO_SuperKit_C_code_for_RaspberryPi/11_Segment/
2. Compile:
   gcc segment1.c –o segment1 –lwiringPi
3. Run:
   sudo ./segment1

Python Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_Python_code_for_RaspberryPi/
2. Run:
   sudo python 11_segment.py

You should see the 7-segment display cycle from 0 to 9, and from A to F.

You can slightly modify the hardware and software based on the basic configuration to make a dice. For hardware, add a button to the original board.



1. Change directory:
   gcc dice.c –o dice –lwiringPi
2. Run:
   sudo ./dice

You should see numbers between 0 and 6 flashing quickly. Press the button on the breadboard and the display will display a random number between 0 and 6 for 2 seconds, and then circularly display random numbers between 0 and 6.

Programming

C Programming

```
#include <wiringPi.h>
#include <stdio.h>
#define SDI 0 //serial data input
#define RCLK 1 //memory clock input(STCP)
#define SRCLK 2 //shift register clock input(SHCP)
unsigned char SegCode[17] =
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,
0x79,0x71,0x80};
void init(void)
{
pinMode(SDI, OUTPUT); //make P0 output
pinMode(RCLK, OUTPUT); //make P0 output
pinMode(SRCLK, OUTPUT); //make P0 output
digitalWrite(SDI, 0);
digitalWrite(RCLK, 0);
digitalWrite(SRCLK, 0);
}
void hc595_shift(unsigned char dat)
{
int i;
for(i=0;i<8;i++){
digitalWrite(SDI, 0x80 & (dat << i));
digitalWrite(SRCLK, 1);
delay(1);
digitalWrite(SRCLK, 0);
}
digitalWrite(RCLK, 1);
delay(1);
digitalWrite(RCLK, 0);
}
int main(void)
```

```
{
int i;
if(wiringPiSetup()  ==  -1){  //when  initialize  wiring  failed,print
messageto screen
printf("setup wiringPi failed !");
return 1;
}
init();
while(1){
for(i=0;i<17;i++){
hc595_shift(SegCode[i]);
delay(500);
}
}
return 0;
}
```

Python Programming

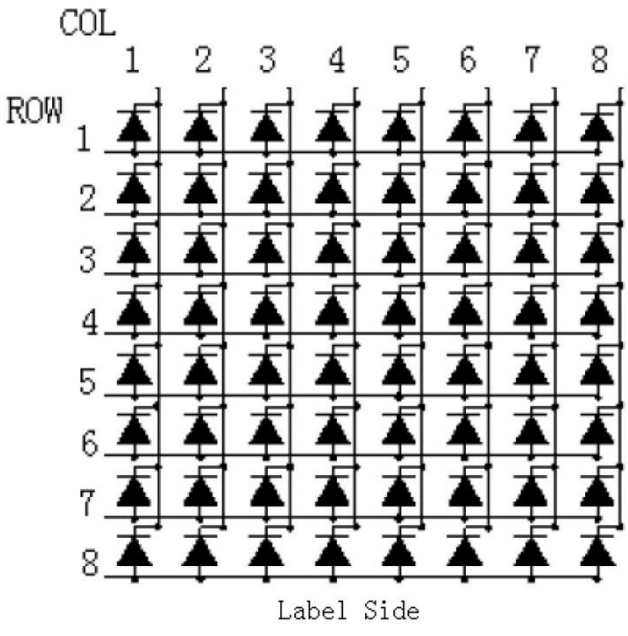
```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time
SDI = 11
RCLK = 12
SRCLK = 13
segCode                                                                =
[0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,
0x79,0x71,0x80]
def print_msg():
print 'Program is running...'
print 'Please press Ctrl+C to end the program...'
def setup():
GPIO.setmode(GPIO.BOARD) #Number GPIOs by its physical location
GPIO.setup(SDI, GPIO.OUT)
GPIO.setup(RCLK, GPIO.OUT)
GPIO.setup(SRCLK, GPIO.OUT)
GPIO.output(SDI, GPIO.LOW)
GPIO.output(RCLK, GPIO.LOW)
GPIO.output(SRCLK, GPIO.LOW)
def hc595_shift(dat):
for bit in range(0, 8):
GPIO.output(SDI, 0x80 & (dat << bit))
GPIO.output(SRCLK, GPIO.HIGH)
```

```
time.sleep(0.001)
GPIO.output(SRCLK, GPIO.LOW)
GPIO.output(RCLK, GPIO.HIGH)
time.sleep(0.001)
GPIO.output(RCLK, GPIO.LOW)
def loop():
while True:
for i in range(0, len(segCode)):
hc595_shift(segCode[i])
time.sleep(0.5)
def destroy(): #When program ending, the function is executed.
GPIO.cleanup()
if __name__ == '__main__': #Program starting from here
print_msg()
setup()
try:
loop()
except KeyboardInterrupt:
destroy()
```

6.11 Driving a Dot Matrix by 74HC595

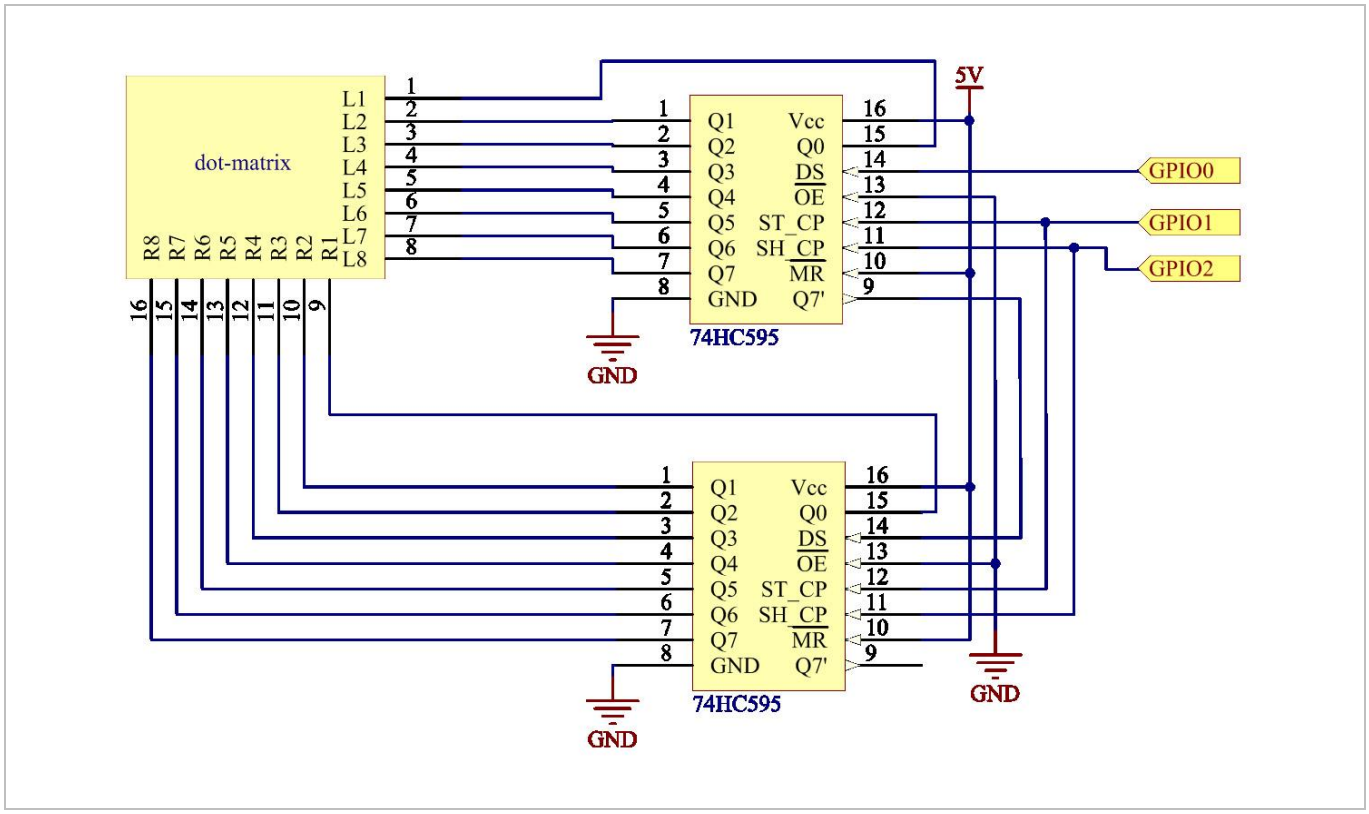
Required Hardware

- 1 x Raspberry Pi®
- 1 x breadboard
- 2 x 74HC595
- 1 x dot matrix
- jumper wires as needed

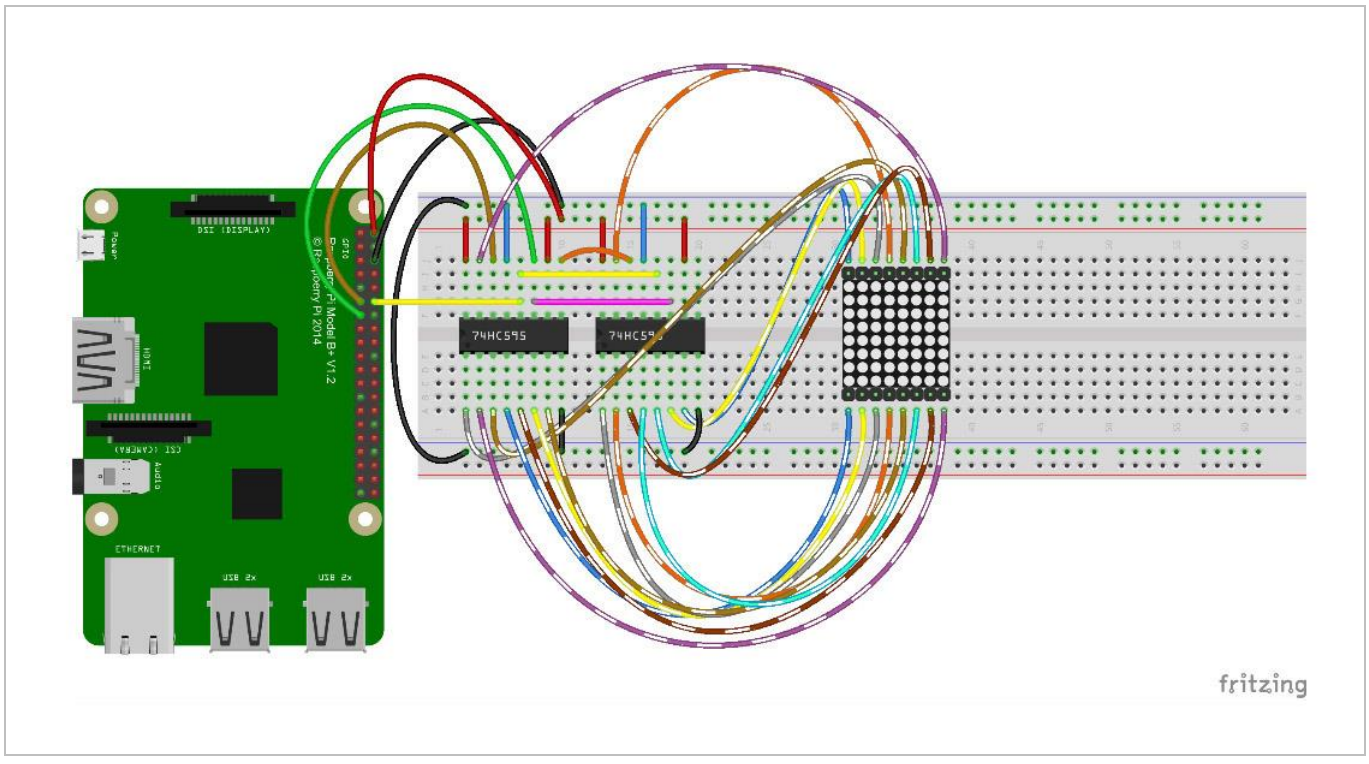


Pin number corresponding to rows and columns:								
COL	1	2	3	4	5	6	7	8
Pin No.	13	3	4	10	6	11	15	16
ROW	1	2	3	4	5	6	7	8
Pin No.	9	14	8	12	1	7	2	5

The 8 x 8 dot matrix is made up of 64 LEDs, each LED is placed at the cross point of a row and a column. When the electrical level of a certain row is high and the electrical level of a certain column is low, then the corresponding LED will light. If you want to light the LED on the first dot, you should set ROW 1 to high and COL 1 to low.



Experiment



C Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_C_code_for_RaspberryPi/12_DotMatrix/
2. Compile:
   gcc dotMatrix.c –o dotMatrix –lwiringPi
3. Run:
   sudo ./dotMatrix

Python Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_Python_code_for_RaspberryPi/
2. Run:
   sudo python 12_DotMatrix.py

You should the LEDs light.

Programming

C Programming

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

#define SDI 0 //serial data input
#define RCLK 1 //memory clock input(STCP)
#define SRCLK 2 //shift register clock input(SHCP)
unsigned                char                code_H[20]                =
{0x01,0xff,0x80,0xff,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0xff,0xff,
0xff,0xff,0xff,0xff,0xff,0xff};
unsigned                char                code_L[20]                =
{0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xfd,
0xfb,0xf7,0xef,0xdf,0xbf,0x7f};
//unsigned char code_L[8] = {0x00,0x00,0x3c,0x42,0x42,0x3c,0x00,0x00};
//unsigned char code_H[8] = {0xff,0xe7,0xdb,0xdb,0xdb,0xdb,0xe7,0xff};
//unsigned char code_L[8] = {0xff,0xff,0xc3,0xbd,0xbd,0xc3,0xff,0xff};
//unsigned char code_H[8] = {0x00,0x18,0x24,0x24,0x24,0x24,0x18,0x00};
void init(void)
{
pinMode(SDI, OUTPUT); //make P0 output
pinMode(RCLK, OUTPUT); //make P0 output
pinMode(SRCLK, OUTPUT); //make P0 output
digitalWrite(SDI, 0);
digitalWrite(RCLK, 0);
digitalWrite(SRCLK, 0);
}
void hc595_in(unsigned char dat)
```

```
{
int i;
for(i=0;i<8;i++){
digitalWrite(SDI, 0x80 & (dat << i));
digitalWrite(SRCLK, 1);
delay(1);
digitalWrite(SRCLK, 0);
}
}
void hc595_out()
{
digitalWrite(RCLK, 1);
delay(1);
digitalWrite(RCLK, 0);
}
int main(void)
{
int i;
if(wiringPiSetup()  ==  -1){  //when  initialize  wiring  failed,print
messageto screen
printf("setup wiringPi failed !");
return 1;
}
init();
while(1){
for(i=0;i<sizeof(code_H);i++){
hc595_in(code_L[i]);
hc595_in(code_H[i]);
hc595_out();
delay(100);
}
for(i=sizeof(code_H);i>=0;i--){
hc595_in(code_L[i]);
hc595_in(code_H[i]);
hc595_out();
delay(100);
}
}
return 0;
}
```

Python Programming

```
#!/usr/bin/env python
import RPi.GPIO as GPIO
import time
SDI = 11
RCLK = 12
SRCLK = 13
code_H =
[0x01,0xff,0x80,0xff,0x01,0x02,0x04,0x08,0x10,0x20,0x40,0x80,0xff,0xff,0xff,0xff,0xf
f,0xff,0xff,0xff]
code_L =
[0x00,0x7f,0x00,0xfe,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0x00,0xfe,0xfd,0xfb,0xf7,
0xef,0xdf,0xbf,0x7f]

def print_msg():
print 'Program is running...'
print 'Please press Ctrl+C to end the program...'

def setup():
GPIO.setmode(GPIO.BOARD) # Number GPIOs by its physical location
GPIO.setup(SDI, GPIO.OUT)
GPIO.setup(RCLK, GPIO.OUT)
GPIO.setup(SRCLK, GPIO.OUT)
GPIO.output(SDI, GPIO.LOW)
GPIO.output(RCLK, GPIO.LOW)
GPIO.output(SRCLK, GPIO.LOW)
def hc595_in(dat):
for bit in range(0, 8):
GPIO.output(SDI, 0x80 & (dat << bit))
GPIO.output(SRCLK, GPIO.HIGH)
time.sleep(0.001)
GPIO.output(SRCLK, GPIO.LOW)
def hc595_out():
GPIO.output(RCLK, GPIO.HIGH)
time.sleep(0.001)
GPIO.output(RCLK, GPIO.LOW)

def loop():
while True:
for i in range(0, len(code_H)):
hc595_in(code_L[i])
hc595_in(code_H[i])
hc595_out()
time.sleep(0.1)
```

```
for i in range(len(code_H)-1, -1, -1):
hc595_in(code_L[i])
hc595_in(code_H[i])
hc595_out()
time.sleep(0.1)
def destroy(): # When program ending, the function is executed.
GPIO.cleanup()
if __name__ == '__main__': # Program starting from here
print_msg()
setup()
try:
loop()
except KeyboardInterrupt:
destroy()
```

6.12 LCD1602 Module

We will see how to use the LCD1602 to display character strings.

Required Hardware

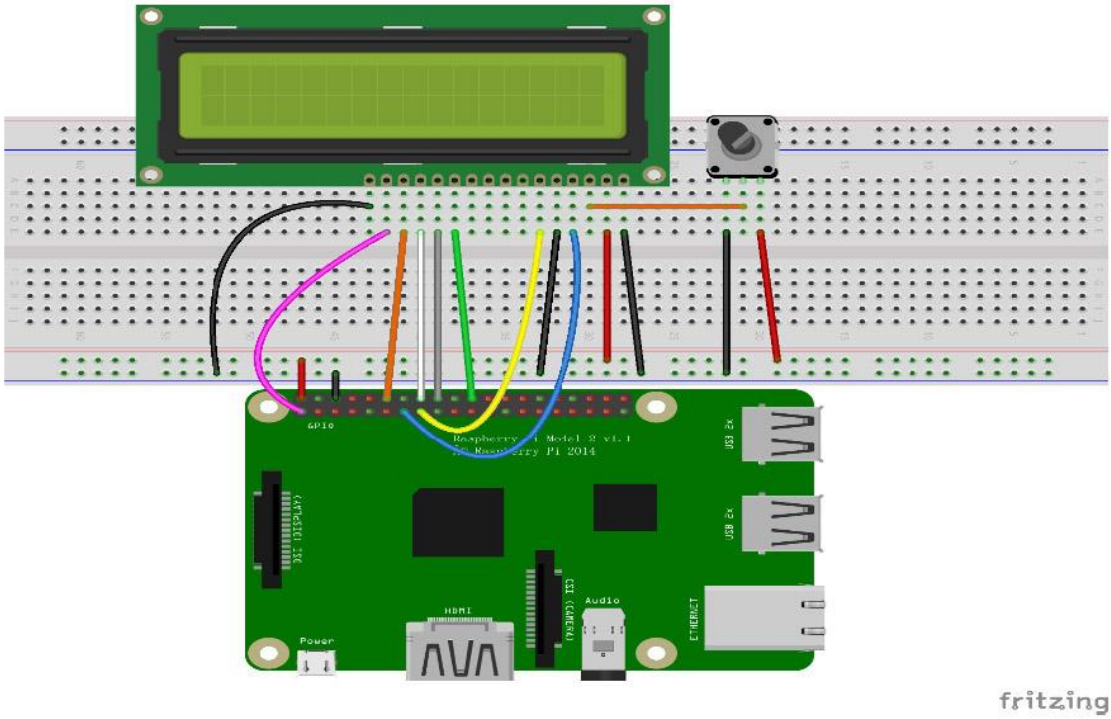
- 1 x Raspberry Pi®
- 1 x breadboard
- 1 x LCD1602
- 1 x potentiometer
- jumper wires as needed

The LCD1602 is a dot matrix used to display characters, symbols, etc. It uses the standard 16-pin port:

Pin 1 (GND)	Connect to ground.
Pin 2 (Vcc)	Connect to 5 V positive power supply.
Pin 3 (Vo)	Used to adjust the contrast of the LCD1602. The level is lowest when it is connected to positive power supply, and highest when connected to ground. You can connect a 10K potentiometer to adjust its contrast when using the LCD1602.
Pin 4 (RS)	Selects the data register when supplied with high level (1) and instruction register when supplied with low level (0).
Pin 5 (R/W)	Reads the signals when supplied with high level (1) and writes signals when supplied with low level (0). Here, we only need to write data to the LCD1602.
Pin 6 (E)	Enable pin when supplied with low level. The LCD module will execute relevant instructions.
Pin 7 (D0-D7)	Pins reading and writing data.
A and K	Power source for LCD backlight.

The LCD1602 has two operating modes: 4-bit and 8-bit. When the IOs of the microprocessor are insufficient, choose the 4-bit mode, under which only pins D4 to D7 are used.

Experiment



C Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_C_code_for_RaspberryPi/13_LCD1602/
2. Compile:
   gcc lcd1602_2.c –o lcd1602_2 –lwiringPi
3. Run:
   sudo ./lcd1602_2

Python Programming

1. Change directory:
   cd/home/pi/IDUINO_SuperKit_Python_code_for_RaspberryPi/
2. Run:
   sudo python 13_lcd1602.py

You should see two lines of characters displayed on the LCD1602.

Programming

C Programming

```
#include <stdio.h>
#include <stdlib.h>
#include <wiringPi.h>
#include <lcd.h>
const unsigned char Buf[] = "---IDUINO---";
const unsigned char myBuf[] = " IDUINO.com";
int main(void)
```

```
{
int fd;
int i;
if (wiringPiSetup() == -1){
exit(1);
}
fd      =      lcdInit(2,16,4,      2,3,      6,5,4,1,0,0,0,0);      //see
/usr/local/include/lcd.h
printf("%d", fd);
if (fd == -1){
printf("lcdInit 1 failed\n") ;
return 1;
}
sleep(1);
lcdClear(fd);
lcdPosition(fd, 0, 0);
lcdPuts(fd, "Welcom To--->");
lcdPosition(fd, 0, 1);
lcdPuts(fd, " IDUINO.com");
sleep(1);
lcdClear(fd);
while(1){
for(i=0;i<sizeof(Buf)-1;i++){
lcdPosition(fd, i, 1);
lcdPutchar(fd, *(Buf+i));
delay(200);
}
lcdPosition(fd, 0, 1);
lcdClear(fd);
sleep(0.5);
for(i=0; i<16; i++){
lcdPosition(fd, i, 0);
lcdPutchar(fd, *(myBuf+i));
delay(100);
}
}
return 0;
}
```

Python Programming

```
#!/usr/bin/env python
#
# based on code from lrvick and LiquidCrystal
# lrvic - https://github.com/lrvick/raspi-hd44780/blob/master/hd44780.py
#                              LiquidCrystal                                -
https://github.com/arduino/Arduino/blob/master/libraries/LiquidCrystal/

LiquidCrystal.cpp
#
from time import sleep
class Adafruit_CharLCD:
# commands
LCD_CLEARDISPLAY = 0x01
LCD_RETURNHOME = 0x02
LCD_ENTRYMODESET = 0x04
LCD_DISPLAYCONTROL = 0x08
LCD_CURSORSHIFT = 0x10
LCD_FUNCTIONSET = 0x20
LCD_SETCGRAMADDR = 0x40
LCD_SETDDRAMADDR = 0x80
# flags for display entry mode
LCD_ENTRYRIGHT = 0x00
LCD_ENTRYLEFT = 0x02
LCD_ENTRYSHIFTINCREMENT = 0x01
LCD_ENTRYSHIFTDECREMENT = 0x00
# flags for display on/off control
LCD_DISPLAYON = 0x04
LCD_DISPLAYOFF = 0x00
LCD_CURSORON = 0x02
LCD_CURSOROFF = 0x00
LCD_BLINKON = 0x01
LCD_BLINKOFF = 0x00
# flags for display/cursor shift
LCD_DISPLAYMOVE = 0x08
LCD_CURSORMOVE = 0x00
# flags for display/cursor shift
LCD_DISPLAYMOVE = 0x08
LCD_CURSORMOVE = 0x00
LCD_MOVERIGHT = 0x04
LCD_MOVELEFT = 0x00
```

```
# flags for function set
LCD_8BITMODE = 0x10
LCD_4BITMODE = 0x00
LCD_2LINE = 0x08
LCD_1LINE = 0x00
LCD_5x10DOTS = 0x04
LCD_5x8DOTS = 0x00

def __init__(self, pin_rs=27, pin_e=22, pins_db=[25, 24, 23, 18], GPIO =
None):
# Emulate the old behavior of using RPi.GPIO if we haven't been given
# an explicit GPIO interface to use
if not GPIO:
import RPi.GPIO as GPIO
self.GPIO = GPIO
self.pin_rs = pin_rs
self.pin_e = pin_e
self.pins_db = pins_db
self.GPIO.setmode(GPIO.BCM)
self.GPIO.setup(self.pin_e, GPIO.OUT)
self.GPIO.setup(self.pin_rs, GPIO.OUT)
for pin in self.pins_db:
self.GPIO.setup(pin, GPIO.OUT)
self.write4bits(0x33) # initialization
self.write4bits(0x32) # initialization
self.write4bits(0x28) # 2 line 5x7 matrix
self.write4bits(0x0C) # turn cursor off 0x0E to enable cursor
self.write4bits(0x06) # shift cursor right
self.displaycontrol   =   self.LCD_DISPLAYON   |   self.LCD_CURSOROFF   |
self.LCD_BLINKOFF
self.displayfunction    =    self.LCD_4BITMODE    |    self.LCD_1LINE    |
self.LCD_5x8DOTS
self.displayfunction |= self.LCD_2LINE
""" Initialize to default text direction (for romance languages) """
self.displaymode = self.LCD_ENTRYLEFT | self.LCD_ENTRYSHIFTDECREMENT
self.write4bits(self.LCD_ENTRYMODESET | self.displaymode) # set the entry
mode
self.clear()

def begin(self, cols, lines):
if (lines > 1):
self.numlines = lines
self.displayfunction |= self.LCD_2LINE
self.currline = 0
```

```
def home(self):
self.write4bits(self.LCD_RETURNHOME) # set cursor position to zero
self.delayMicroseconds(3000) # this command takes a long time!
def clear(self):
self.write4bits(self.LCD_CLEARDISPLAY) # command to clear display
self.delayMicroseconds(3000)  #  3000  microsecond  sleep,  clearing  the
display takes a long time

def setCursor(self, col, row):
self.row_offsets = [ 0x00, 0x40, 0x14, 0x54 ]
if ( row > self.numlines ):
row = self.numlines - 1 # we count rows starting w/0
self.write4bits(self.LCD_SETDDRAMADDR | (col + self.row_offsets[row]))

def noDisplay(self):
""" Turn the display off (quickly) """
self.displaycontrol &= ~self.LCD_DISPLAYON
self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

def display(self):
""" Turn the display on (quickly) """
self.displaycontrol |= self.LCD_DISPLAYON
self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)
def noCursor(self):
""" Turns the underline cursor on/off """
self.displaycontrol &= ~self.LCD_CURSORON
self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

def cursor(self):
""" Cursor On """
self.displaycontrol |= self.LCD_CURSORON
self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

def noBlink(self):
""" Turn on and off the blinking cursor """
self.displaycontrol &= ~self.LCD_BLINKON
self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)

def noBlink(self):
""" Turn on and off the blinking cursor """
self.displaycontrol &= ~self.LCD_BLINKON
self.write4bits(self.LCD_DISPLAYCONTROL | self.displaycontrol)
```

```
def DisplayLeft(self):
""" These commands scroll the display without changing the RAM """
self.write4bits(self.LCD_CURSORSHIFT    |    self.LCD_DISPLAYMOVE    |
self.LCD_MOVELEFT)

def scrollDisplayRight(self):
""" These commands scroll the display without changing the RAM """
self.write4bits(self.LCD_CURSORSHIFT    |    self.LCD_DISPLAYMOVE    |
self.LCD_MOVERIGHT);

def leftToRight(self):
""" This is for text that flows Left to Right """
self.displaymode |= self.LCD_ENTRYLEFT
self.write4bits(self.LCD_ENTRYMODESET | self.displaymode);

def rightToLeft(self):
""" This is for text that flows Right to Left """
self.displaymode &= ~self.LCD_ENTRYLEFT
self.write4bits(self.LCD_ENTRYMODESET | self.displaymode)

def autoscroll(self):
""" This will 'right justify' text from the cursor """
self.displaymode |= self.LCD_ENTRYSHIFTINCREMENT
self.write4bits(self.LCD_ENTRYMODESET | self.displaymode)

def noAutoscroll(self):
""" This will 'left justify' text from the cursor """
self.displaymode &= ~self.LCD_ENTRYSHIFTINCREMENT
self.write4bits(self.LCD_ENTRYMODESET | self.displaymode)

def write4bits(self, bits, char_mode=False):
""" Send command to LCD """
self.delayMicroseconds(1000) # 1000 microsecond sleep
bits=bin(bits)[2:].zfill(8)
self.GPIO.output(self.pin_rs, char_mode)
for pin in self.pins_db:
self.GPIO.output(pin, False)
for i in range(4):
if bits[i] == "1":
self.GPIO.output(self.pins_db[::-1][i], True)
self.pulseEnable()

for pin in self.pins_db:
self.GPIO.output(pin, False)
for i in range(4,8):
if bits[i] == "1":
self.GPIO.output(self.pins_db[::-1][i-4], True)
```

```
self.pulseEnable()

def delayMicroseconds(self, microseconds):
seconds = microseconds / float(1000000) # divide microseconds by 1 million
for seconds
sleep(seconds)

def pulseEnable(self):
self.GPIO.output(self.pin_e, False)
self.delayMicroseconds(1) # 1 microsecond pause - enable pulse must be >
450ns
self.GPIO.output(self.pin_e, True)
self.delayMicroseconds(1) # 1 microsecond pause - enable pulse must be >
450ns
self.GPIO.output(self.pin_e, False)
self.delayMicroseconds(1) # commands need > 37us to settle
def message(self, text):
""" Send string to LCD. Newline wraps to second line"""
for char in text:
if char == '\n':
self.write4bits(0xC0) # next line
else:
self.write4bits(ord(char),True)
def loop():
lcd = Adafruit_CharLCD()
while True:
lcd.clear()
lcd.message(" LCD 1602 Test \n123456789ABCDEF")
sleep(2)
lcd.clear()
lcd.message(" IDUINO \nHello World ! :)")
sleep(2)
lcd.clear()
lcd.message("Welcom to --->\n IDUINO.com")
sleep(2)
lcd.clear()
lcd.message("Welcom to --->\n IDUINO.com")
sleep(2)
if __name__ == '__main__':
loop()
```

Use this device with original accessories only. Velleman nv cannot be held responsible in the event of damage or injury resulting from (incorrect) use of this device. For more info concerning this product and the latest version of this manual, please visit our website www.velleman.eu. The information in this manual is subject to change without prior notice.

Velleman® Service and Quality Warranty

Since its foundation in 1972, Velleman® acquired extensive experience in the electronics world and currently distributes its products in over 85 countries.

All our products fulfil strict quality requirements and legal stipulations in the EU. In order to ensure the quality, our products regularly go through an extra quality check, both by an internal quality department and by specialized external organisations. If, all precautionary measures notwithstanding, problems should occur, please make appeal to our warranty (see guarantee conditions).

General Warranty Conditions Concerning Consumer Products (for EU):

• All consumer products are subject to a 24-month warranty on production flaws and defective material as from the original date of purchase.

• Velleman® can decide to replace an article with an equivalent article, or to refund the retail value totally or partially when the complaint is valid and a free repair or replacement of the article is impossible, or if the expenses are out of proportion.

You will be delivered a replacing article or a refund at the value of 100% of the purchase price in case of a flaw occurred in the first year after the date of purchase and delivery, or a replacing article at 50% of the purchase price or a refund at the value of 50% of the retail value in case of a flaw occurred in the second year after the date of purchase and delivery.

• Not covered by warranty:

- all direct or indirect damage caused after delivery to the article (e.g. by oxidation, shocks, falls, dust, dirt, humidity...), and by the article, as well as its contents (e.g. data loss), compensation for loss of profits;

- consumable goods, parts or accessories that are subject to an aging process during normal use, such as batteries (rechargeable, non-rechargeable, built-in or replaceable), lamps, rubber parts, drive belts... (unlimited list);

- flaws resulting from fire, water damage, lightning, accident, natural disaster, etc.…;

- flaws caused deliberately, negligently or resulting from improper handling, negligent maintenance, abusive use or use contrary to the manufacturer's instructions;

- damage caused by a commercial, professional or collective use of the article (the warranty validity will be reduced to six (6) months when the article is used professionally);

- damage resulting from an inappropriate packing and shipping of the article;

- all damage caused by modification, repair or alteration performed by a third party without written permission by Velleman®.

• Articles to be repaired must be delivered to your Velleman® dealer, solidly packed (preferably in the original packaging), and be completed with the original receipt of purchase and a clear flaw description.

• Hint: In order to save on cost and time, please reread the manual and check if the flaw is caused by obvious causes prior to presenting the article for repair. Note that returning a non-defective article can also involve handling costs.

• Repairs occurring after warranty expiration are subject to shipping costs.

• The above conditions are without prejudice to all commercial warranties.

The above enumeration is subject to modification according to the article (see article's manual).

Made in PRC
Imported by Velleman nv
Legen Heirweg 33, 9890 Gavere, Belgium
www.velleman.eu