```
61      }
62  }
63  int main(void){
64      int i;
65      printf("Program is starting ...\n");
66      wiringPiSetup();
67      if(detectI2C(0x27)){
68          pcf8574_address = 0x27;
69      }else if(detectI2C(0x3F)){
70          pcf8574_address = 0x3F;
71      }else{
72          printf("No correct I2C address found, \n"
73          "Please use command 'i2cdetect -y 1' to check the I2C address! \n"
74          "Program Exit. \n");
75          return -1;
76      }
77      pcf8574Setup(BASE,pcf8574_address);//initialize PCF8574
78      for(i=0;i<8;i++){
79          pinMode(BASE+i,OUTPUT);      //set PCF8574 port to output mode
80      }
81      digitalWrite(LED,HIGH);      //turn on LCD backlight
82      digitalWrite(RW,LOW);        //allow writing to LCD
83      lcdhd = lcdInit(2,16,4,RS,EN,D4,D5,D6,D7,0,0,0,0);// initialize LCD and return  “handle”
84  used to handle LCD
85      if(lcdhd == -1){
86          printf("lcdInit failed !");
87          return 1;
88      }
89      while(1){
90          printCPUTemperature();//print CPU temperature
91          printDataTime();        // print system time
92          delay(1000);
93      }
94      return 0;
95  }
```

From the code, we can see that the PCF8591 and the PCF8574 have many similarities in using the I2C interface to expand the GPIO RPI.

First, define the I2C address of the PCF8574 and the Extension of the GPIO pin, which is connected to the GPIO pin of the LCD1602. LCD1602 has two different i2c addresses. Set 0x27 as default.

```
int pcf8574_address = 0x27;        // PCF8574T:0x27, PCF8574AT:0x3F
#define BASE 64         // BASE any number above 64
//Define the output pins of the PCF8574, which are directly connected to the LCD1602 pin.
#define RS      BASE+0
#define RW      BASE+1
```

```
#define EN      BASE+2
#define LED     BASE+3
#define D4      BASE+4
#define D5      BASE+5
#define D6      BASE+6
#define D7      BASE+7
```

Then, in main function, initialize the PCF8574, set all the pins to output mode, and turn ON the LCD1602 backlight (without the backlight the Display is difficult to read).

```
    pcf8574Setup(BASE, pcf8574_address);// initialize PCF8574
    for(i=0;i<8;i++){
        pinMode(BASE+i,OUTPUT);      // set PCF8574 port to output mode
    }
    digitalWrite(LED,HIGH);      // turn on LCD backlight
```

Then use lcdInit() to initialize LCD1602 and set the RW pin of LCD1602 to 0 (can be written) according to requirements of this function. The return value of the function called "Handle" is used to handle LCD1602".

```
        lcdhd = lcdInit(2, 16, 4, RS, EN, D4, D5, D6, D7, 0, 0, 0, 0);// initialize LCD and return
“handle” used to handle LCD
```

Details about lcdInit():

int lcdInit (int rows, int cols, int bits, int rs, int strb,
int d0, int d1, int d2, int d3, int d4, int d5, int d6, int d7) ;

This is the main initialization function and must be executd first before you use any other LCD functions. **Rows** and **cols** are the rows and columns of the Display (e.g. 2, 16 or 4, 20). **Bits** is the number of how wide the number of bits is on the interface (4 or 8). The **rs** and **strb** represent the pin numbers of the Display's RS pin and Strobe (E) pin. The parameters **d0** through **d7** are the pin numbers of the 8 data pins connected from the RPi to the display. Only the first 4 are used if you are running the display in 4-bit mode.

The return value is the 'handle' to be used for all subsequent calls to the lcd library when dealing with that LCD, or -1 to indicate a fault (usually incorrect parameter)

For more details about LCD Library, please refer to: https://projects.drogon.net/raspberry-pi/wiringpi/lcd-library/

In the next “while”, two subfunctions are called to display the RPi's CPU Temperature and the SystemTime. First look at subfunction printCPUTemperature(). The CPU temperature data is stored in the "/sys/class/thermal/thermal_zone0/temp" file. We need to read the contents of this file, which converts it to temperature value stored in variable CPU_temp and uses lcdPrintf() to display it on LCD.

```
void printCPUTemperature(){//subfunction used to print CPU temperature

    FILE *fp;
    char str_temp[15];
    float CPU_temp;
    // CPU temperature data is stored in this directory.
    fp=fopen("/sys/class/thermal/thermal_zone0/temp","r");
    fgets(str_temp,15,fp);       // read file temp
    CPU_temp = atof(str_temp)/1000.0;    // convert to Celsius degrees
    printf("CPU's temperature : %.2f \n",CPU_temp);
```

```
    lcdPosition(lcdhd,0,0);     // set the LCD cursor position to (0,0)
    lcdPrintf(lcdhd,"CPU:%.2fC",CPU_temp);// Display CPU temperature on LCD
    fclose(fp);
}
```

Details about lcdPosition() and lcdPrintf():

lcdPosition (int handle, int x, int y);
Set the position of the cursor for subsequent text entry.
lcdPutchar (int handle, uint8_t data) **lcdPuts (int handle, char *string)** **lcdPrintf (int handle, char *message, …)**
These output a single ASCII character, a string or a formatted string using the usual print formatting commands to display individual characters (it is how you are able to see characters on your computer monitor).

Next is subfunction printDataTime() used to display System Time. First, it gets the Standard Time and stores it into variable Rawtime, and then converts it to the Local Time and stores it into timeinfo, and finally displays the Time information on the LCD1602 Display.

```
void printDataTime(){//used to print system time
    time_t rawtime;
    struct tm *timeinfo;
    time(&rawtime);// get system time
    timeinfo = localtime(&rawtime);// convert to local time
    printf("%s \n",asctime(timeinfo));
    lcdPosition(lcdhd,0,1);// set the LCD cursor position to (0,1)
    lcdPrintf(lcdhd,"Time:%d:%d:%d",timeinfo->tm_hour,timeinfo->tm_min,timeinfo->tm_sec);
//Display system time on LCD
}
```

Python Code 20.1.1 I2CLCD1602

If you did not configure I2C and install Smbus, please refer to Chapter 7. If you did, continue.

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 20.1.1_ I2CLCD1602 directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/20.1.1_I2CLCD1602
```

2. Use Python command to execute Python code "I2CLCD1602.py".

```
python I2CLCD1602.py
```

After the program is executed, the LCD1602 Screen will display your RPi's CPU Temperature and System Time.

NOTE: After the program is executed, if you cannot see anything on the display or the display is not clear, try rotating the white knob on back of LCD1602 slowly, which adjusts the contrast, until the screen can display the Time and Temperature clearly.

The following is the program code:

```
from PCF8574 import PCF8574_GPIO
from Adafruit_LCD1602 import Adafruit_CharLCD

from time import sleep, strftime
from datetime import datetime

def get_cpu_temp():     # get CPU temperature and store it into file
"/sys/class/thermal/thermal_zone0/temp"
    tmp = open('/sys/class/thermal/thermal_zone0/temp')
    cpu = tmp.read()
    tmp.close()
    return '{:.2f}'.format( float(cpu)/1000 ) + ' C'

def get_time_now():     # get system time
    return datetime.now().strftime('    %H:%M:%S')

def loop():
    mcp.output(3,1)     # turn on LCD backlight
    lcd.begin(16,2)     # set number of LCD lines and columns
    while(True):
        #lcd.clear()
```

```
22          lcd.setCursor(0,0)  # set cursor position
23          lcd.message( 'CPU: ' + get_cpu_temp()+'\n' )# display CPU temperature
24          lcd.message( get_time_now() )   # display the time
25          sleep(1)
26
27  def destroy():
28      lcd.clear()
29
30  PCF8574_address = 0x27  # I2C address of the PCF8574 chip.
31  PCF8574A_address = 0x3F  # I2C address of the PCF8574A chip.
32  # Create PCF8574 GPIO adapter.
33  try:
34      mcp = PCF8574_GPIO(PCF8574_address)
35  except:
36      try:
37          mcp = PCF8574_GPIO(PCF8574A_address)
38      except:
39          print ('I2C Address Error !')
40          exit(1)
41  # Create LCD, passing in MCP GPIO adapter.
42  lcd = Adafruit_CharLCD(pin_rs=0, pin_e=2, pins_db=[4,5,6,7], GPIO=mcp)
43
44  if __name__ == '__main__':
45      print ('Program is starting ... ')
46      try:
47          loop()
48      except KeyboardInterrupt:
49          destroy()
```

Two modules are used in the code, PCF8574.py and Adafruit_LCD1602.py. These two documents and the code files are stored in the same directory, and neither of them is dispensable. Please DO NOT DELETE THEM! PCF8574.py is used to provide I2C communication mode and operation method of some of the ports for the RPi and PCF8574 IC Chip. Adafruit module Adafruit_LCD1602.py is used to provide some functional operation method for the LCD1602 Display.

In the code, first get the object used to operate the PCF8574's port, then get the object used to operate the LCD1602.

```
address = 0x27  # I2C address of the PCF8574 chip.
# Create PCF8574 GPIO adapter.
mcp = PCF8574_GPIO(address)
# Create LCD, passing in MCP GPIO adapter.
lcd = Adafruit_CharLCD(pin_rs=0, pin_e=2, pins_db=[4,5,6,7], GPIO=mcp)
```

According to the circuit connection, port 3 of PCF8574 is connected to the positive pole of the LCD1602 Display's backlight. Then in the loop () function, use of mcp.output (3,1) to turn the LCD1602 Display's backlight ON and then set the number of LCD lines and columns.

```
def loop():
    mcp.output(3,1)     # turn on the LCD backlight
    lcd.begin(16,2)     # set number of LCD lines and columns
```

In the next while loop, set the cursor position, and display the CPU temperature and time.

```
    while(True):
        #lcd.clear()
        lcd.setCursor(0,0)  # set cursor position
        lcd.message( 'CPU: ' + get_cpu_temp()+'\n' )# display CPU temperature
        lcd.message( get_time_now() )   # display the time
        sleep(1)
```

CPU temperature is stored in file "/sys/class/thermal/thermal_zone0/temp". Open the file and read content of the file, and then convert it to Celsius degrees and return. Subfunction used to get CPU temperature is shown below:

```
def get_cpu_temp():     # get CPU temperature and store it into file
  "/sys/class/thermal/thermal_zone0/temp"
    tmp = open('/sys/class/thermal/thermal_zone0/temp')
    cpu = tmp.read()
    tmp.close()
    return '{:.2f}'.format( float(cpu)/1000 ) + ' C'
```

Subfunction used to get time:

```
def get_time_now():     # get the time
    return datetime.now().strftime('     %H:%M:%S')
```

Details about PCF8574.py and Adafruit_LCD1602.py:

Module PCF8574

This module provides two classes **PCF8574_I2C** and **PCF8574_GPIO**.
Class **PCF8574_I2C**: provides reading and writing method for PCF8574.
Class **PCF8574_GPIO**: provides a standardized set of GPIO functions.
More information can be viewed through opening PCF8574.py.
Adafruit_LCD1602 Module

Module Adafruit_LCD1602

This module provides the basic operation method of LCD1602, including class Adafruit_CharLCD. Some member functions are described as follows:
def begin(self, cols, lines): set the number of lines and columns of the screen.
def clear(self): clear the screen
def setCursor(self, col, row): set the cursor position
def message(self, text): display contents
More information can be viewed through opening Adafruit_CharLCD.py.

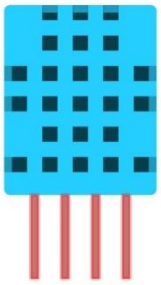
Chapter 21 Hygrothermograph DHT11

In this chapter, we will learn about a commonly used sensor called a Hygrothermograph DHT11.

Project 21.1 Hygrothermograph

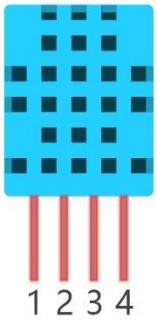
Hygrothermograph is an important tool in our lives to give us data on the temperature and humidity in our environment. In this project, we will use the RPi to read Temperature and Humidity data of the DHT11 Module.

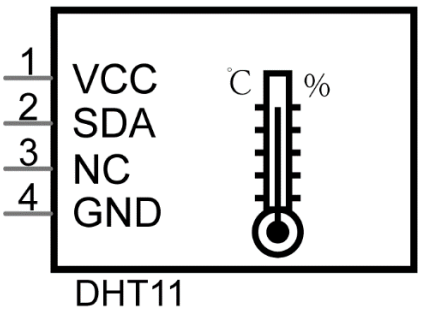
Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Ribbon Cable x1 Breadboard x1	DHT11 x1 	Resistor 10kΩ x1 
Jumper Wire x4 		

Component knowledge

The Temperature & Humidity Sensor DHT11 is a compound temperature & humidity sensor, and the output digital signal has been calibrated by its manufacturer.

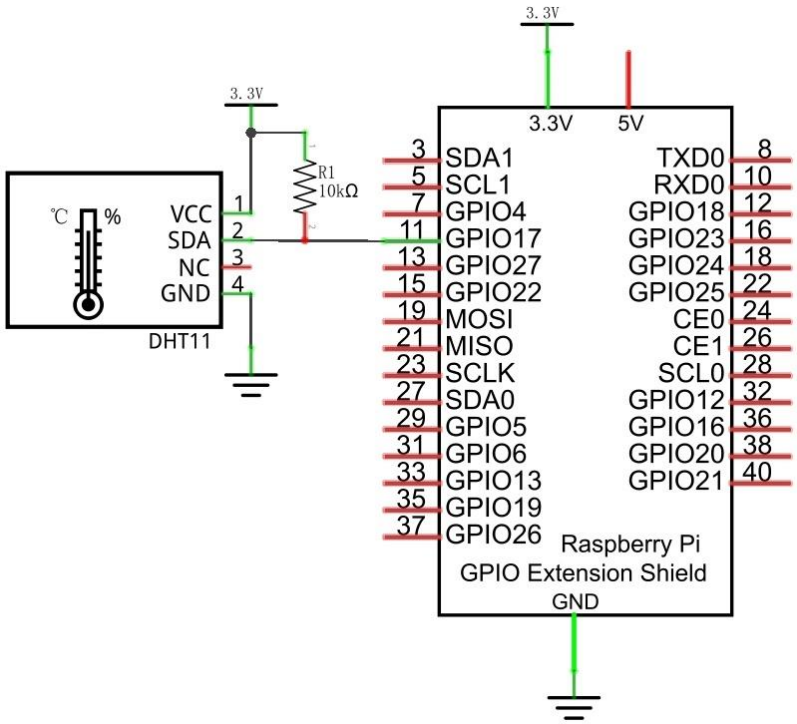




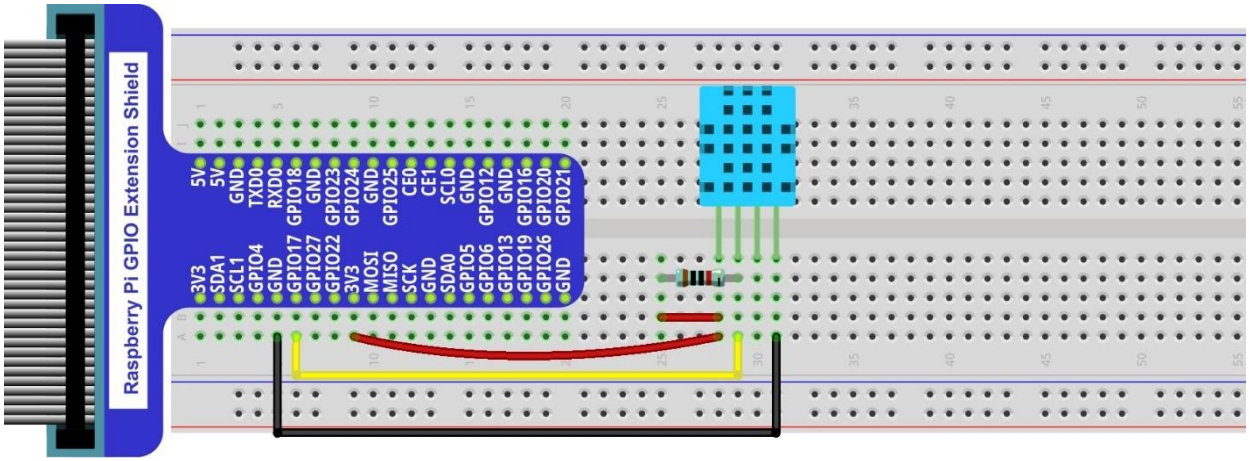
After being powered up, it will initialize in 1 second. Its operating voltage is within the range of 3.3V-5.5V. The SDA pin is a data pin, which is used to communicate with other devices.
The NC pin (Not Connected Pin) are a type of pin found on various integrated circuit packages. Those pins have no functional purpose to the outside circuit (but may have an unknown functionality during manufacture and test). Those pins **should not be connected** to any of the circuit connections.

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

The code is used to read the temperature and humidity data of DHT11, and display them.

C Code 21.1.1 DHT11

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 21.1.1_DHT11 directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/21.1.1_DHT11
```

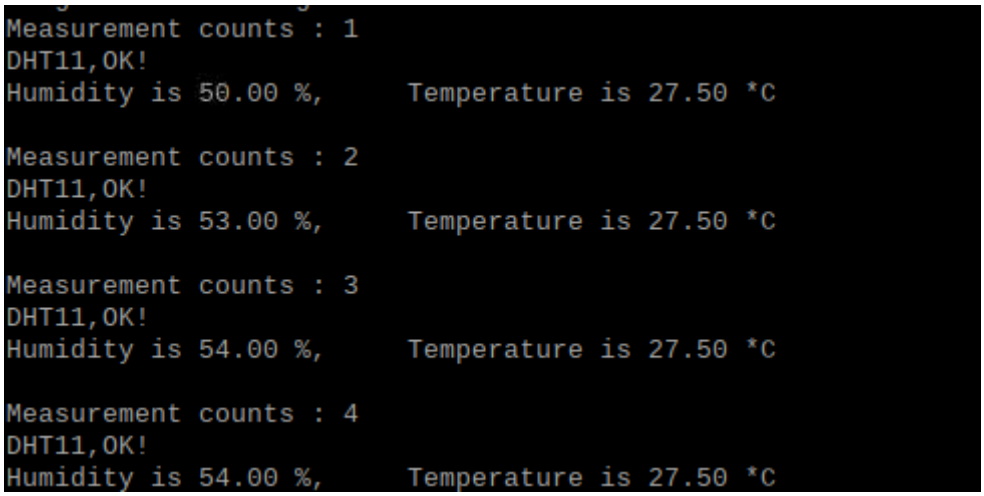
2. The code used in this project contains a custom header file. Use the following command to compile the code DHT11.cpp and DHT.cpp and generate executable file DHT11. The custom header file will be compiled at the same time.

```
gcc DHT.cpp DHT11.cpp -o DHT11 -lwiringPi
```

3. Run the generated file "DHT11".

```
sudo ./DHT11
```

After the program is executed, the Terminal window will display the current total number of read times, the read state, as well as temperature and humidity values as is shown below:



The following is the program code:

```
1  #include <wiringPi.h>
2  #include <stdio.h>
3  #include <stdint.h>
4  #include "DHT.hpp"
5
6  #define DHT11_Pin  0      //define the pin of sensor
7
8  int main(){
9      DHT dht;          //create a DHT class object
10     int chk, counts;      //chk:read the return value of sensor; sumCnt:times of reading
11 sensor
12
13     printf("Program is starting ...\n");
14
```

```
    while (1){
        counts++; //counting number of reading times
        printf("Measurement counts : %d \n", counts);
        for (int i = 0; i < 15; i++){
            chk = dht.readDHT11(DHT11_Pin); //read DHT11 and get a return value. Then
determine whether data read is normal according to the return value.
            if(chk == DHTLIB_OK){
                printf("DHT11,OK! \n");
                break;
            }
            delay(100);
        }
        printf("Humidity is %.2f %%, \t Temperature is %.2f *C\n\n",dht.humidity,
dht.temperature);
        delay(2000);
    }
    return 1;
}
```

In this project code, we use a custom library file "DHT.hpp". It is located in the same directory with the program files "DHT11.cpp" and "DHT.cpp", and methods for reading DHT sensor are provided in the library file. By using this library, we can easily read the DHT Sensor. First, we create a DHT class object in the code.

```
    DHT dht;
```

Then in the "while" loop, use chk = dht.**readDHT11** (DHT11_Pin) to read the DHT11, and determine whether the data read is normal according to the return value "chk". If the value is OK, end for loop and move on. Otherwise, try 15 times in total. Then use variable counts to record number of times to read.

```
while (1){
        counts++; //counting number of reading times
        printf("Measurement counts : %d \n", counts);
        for (int i = 0; i < 15; i++){
            chk = dht.readDHT11(DHT11_Pin); //read DHT11 and get a return value. Then
determine whether data read is normal according to the return value.
            if(chk == DHTLIB_OK){
                printf("DHT11,OK! \n");
                break;
            }
            delay(100);
        }
        printf("Humidity is %.2f %%, \t Temperature is %.2f *C\n\n",dht.humidity,
dht.temperature);
        delay(2000);
    }
```

Finally display the results:

```
    printf("Humidity is %.2f %%, \t Temperature is %.2f *C\n\n",dht.humidity,dht.temperature);
```

Library file "DHT.hpp" contains a DHT class and this public member function int **readDHT11** (int pin) is used to read sensor DHT11 and store the temperature and humidity data read to member variables double humidity and temperature. The implementation method of the function is included in the file "DHT.cpp".

```
#define _DHT_H_

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

////read return flag of sensor
#define DHTLIB_OK               0
#define DHTLIB_ERROR_CHECKSUM   -1
#define DHTLIB_ERROR_TIMEOUT    -2
#define DHTLIB_INVALID_VALUE    -999

#define DHTLIB_DHT11_WAKEUP     20
#define DHTLIB_DHT_WAKEUP       1

#define DHTLIB_TIMEOUT          100

class DHT{
    public:
        DHT();
        double humidity,temperature;    //use to store temperature and humidity data read
        int readDHT11Once(int pin);     //read DHT11
        int readDHT11(int pin);     //read DHT11
    private:
        uint8_t bits[5];    //Buffer to receiver data
        int readSensor(int pin,int wakeupDelay);    //

};
```

Python Code 21.1.1 DHT11

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 21.1.1_DHT11 directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/21.1.1_DHT11
```

2. Use Python command to execute code "DHT11.py".

```
python DHT11.py
```

After the program is executed, the Terminal window will display the current total number of read times, the read state, as well as temperature and humidity values as is shown below:

```
Measurement counts:  2
DHT11,OK!
Humidity : 53.00,        Temperature : 27.60

Measurement counts:  3
DHT11,OK!
Humidity : 53.00,        Temperature : 27.50

Measurement counts:  4
DHT11,OK!
Humidity : 53.00,        Temperature : 27.50

Measurement counts:  5
DHT11,OK!
Humidity : 52.00,        Temperature : 27.50
```

The following is the program code:

```
1  import RPi.GPIO as GPIO
2  import time
3  import Freenove_DHT as DHT
4  DHTPin = 11     #define the pin of DHT11
5
6  def loop():
7      dht = DHT.DHT(DHTPin)   #create a DHT class object
8      counts = 0 # Measurement counts
9      while(True):
10         counts += 1
11         print("Measurement counts: ", counts)
12         for i in range(0,15):
13             chk = dht.readDHT11()     #read DHT11 and get a return value. Then determine
14 whether data read is normal according to the return value.
15             if (chk is dht.DHTLIB_OK):      #read DHT11 and get a return value. Then determine
16 whether data read is normal according to the return value.
17                 print("DHT11,OK!")
18                 break
19             time.sleep(0.1)
20         print("Humidity : %.2f, \t Temperature : %.2f \n"%(dht.humidity,dht.temperature))
21         time.sleep(2)
22
```

```
23  if __name__ == '__main__':
24      print ('Program is starting ... ')
25      try:
26          loop()
27      except KeyboardInterrupt:
28          GPIO.cleanup()
29          exit()
```

In this project code, we use a module "**Freenove_DHT**.py", which provides the method of reading the DHT Sensor. It is located in the same directory with program files "**DHT11**.py". By using this library, we can easily read the DHT Sensor. First, we create a DHT class object in the code.

```
    dht = DHT.DHT(DHTPin)    #create a DHT class object
```

Then in the "while" loop, use chk = dht.**readDHT11** (DHT11Pin) to read the DHT11, and determine whether the data read is normal according to the return value "chk". Then use variable sumCnt to record the number of times read.

```
    while(True):
        counts += 1
        print("Measurement counts: ", counts)
        for i in range(0,15):
            chk = dht.readDHT11()      #read DHT11 and get a return value. Then determine
whether data read is normal according to the return value.
            if (chk is dht.DHTLIB_OK):      #read DHT11 and get a return value. Then determine
whether data read is normal according to the return value.
                print("DHT11,OK!")
                break
            time.sleep(0.1)
        print("Humidity : %.2f, \t Temperature : %.2f \n"%(dht.humidity,dht.temperature))
        time.sleep(2)
```

Finally display the results:

```
print("Humidity : %.2f, \t Temperature : %.2f \n"%(dht.humidity,dht.temperature))
```

Module "**Freenove_DHT.py**" contains a DHT class. The class function of the def **readDHT11** (pin) is used to read the DHT11 Sensor and store the temperature and humidity data read to member variables humidity and temperature.

Freenove_DHT Module

This is a Python module for reading the temperature and humidity data of the DHT Sensor. Partial functions and variables are described as follows:

Variable **humidity**: store humidity data read from sensor

Variable **temperature**: store temperature data read from sensor

def readDHT11 **(pin)**: read the temperature and humidity of sensor DHT11, and return values used to determine whether the data is normal.

Chapter 22 Matrix Keypad

Earlier we learned about a single Push Button Switch. In this chapter, we will learn about Matrix Keyboards, which integrates a number of Push Button Switches as Keys for the purposes of Input.

Project 22.1 Matrix Keypad

In this project, we will attempt to get every key code on the Matrix Keypad to work.

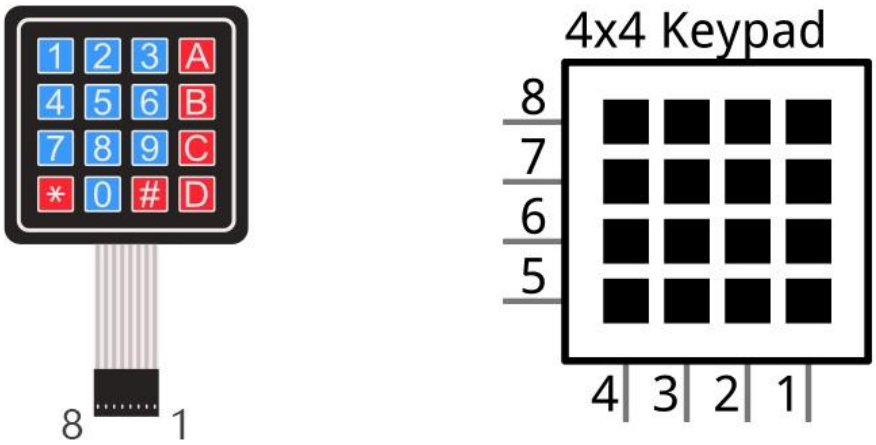
Component List

Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Wire x1 Breadboard x1	4x4 Matrix Keypad x1 
Jumper wire 	
Resistor 10kΩ x4 	

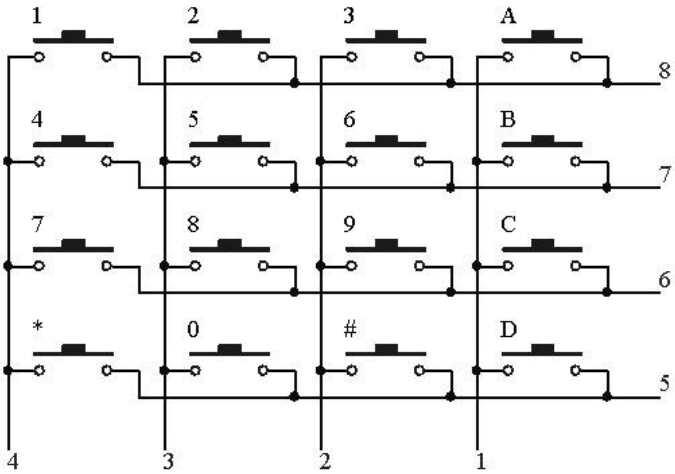
Component knowledge

4x4 Matrix Keypad

A Keypad Matrix is a device that integrates a number of keys in one package. As is shown below, a 4x4 Keypad Matrix integrates 16 keys (think of this as 16 Push Button Switches in one module):



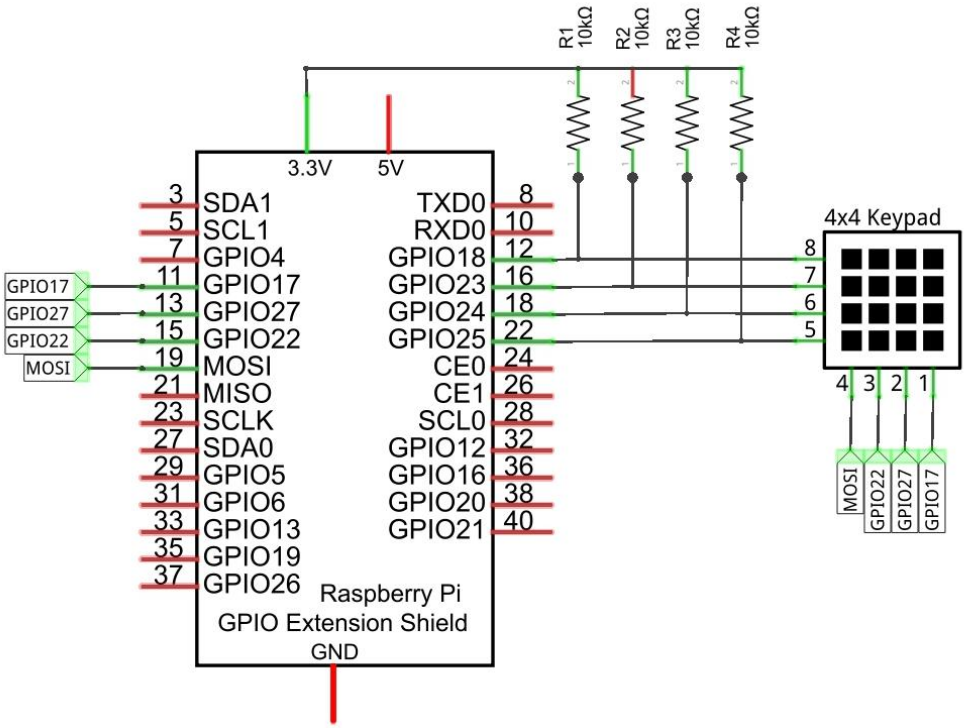
Similar to the integration of an LED Matrix, the 4x4 Keypad Matrix has each row of keys connected with one pin and this is the same for the columns. Such efficient connections reduce the number of processor ports required. The internal circuit of the Keypad Matrix is shown below.



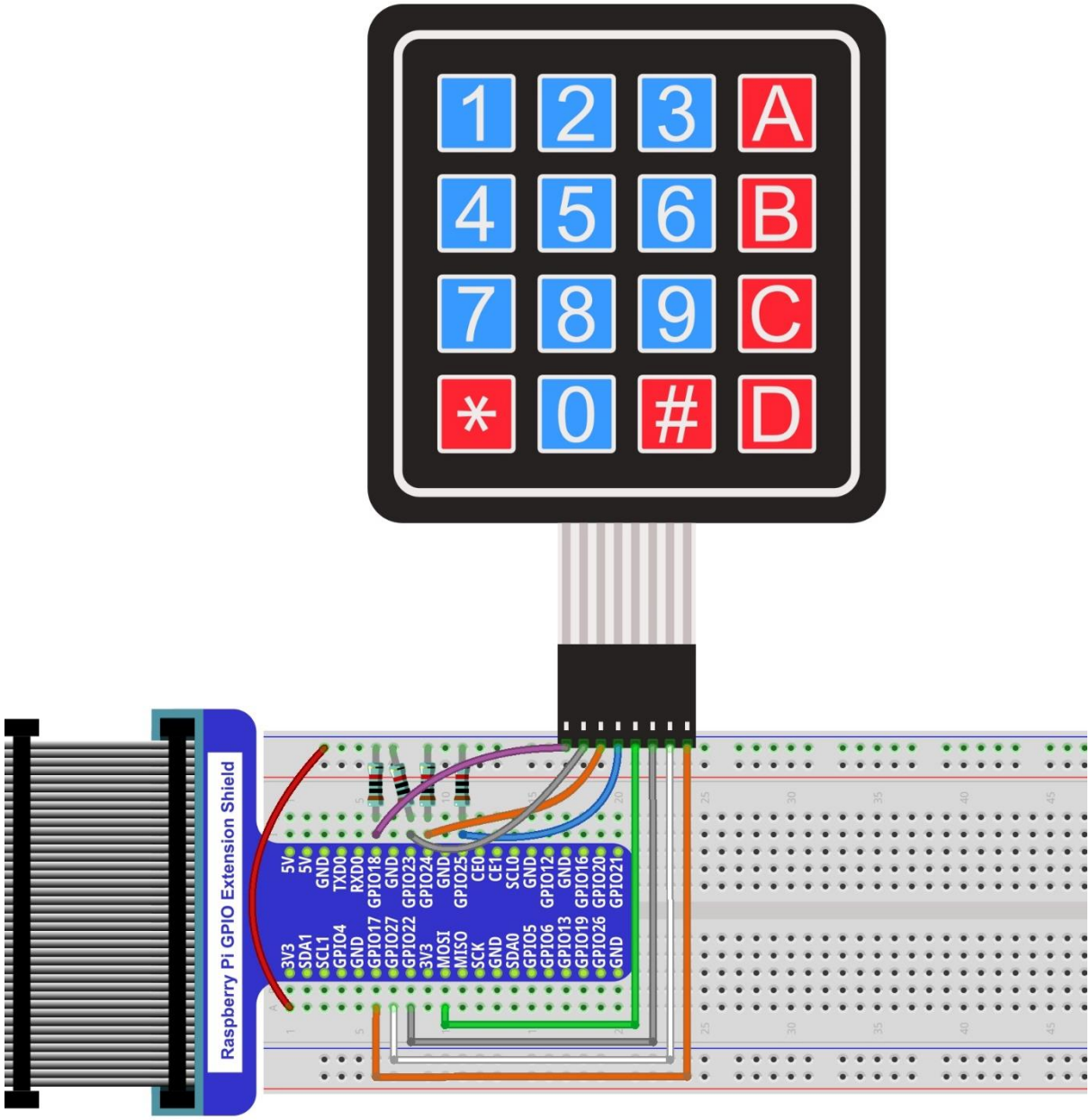
The method of usage is similar to the Matrix LED, by using a row or column scanning method to detect the state of each key's position by column and row. Take column scanning method as an example, send low level to the first 1 column (Pin1), detect level state of row 5, 6, 7, 8 to judge whether the key A, B, C, D are pressed. Then send low level to column 2, 3, 4 in turn to detect whether other keys are pressed. Therefore, you can get the state of all of the keys.

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

This code is used to obtain all key codes of the 4x4 Matrix Keypad, when one of the keys is pressed, the key code will be displayed in the terminal window.

C Code 22.1.1 MatrixKeypad

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 22.1.1_MatrixKeypad directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/22.1.1_MatrixKeypad
```

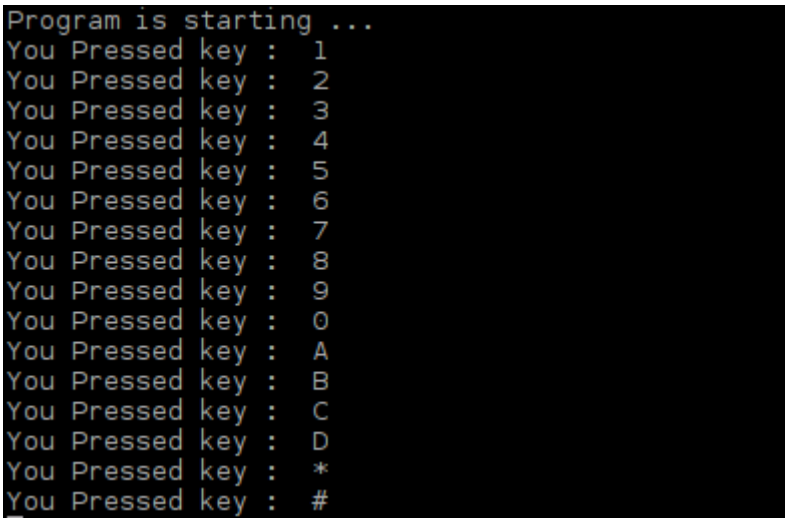
2. Code of this project contains a custom header file. Use the following command to compile the code MatrixKeypad.cpp, Keypad.cpp and Key.cpp generate executable file MatrixKeypad. The custom header file will be compiled at the same time.

```
gcc MatrixKeypad.cpp Keypad.cpp Key.cpp -o MatrixKeypad -lwiringPi
```

3. Run the generated file "MatrixKeypad".

```
sudo ./MatrixKeypad
```

After the program is executed, pressing any key on the MatrixKeypad, will display the corresponding key code on the Terminal. As is shown below:



The following is the program code:

```
#include "Keypad.hpp"
#include <stdio.h>
const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
char keys[ROWS][COLS] = {  //key code
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {1, 4, 5, 6 }; //define the row pins for the keypad
byte colPins[COLS] = {12,3, 2, 0 }; //define the column pins for the keypad
```

```
13  //create Keypad object
14  Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
15
16  int main(){
17      printf("Program is starting ... \n");
18
19      wiringPiSetup();
20
21      char key = 0;
22      keypad.setDebounceTime(50);
23      while(1){
24          key = keypad.getKey();  //get the state of keys
25          if (key){       //if a key is pressed, print out its key code
26              printf("You Pressed key :  %c \n",key);
27          }
28      }
29      return 1;
30  }
```

In this project code, we use two custom library file "**Keypad**.hpp" and "**Key**.hpp". They are located in the same directory with program files "**MatrixKeypad**.cpp", "**Keypad**.cpp" and "**Key**.cpp". The Library Keypad is "transplanted" from the Arduino Library Keypad. This library file provides a method to read the Matrix Keyboard's input. By using this library, we can easily read the pressed keys of the Matrix Keyboard.

First, we define the information of the Matrix Keyboard used in this project: the number of rows and columns, code designation of each key and GPIO pin connected to each column and row. It is necessary to include the header file "**Keypad**.hpp".

```
#include "Keypad.hpp"
#include <stdio.h>
const byte ROWS = 4; //four rows
const byte COLS = 4; //four columns
char keys[ROWS][COLS] = {  //key code
  {'1','2','3','A'},
  {'4','5','6','B'},
  {'7','8','9','C'},
  {'*','0','#','D'}
};
byte rowPins[ROWS] = {1, 4, 5, 6 }; //connect to the row pinouts of the keypad
byte colPins[COLS] = {12,3, 2, 0 }; //connect to the column pinouts of the keypad
```

Then, based on the above information, initiates a Keypad class object to operate the Matrix Keyboard.

```
Keypad keypad = Keypad( makeKeymap(keys), rowPins, colPins, ROWS, COLS );
```

Set the debounce time to 50ms, and this value can be set based on the actual characteristics of the keyboard's flexibly, with a default time of 10ms.

```
keypad.setDebounceTime(50);
```

In the "while" loop, use the function key= keypad.**getKey** () to read the keyboard constantly. If there is a key pressed, its key code will be stored in the variable "key", then be displayed.

```
    while(1){
        key = keypad.getKey();  //get the state of keys
        if (key){       // if a key is pressed, print out its key code
            printf("You Pressed key :  %c \n",key);
        }
    }
```

The Keypad Library used for the RPi is transplanted from the Arduino Keypad Library. And the source files can be obtained by visiting http://playground.arduino.cc/Code/Keypad. As for transplanted function library, the function and method of all classes, functions, variables, etc. are the same as the original library. Partial contents of the Keypad library are described below:

class Keypad

Keypad(char *userKeymap, byte *row, byte *col, byte numRows, byte numCols);

Constructor, the parameters are: key code of keyboard, row pin, column pin, the number of rows, the number of columns.

char getKey();

Get the key code of the pressed key. If no key is pressed, the return value is NULL.

void setDebounceTime(uint);

Set the debounce time. And the default time is 10ms.

void setHoldTime(uint);

Set the time when the key holds stable state after pressed.

bool isPressed(char keyChar);

Judge whether the key with code "keyChar" is pressed.

char waitForKey();

Wait for a key to be pressed, and return key code of the pressed key.

KeyState getState();

Get state of the keys.

bool keyStateChanged();

Judge whether there is a change of key state, then return True or False.

For More information about Keypad, please visit: http://playground.arduino.cc/Code/Keypad or through the opening file "Keypad.hpp".

Python Code 22.1.1 MatrixKeypad

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 22.1.1_MatrixKeypad directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/22.1.1_MatrixKeypad
```

2. Use Python command to execute code "MatrixKeypad.py".

```
python MatrixKeypad.py
```

After the program is executed, pressing any key on the MatrixKeypad, will display the corresponding key code on the Terminal. As is shown below:

```
Program is starting ...
You Pressed Key : 1
You Pressed Key : 2
You Pressed Key : 3
You Pressed Key : 4
You Pressed Key : 5
You Pressed Key : 6
You Pressed Key : 7
You Pressed Key : 8
You Pressed Key : 9
You Pressed Key : *
You Pressed Key : 0
You Pressed Key : #
You Pressed Key : A
You Pressed Key : B
You Pressed Key : C
You Pressed Key : D
```

The following is the program code:

```
import RPi.GPIO as GPIO
import Keypad        #import module Keypad
ROWS = 4         # number of rows of the Keypad
COLS = 4         #number of columns of the Keypad
keys =  [   '1','2','3','A',     #key code
            '4','5','6','B',
            '7','8','9','C',
            '*','0','#','D'      ]
rowsPins = [12,16,18,22]         #connect to the row pinouts of the keypad
colsPins = [19,15,13,11]         #connect to the column pinouts of the keypad

def loop():
    keypad = Keypad.Keypad(keys,rowsPins,colsPins,ROWS,COLS)     #creat Keypad object
    keypad.setDebounceTime(50)       #set the debounce time
    while(True):
        key = keypad.getKey()        #obtain the state of keys
        if(key != keypad.NULL):      #if there is key pressed, print its key code.
            print ("You Pressed Key : %c "%(key))

if __name__ == '__main__':
    print ("Program is starting ... ")
```

```
22      try:
23          loop()
24      except KeyboardInterrupt:
25          GPIO.cleanup()
```

In this project code, we use two custom library files "**Keypad**.hpp" and "**Key**.hpp". They are located in the same directory with program files "**MatrixKeypad**.cpp", "**Keypad**.cpp" and "**Key**.cpp". The Library Keypad is "transplanted" from the Arduino Library Keypad. This library file provides a method to read the Matrix Keyboard's input. First, import the module Keypad. Then define the information of the matrix keyboard used in this project: the number of rows and columns, code of each key and GPIO pin connected to each column and each row.

```
import Keypad       #import module Keypad
ROWS = 4        #number of rows of the Keypad
COLS = 4        #number of columns of the Keypad
keys =  [   '1','2','3','A',     #key code
            '4','5','6','B',
            '7','8','9','C',
            '*','0','#','D'     ]
rowsPins = [12,16,18,22]        #connect to the row pinouts of the keypad
colsPins = [19,15,13,11]        #connect to the column pinouts of the keypad
```

Then, based on the above information, initiates a Keypad class object to operate the Matrix Keyboard.

```
keypad = Keypad.Keypad(keys,rowsPins,colsPins,ROWS,COLS)
```

Set the debounce time to 50ms, and this value can be set based on the actual characteristics of the keyboard's flexibly, with a default time of 10ms.

```
keypad.setDebounceTime(50)
```

In the "while" loop, use the function key= keypad.**getKey** () to read the keyboard constantly. If there is a key pressed, its key code will be stored in the variable "key", and then be displayed.

```
    while(True):
        key = keypad.getKey()       #get the state of keys
        if(key != keypad.NULL):     # if a key is pressed, print out its key code
            print ("You Pressed Key : %c "%(key))
```

The Keypad Library used for the RPi is “transplanted” from the Arduino Keypad Library. The source files is written by language C++ and translated into Python can be obtained by visiting http://playground.arduino.cc/Code/Keypad. As for the “transplanted” function library, the function and method of all classes, functions, variables, etc. are the same as the original library. Partial contents of the Keypad Library are described below:

class Keypad

```
def __init__(self,usrKeyMap,row_Pins,col_Pins,num_Rows,num_Cols):
```
Constructed function, the parameters are: key code of keyboard, row pin, column pin, the number of rows, the number of columns.

```
def getKey(self):
```
Get a pressed key. If no key is pressed, the return value is keypad NULL.

```
def setDebounceTime(self,ms):
```
Set the debounce time. And the default time is 10ms.

```
def setHoldTime(self,ms):
```
Set the time when the key holds stable state after pressed.

```
def isPressed(keyChar):
```
Judge whether the key with code "keyChar" is pressed.

```
def waitForKey():
```
Wait for a key to be pressed, and return key code of the pressed key.

```
def getState():
```
Get state of the keys.

```
def keyStateChanged():
```
Judge whether there is a change of key state, then return True or False.

For More information about Keypad, please visit: http://playground.arduino.cc/Code/Keypad or through the opening file "Keypad.py".

Chapter 23 Ultrasonic Ranging

In this chapter, we learn a module which use ultrasonic to measure distance, HC SR04.

Project 23.1 Ultrasonic Ranging

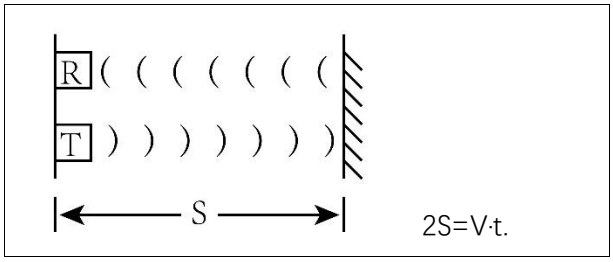
In this project, we use ultrasonic ranging module to measure distance, and print out the data in the terminal.

Component List

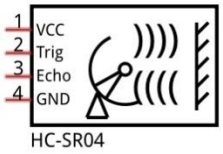
Raspberry Pi (with 40 GPIO) x1 GPIO Expansion Board & Ribbon Cable x1 Breadboard x1	HC SR04 x1 
Jumper Wire x4 	Resistor 1kΩ x1 

Component Knowledge

The Ultrasonic Ranging Module uses the principle that ultrasonic waves will be reflected when they encounter any obstacles. This is possible by counting the time interval between when the ultrasonic wave is transmitted to when the ultrasonic wave reflects back after encountering an obstacle. Time interval counting will end after an ultrasonic wave is received, and the time difference (delta) is the total time of the ultrasonic wave's journey from being transmitted to being received. Because the speed of sound in air is a constant, and is about v=340m/s, we can calculate the distance between the Ultrasonic Ranging Module and the obstacle: s=vt/2.



The HC-SR04 Ultrasonic Ranging Module integrates a both an ultrasonic transmitter and a receiver. The transmitter is used to convert electrical signals (electrical energy) into high frequency (beyond human hearing) sound waves (mechanical energy) and the function of the receiver is opposite of this. The picture and the diagram of the HC SR04 Ultrasonic Ranging Module are shown below:



Pin description:

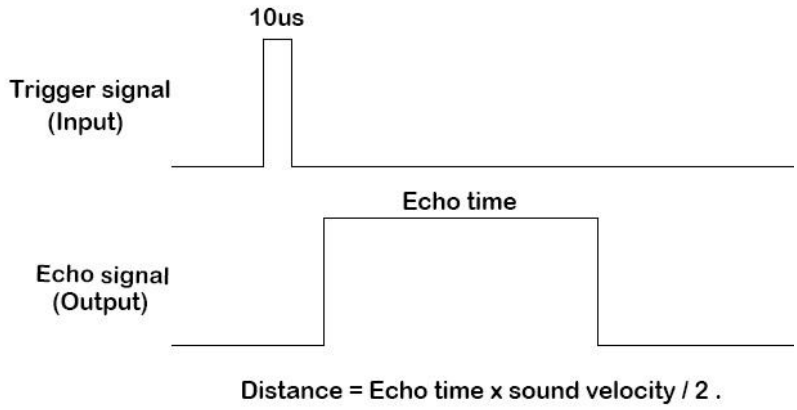
VCC	power supply pin
Trig	trigger pin
Echo	Echo pin
GND	GND

Technical specs:

Working voltage: 5V Working current: 12mA

Minimum measured distance: 2cm Maximum measured distance: 200cm

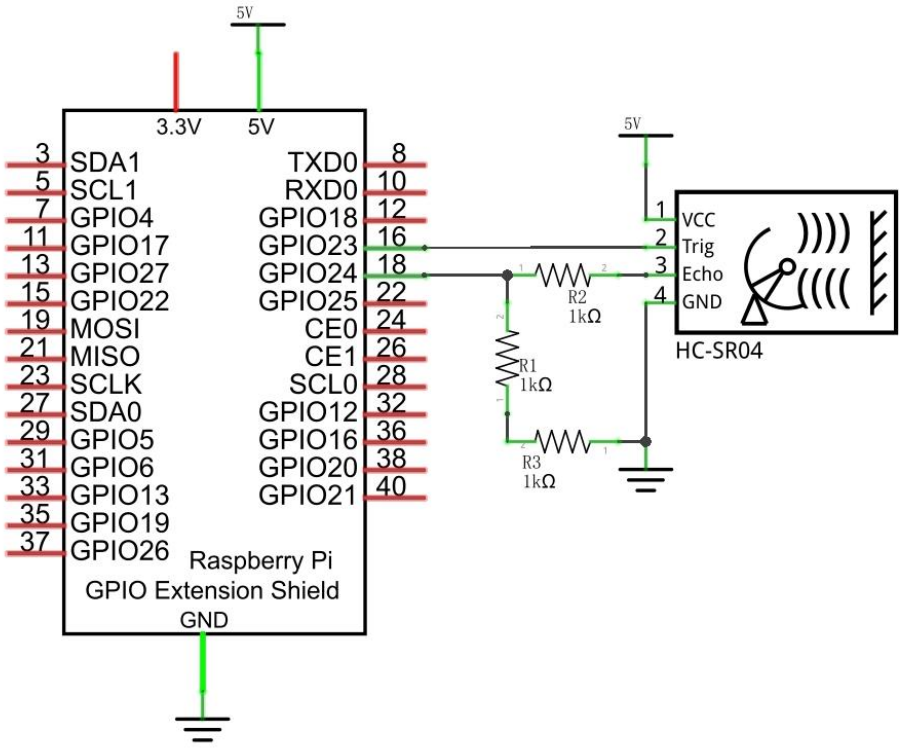
Instructions for Use: output a high-level pulse in Trig pin lasting for least 10uS, the module begins to transmit ultrasonic waves. At the same time, the Echo pin is pulled up. When the module receives the returned ultrasonic waves from encountering an obstacle, the Echo pin will be pulled down. The duration of high level in the Echo pin is the total time of the ultrasonic wave from transmitting to receiving, s=vt/2. This is done constantly.



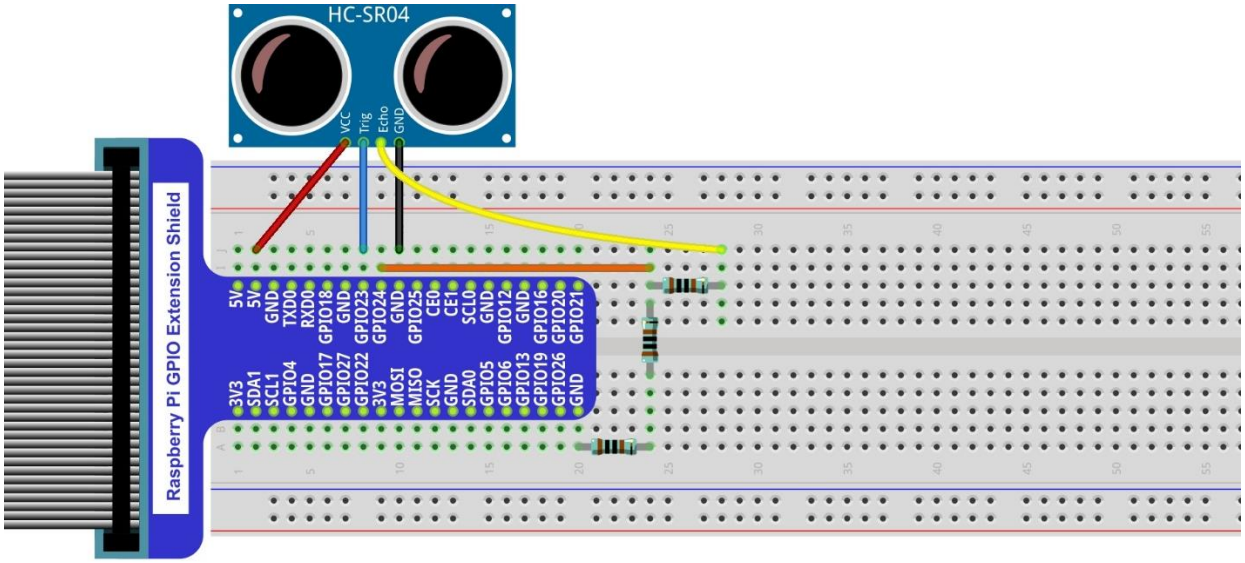
Circuit

Note that the voltage of ultrasonic module is 5V in this circuit.

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Code

C Code 23.1.1 UltrasonicRanging

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 23.1.1_UltrasonicRanging directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/23.1.1_UltrasonicRanging
```

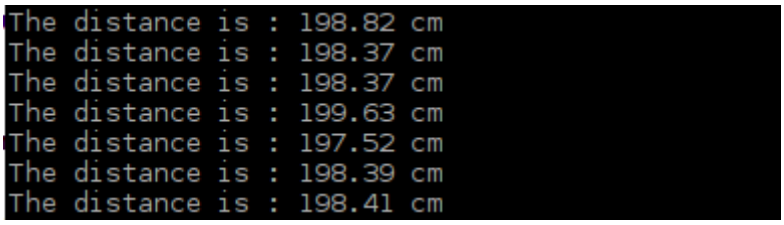
2. Use following command to compile "UltrasonicRanging.c" and generate executable file "UltrasonicRanging".

```
gcc UltrasonicRanging.c -o UltrasonicRanging -lwiringPi
```

3. Then run the generated file "UltrasonicRanging".

```
sudo ./UltrasonicRanging
```

After the program is executed, aim the Ultrasonic Ranging Module's detectors ("eyes") perpendicular to the surface of an object (try using your hand). The distance between the ultrasonic module and the object will be displayed in the terminal. As is shown below:



The following is the program code:

```
#include <wiringPi.h>
#include <stdio.h>
#include <sys/time.h>

#define trigPin 4
#define echoPin 5
#define MAX_DISTANCE 220        // define the maximum measured distance
#define timeOut MAX_DISTANCE*60 // calculate timeout according to the maximum measured
distance
//function pulseIn: obtain pulse time of a pin
int pulseIn(int pin, int level, int timeout);
float getSonar(){   //get the measurement result of ultrasonic module with unit: cm
    long pingTime;
    float distance;
    digitalWrite(trigPin,HIGH); //send 10us high level to trigPin
    delayMicroseconds(10);
    digitalWrite(trigPin,LOW);
    pingTime = pulseIn(echoPin,HIGH,timeOut);   //read plus time of echoPin
    distance = (float)pingTime * 340.0 / 2.0 / 10000.0; //calculate distance with sound speed
340m/s
    return distance;
```

```
}

int main(){
    printf("Program is starting ... \n");

    wiringPiSetup();

    float distance = 0;
    pinMode(trigPin,OUTPUT);
    pinMode(echoPin,INPUT);
    while(1){
        distance = getSonar();
        printf("The distance is : %.2f cm\n",distance);
        delay(1000);
    }
    return 1;
}
```

First, define the pins and the maximum measurement distance.

```
#define trigPin 4
#define echoPin 5
#define MAX_DISTANCE 220          //define the maximum measured distance
```

If the module does not return high level, we cannot wait for this forever, so we need to calculate the time period for the maximum distance, that is, time Out. **timeOut= 2*MAX_DISTANCE/100/340*1000000**. The result of the constant part in this formula is approximately 58.8.

```
#define timeOut MAX_DISTANCE*60
```

Subfunction **getSonar** () function is used to start the Ultrasonic Module to begin measurements and return the measured distance in cm units. In this function, first let trigPin send 10us high level to start the Ultrasonic Module. Then use **pulseIn** () to read the Ultrasonic Module and return the duration time of high level. Finally, the measured distance according to the time is calculated.

```
float getSonar(){   // get the measurement results of ultrasonic module, with unit: cm
    long pingTime;
    float distance;
    digitalWrite(trigPin,HIGH); //trigPin send 10us high level
    delayMicroseconds(10);
    digitalWrite(trigPin,LOW);
    pingTime = pulseIn(echoPin,HIGH,timeOut);   //read plus time of echoPin
    distance = (float)pingTime * 340.0 / 2.0 / 10000.0; // the sound speed is 340m/s, and
calculate distance
    return distance;
}
```

Lastly, in the while loop of main function, get the measurement distance and display it continually.

```
    while(1){
        distance = getSonar();
        printf("The distance is : %.2f cm\n",distance);
        delay(1000);
    }
```

About function **pulseIn**():

int pulseIn(int pin, int level, int timeout);

Return the length of the pulse (in microseconds) or 0 if no pulse is completed before the timeout (unsigned long).

Python Code 23.1.1 UltrasonicRanging

First, observe the project result, and then learn about the code in detail.

If you have any concerns, please contact us via: support@freenove.com

1. Use cd command to enter 23.1.1_UltrasonicRanging directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/23.1.1_UltrasonicRanging
```

2. Use Python command to execute code "UltrasonicRanging.py".

```
python UltrasonicRanging.py
```

After the program is executed, aim the Ultrasonic Ranging Module's detectors ("eyes") perpendicular to the surface of an object (try using your hand). The distance between the ultrasonic module and the object will be displayed in the terminal. As is shown below:

```
The distance is : 198.75 cm
The distance is : 199.22 cm
The distance is : 198.42 cm
The distance is : 198.74 cm
The distance is : 198.37 cm
The distance is : 198.47 cm
The distance is : 198.41 cm
```

The following is the program code:

```
1  import RPi.GPIO as GPIO
2  import time
3
4  trigPin = 16
5  echoPin = 18
6  MAX_DISTANCE = 220          #define the maximum measured distance(cm)
7  timeOut = MAX_DISTANCE*60   #calculate timeout(μs) according to the maximum measured
8  distance
9
10 def pulseIn(pin,level,timeOut): # function pulseIn: obtain pulse time of a pin
11     t0 = time.time()
12     while(GPIO.input(pin) != level):
13         if((time.time() - t0) > timeOut*0.000001):
14             return 0;
15     t0 = time.time()
16     while(GPIO.input(pin) == level):
17         if((time.time() - t0) > timeOut*0.000001):
18             return 0;
19     pulseTime = (time.time() - t0)*1000000
20     return pulseTime
21
22 def getSonar():     #get the measurement results of ultrasonic module,with unit: cm
23     GPIO.output(trigPin,GPIO.HIGH)      #make trigPin send 10us high level
24     time.sleep(0.00001)     #10us
25     GPIO.output(trigPin,GPIO.LOW)
26     pingTime = pulseIn(echoPin,GPIO.HIGH,timeOut)   #read plus time of echoPin
27
```

```
28      distance = pingTime * 340.0 / 2.0 / 10000.0     # the sound speed is 340m/s, and
29  calculate distance (cm)
30      return distance
31
32  def setup():
33      print ('Program is starting...')
34      GPIO.setmode(GPIO.BOARD)        #numbers GPIOs by physical location
35      GPIO.setup(trigPin, GPIO.OUT)   # set trigPin to output mode
36      GPIO.setup(echoPin, GPIO.IN)    # set echoPin to input mode
37
38  def loop():
39      while(True):
40          distance = getSonar()
41          print ("The distance is : %.2f cm"%(distance))
42          time.sleep(1)
43
44  if __name__ == '__main__':     #program start from here
45      setup()
46      try:
47          loop()
48      except KeyboardInterrupt:
            GPIO.cleanup()
```

First, define the pins and the maximum measurement distance.

```
trigPin = 16
echoPin = 18
MAX_DISTANCE = 220          # define the maximum measured distance 220cm
```

If the module does not return high level, we cannot wait for this forever, so we need to calculate the time period for the maximum distance (200cm). Then **timOut= 2*MAX_DISTANCE/100/340*1000000**. The result of the constant part in this formula is approximately 58.8.

```
timeOut = MAX_DISTANCE*60
```

Subfunction **getSonar** () function is used to start the Ultrasonic Module to begin measurements, and return the measured distance in cm units. In this function, first let trigPin send 10us high level to start the Ultrasonic Module. Then use **pulseIn** () to read the Ultrasonic Module and return the duration time of high level. Finally, the measured distance according to the time is calculated.

```
def getSonar():     #get the measurement results of ultrasonic module, with unit: cm
    GPIO.output(trigPin,GPIO.HIGH)      #make trigPin send 10us high level
    time.sleep(0.00001)     #10us
    GPIO.output(trigPin,GPIO.LOW)
    pingTime = pulseIn(echoPin,GPIO.HIGH,timeOut)   #read plus time of echoPin
    distance = pingTime * 340.0 / 2.0 / 10000.0     # the sound speed is 340m/s, and
calculate distance
    return distance
```

Finally, in the while loop of main function, get the measurement distance and display it continually.

```
    while(True):
        distance = getSonar()
        print ("The distance is : %.2f cm"%(distance))
        time.sleep(1)
```

About function `def pulseIn(pin,level,timeOut):`

def pulseIn(pin,level,timeOut):

Return the length of the pulse (in microseconds) or 0 if no pulse is completed before the timeout (unsigned long).

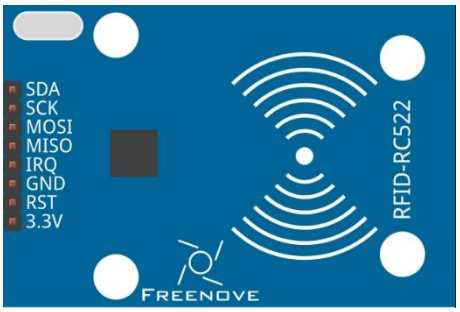
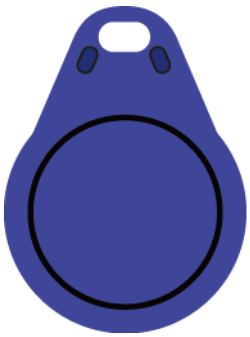
Chapter 24 RFID

In this chapter, we will learn how to use RFID.

Project 24.1 RFID

In this project, we will use RC522 RFID card reader to read and write the M1-S50 card.

Component List

Raspberry Pi 3B x1 GPIO Extension Board & Wire x1 Breadboard x1	RC522 module x1 
Jemper M/F x7 	
Mifare1 S50 Standard card x1 	Mifare1 S50 Non-standard card x1 

Component Knowledge

RFID

RFID (Radio Frequency Identification) is a form of wireless communication technology. A complete RFID system is generally composed of a transponder and a reader. Generally, the transponder may be known as a tag, and each tag has a unique code, which is attached to an object to identify the target object. The reader is a device that reads (or writes) information in the tag.

Products derived from RFID technology can be divided into three categories: passive RFID products, active RFID products and semi active RFID products, among which, Passive RFID products are the earliest, the most mature and most widely used products in the market. It can be seen everywhere in our daily life such as, the bus card, dining card, bank card, hotel access cards, etc., and all of them are classified as close-range contact recognition. The main operating frequency of Passive RFID products are: 125KHZ (low frequency), 13.56MHZ (high frequency), 433MHZ (ultrahigh frequency), 915MHZ (ultrahigh frequency). Active and semi active RFID products work at higher frequencies.

The RFID module we use is a passive RFID product with the operating frequency of 13.56MHz.

MFRC522

The MFRC522 is a highly integrated reader/writer IC for contactless communication at 13.56MHz.

The MFRC522's internal transmitter is able to drive a reader/writer antenna designed to communicate with ISO/IEC 14443 A/MIFARE cards and transponders without additional active circuitry. The receiver module provides a robust and efficient implementation for demodulating and decoding signals from ISO/IEC 14443 A/MIFARE compatible cards and transponders. The digital module manages the complete ISO/IEC 14443A framing and error detection (parity and CRC) functionality

This RFID Module uses MFRC522 as the control chip, and SPI (Peripheral Interface Serial) as the reserved interface.

Technical specs:

Operating Voltage	13-26mA(DC)\3.3V
Idle current	10-13mA(DC)\3.3V
Sleep current in the	<80uA
Peak current	<30mA
Operating frequency	13.56MHz
Supported card type	Mifare1 S50、Mifare1 S70、Mifare Ultralight、Mifare Pro、Mifare Desfire
Size	40mmX60mm
Operation temperature	20-80 degrees(Celsius)
Storage temperature	40-85 degrees (Celsius)
Operation humidity	5%-95%(Relative humidity)

Mifare1 S50 Card

Mifare S50 is often called Mifare Standard with the capacity of 1K bytes. And each card has a 4-bytes global unique identifier number (USN/UID), which can be rewritten 100 thousand times and read infinite times. Its storage period can last for 10 years.

The Mifare S50 capacity (1K byte) is divided into 16 sectors (Sector0-Sector15). Each sector contains 4 data block (Block0-Block3. 64 blocks of 16 sectors will be numbered according absolute address, from 0 to 63).

And each block contains 16 bytes (Byte0-Byte15), 64*16=1024. As is shown in the following table:

Sector No.	Block No.	Storage area	Block type	Absolute block No.
sector 0	block 0	vendor code	vendor block	0
	block 1		data block	1
	block 2		data block	2
	block 3	Password A-access control-password B	control block	3
sector 1	block 0		data block	4
	block 1		data block	5
	block 2		data block	6
	block 3	Password A-access control-password B	control block	7
……	……	……	……	
sector 15	block 0		data block	60
	block 1		data block	61
	block 2		data block	62
	block 3	Password A-access control-password B	control block	63

Each sector has a set of independent password and access control put in its last block, that is, Block 3, which is also known as sector trailer. Sector 0, block 0 (namely absolute address 0) of S50 is used to store the card serial number and vendor code, which has been solidified and can't be changed. Except the manufacturer and the control block, the rest of the cards are data blocks, which can be used to store data. Data block can be used for two kinds of applications:

(1) used as general data storage and can be operated for reading and writing data.

(2) used as data value, and can be operated for initializing, adding, subtracting and reading the value.

The sector trailer block in each sector is the control block, including a 6-byte password A, a 4-byte access control and a 6-byte password B. For example, the control block of a brand new card is as follows:

A0 A1 A2 A3 A4 A5	FF 07 80 69	B0 B1 B2 B3 B4 B5
password A	access control	password B

The default password of a brand new card is generally 0A1A2A3A4A5 for password A and B0B1B2B3B4B5 for password B, or both the password A and password B are 6 FF. Access control is used to set the access conditions for each block (including the control block itself) in a sector.

Blocks of S50 are divided into data blocks and control blocks. There are four operations, "read", "write", "add value", "subtract value (including transmission and storage)" for data blocks, and there are two operations, "read" and "write" for control blocks.

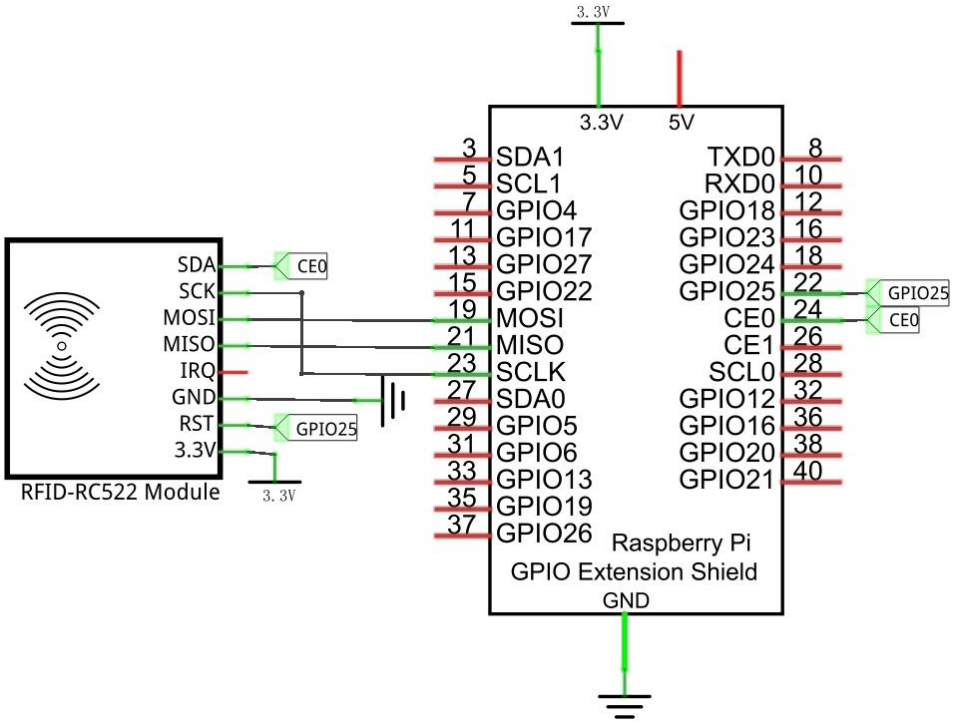
For more details about how to set data blocks and control blocks, please refer to Datasheet.

By default, after verifying password A or password B, we can do reading or writing operation to data blocks. And after verifying password A, we can do reading or writing operation to control blocks. But password A can never be read, so if you choose to verify password A but forget the password A, the block will never be able to read again. **It is highly recommended that beginners should not try to change the contents of control blocks.**

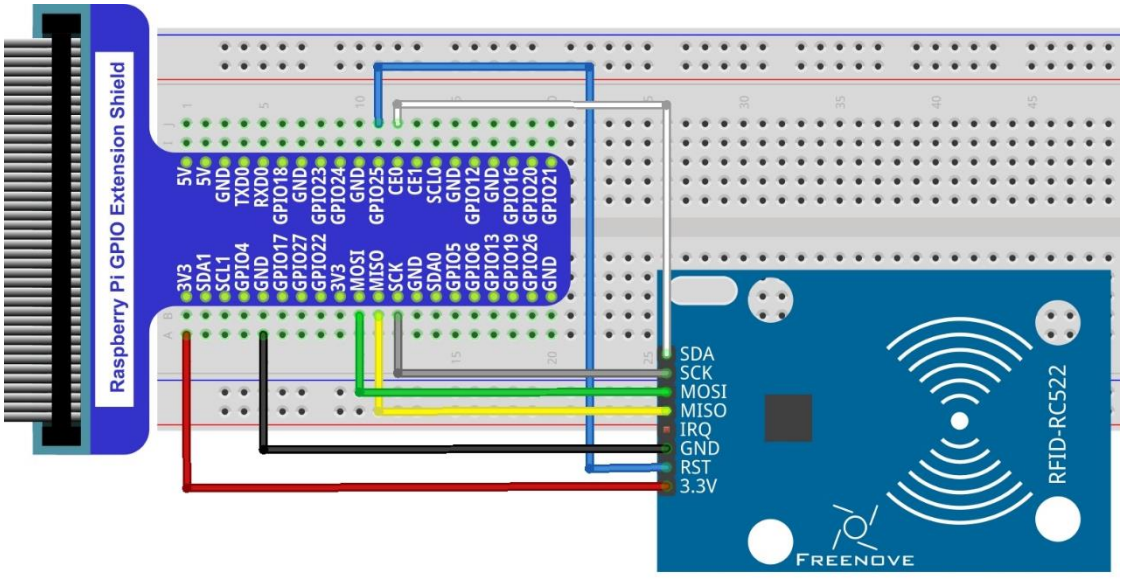
For Mifare1 S50 card equipped in Freenove RFID Kit, the default password A and B are both FFFFFFFFFFFF.

Circuit

Schematic diagram:



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Configure SPI

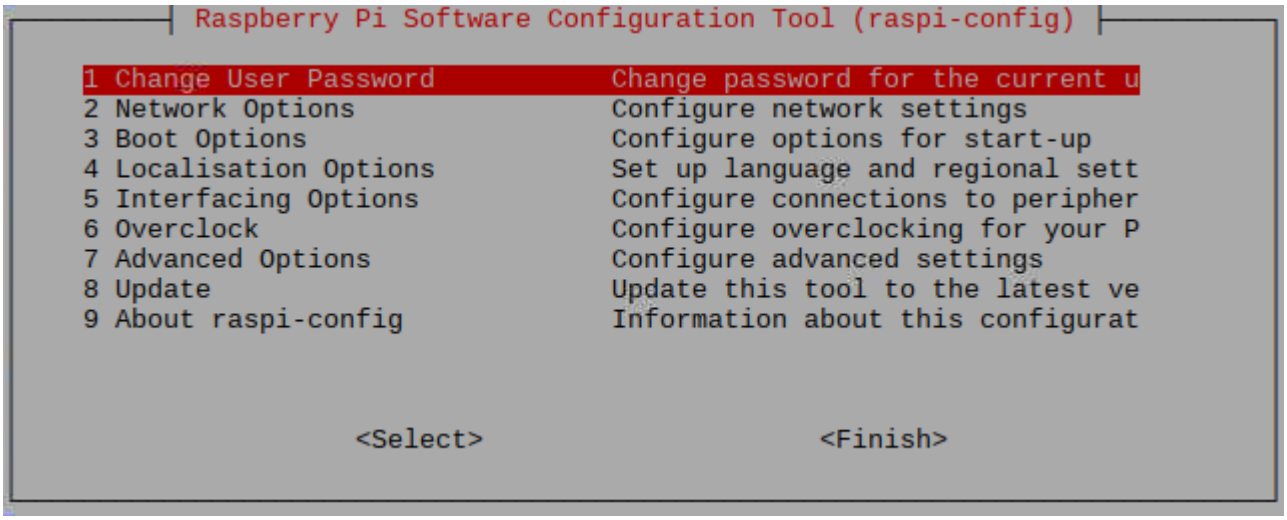
Enable SPI

The SPI interface of raspberry pi is closed by default. You need to open it manually. You can enable the SPI interface in the following way.

Type the following command in the terminal:

```
sudo raspi-config
```

Then open the following dialog box:



Choose “5 Interfacing Options”→“P4 SPI”→“Yes”→“Finish” in order and then restart your RPi. Then the SPI module is started.

Type the following command to check whether the module SPI is loaded successfully:

```
ls /dev/sp*
```

The following result indicates that the module SPI has been loaded successfully:

```
pi@raspberrypi:~ $ ls /dev/sp*
/dev/spidev0.0  /dev/spidev0.1
```

Install Python module SPI-Py

If you use Python language to write the code, please follow the steps below to install the module SPI-Py. If you use C/C++ language, you can skip this step.

Open the terminal and type the following command to install:

```
git clone https://github.com/Freenove/SPI-Py
cd SPI-Py
sudo python3 setup.py install
sudo python2 setup.py install
```

Code

The project code uses human-computer interaction command line mode to read and write the M1-S50 card.

C Code 24.1.1 RFID

First observe the running result, and then learn about the code in detail.

If you need any support, please contact us via: support@freenove.com

1. Use cd command to enter RFID directory of C code.

```
cd Freenove_Kit/Code/C_Code/24.1.1_RFID
```

2. Use the following command to compile and generate executable file "RFID".

```
sudo sh ./build.sh
```

3. Then run the generated file "RFID".

```
sudo ./RFID
```

After the program is executed, the following contents will be displayed in the terminal:

```
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/24.1.1_RFID $ sudo sh ./build.sh
Build finished!
pi@raspberrypi:~/Freenove_Kit/Code/C_Code/24.1.1_RFID $ sudo ./RFID
Try to open device /dev/spidev0.0
Device opened
Device Number:3
SPI mode [OK]
SPI word bits[OK]
SPI max speed[OK]
User Space RC522 Application
RC522>
```

Here, type the command “quit” to exit the program.

Type command "scan", and then the program begins to detect whether there is a card close to the sensing area of MFRC522 reader. Place a M1-S50 card in the sensing area. The following results indicate that the M1-S50 card has been detected, the UID of which is E6CF5C8EFB (HEX).

```
RC522>scan
Scanning
.....
.............Card detected    0xE6 0xCF 0x5C 0x8E, Check Sum = 0xFB
Card Selected, Type:PICC_TYPE_MIFARE_1K
RC522>E6CF5C8E>
```

When the Card is placed in the sensing area, you can read and write the card with the following command.

```
Usage:
        read <blockstart>
        dump
        halt
        clean <blockaddr>
        write <blockaddr> <data>
```

In the command read<blockstart>, the parameter blockstart is the address of the data block, and the range is 0-63. This command is used to display all the data from blockstart address to the end of the sector. For example, sector 0 contains data block 0,1,2,3. Using the command “read 0” can display all contents of data block 0,1,2,3. Using the command “read 1” can display all contents of data block 1,2,3. As is shown below:

```
RC522>E6CF5C8E>read 0
read
Auth Block (0x00) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Read block address 0x00 ....OK read 144 bits
Read block address 0x01 ....OK read 144 bits
Read block address 0x02 ....OK read 144 bits
Read block address 0x03 ....OK read 144 bits
     0: e6 cf 5c 8e fb 08 04 00 62 63 64 65 66 67 68 69 : ..\.....bcdefghi
    16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : ................
    32: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : ................
    48: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .........i......
RC522>E6CF5C8E>read 1
read
Auth Block (0x01) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Read block address 0x01 ....OK read 144 bits
Read block address 0x02 ....OK read 144 bits
Read block address 0x03 ....OK read 144 bits
     0: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : ................
    16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : ................
    32: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .........i......
RC522>E6CF5C8E>
```

Command “dump” is used to display the content of all data blocks in all sectors.

Command <address> <data> is used to write “data" to data block with address “address”, where the address range is 0-63 and the data length is 0-16. For example, if you want to write the string "Freenove" to the data block with address “1”, you can type the following command.

```
write 1 Freenove
```

```
RC522>E6CF5C8E>write 1 Freenove
write
Auth Block (0x01) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Try to write block 1 with 8 byte data...OK
```

Read the contents of this sector and check the data just written.

```
read 0
```

The following results indicate that the string "Freenove" has been written successfully into the data block 1.

```
Read block address 0x03 ....OK read 144 bits
     0: e6 cf 5c 8e fb 08 04 00 62 63 64 65 66 67 68 69 : ..\.....bcdefghi
    16: 46 72 65 65 6e 6f 76 65 00 00 00 00 00 00 00 00 : Freenove........
    32: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : ................
    48: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .........i......
```

Command “clean <address>” is used to remove the contents of the data block with address "address". For example, if you want to clear the contents of the data block 1 that has just been written, you can type the following command.

```
clean 1
```

```
RC522>E6CF5C8E>clean 1
clean
Auth Block (0x01) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Try to clean block 1...OK
```

Read the contents of data blocks in this sector again to check whether the data is erased. The following results indicate that the contents of data block 1 have been erased.

```
RC522>E6CF5C8E>read 0
read
Auth Block (0x00) with key 0xFF 0xFF 0xFF 0xFF 0xFF ...OK
Read block address 0x00 ....OK read 144 bits
Read block address 0x01 ....OK read 144 bits
Read block address 0x02 ....OK read 144 bits
Read block address 0x03 ....OK read 144 bits
     0: e6 cf 5c 8e fb 08 04 00 62 63 64 65 66 67 68 69 : ..\.....bcdefghi
    16: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : ................
    32: 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 : ................
    48: 00 00 00 00 00 00 ff 07 80 69 ff ff ff ff ff ff : .........i......
```

Command "halt" is used to quit the selection state of the card.

```
RC522>E6CF5C8E>halt
halt
Halt
RC522>
```

The following is the program code:

```
#include <stdio.h>
#include <stdint.h>
#include <unistd.h>
#include <string.h>
#include <getopt.h>
#include <stdlib.h>
#include "mfrc522.h"
#define DISP_COMMANDLINE()  printf("RC522>")

int scan_loop(uint8_t *CardID);
int tag_select(uint8_t *CardID);
int main(int argc, char **argv) {
    MFRC522_Status_t ret;
    //Recognized card ID
    uint8_t CardID[5] = { 0x00, };
    static char command_buffer[1024];

    ret = MFRC522_Init('A');
    if (ret < 0) {
        perror("Failed to initialize");
        exit(-1);
    }

    printf("User Space RC522 Application\r\n");

    while (1) {
        /*Main Loop Start*/
        DISP_COMMANDLINE();

```

```
30          scanf("%s", command_buffer);
31          if (strcmp(command_buffer, "scan") == 0) {
32              puts("Scanning");
33              while (1) {
34                  ret = MFRC522_Check(CardID);
35                  if (ret != MI_OK) {
36                      printf(".");
37                      fflush(stdout);
38                      continue;
39                  }
40                  ret |= tag_select(CardID);
41                  if (ret == MI_OK) {
42                      ret = scan_loop(CardID);
43                      if (ret < 0) {
44                          goto END_SCAN;
45                      } else if (ret == 1) {
46                          goto HALT;
47                      }
48                  }
49              }
50              END_SCAN: printf("Card error...");
51              HALT: puts("Halt");
52          } else if (strcmp(command_buffer, "quit") == 0
53                  || strcmp(command_buffer, "exit") == 0) {
54              return 0;
55          } else {
56              puts("Unknown command");
57              puts("scan:scan card and dump");
58              puts("quit:exit program");
59          }
60          /*Main Loop End*/
61      }
62  }
63  int scan_loop(uint8_t *CardID) {
64
65      while (1) {
66
67          char input[32];
68          int block_start;
69          DISP_COMMANDLINE();
70          printf("%02X%02X%02X%02X>", CardID[0], CardID[1], CardID[2], CardID[3]);
71          scanf("%s", input);
72          puts((char*)input);
73          if (strcmp(input, "halt") == 0) {
```

```
74              return 1;
75          } else if (strcmp(input, "dump") == 0) {
76              if (MFRC522_Debug_CardDump(CardID) < 0)
77                  return -1;
78          } else if (strcmp(input, "read") == 0) {
79              scanf("%d", &block_start);
80              if (MFRC522_Debug_DumpSector(CardID, block_start) < 0) {
81                  return -1;
82              }
83          } else if(strcmp(input, "clean") == 0){
84              char c;
85              scanf("%d", &block_start);
86              while ((c = getchar()) != '\n' && c != EOF)
87                  ;
88              if (MFRC522_Debug_Clean(CardID, block_start)) {
89                  return -1;
90              }
91
92          } else if (strcmp(input, "write") == 0) {
93              char write_buffer[256];
94              size_t len = 0;
95              scanf("%d", &block_start);
96              scanf("%s",write_buffer);
97              if (len >= 0) {
98                  if (MFRC522_Debug_Write(CardID, block_start, write_buffer,
99                          strlen(write_buffer)) < 0) {
100                     return -1;
101                 }
102             }
103         } else {
104
105             printf(
106                     "Usage:\r\n" "\tread <blockstart>\r\n" "\tdump\r\n"
107 "\thalt\r\n" "\tclean <blockaddr>\r\n" "\twrite <blockaddr> <data>\r\n");
108             return 0;
109         }
110     }
111     return 0;
112
113 }
114 int tag_select(uint8_t *CardID) {
115     int ret_int;
116     printf(
117             "Card detected    0x%02X 0x%02X 0x%02X 0x%02X, Check Sum = 0x%02X\r\n",
```

```
            CardID[0], CardID[1], CardID[2], CardID[3], CardID[4]);
    ret_int = MFRC522_SelectTag(CardID);
    if (ret_int == 0) {
        printf("Card Select Failed\r\n");
        return -1;
    } else {
        printf("Card Selected, Type:%s\r\n",
                MFRC522_TypeToString(MFRC522_ParseType(ret_int)));
    }
    ret_int = 0;
    return ret_int;
}
```

In the code, first initialize the MFRC522. If the initialization fails, the program will exit.

```
    ret = MFRC522_Init('A');
    if (ret < 0) {
        perror("Failed to initialize");
        exit(-1);
    }
```

In the main function, wait for the command input. If command "scan" is received, the function will begin to detect whether there is a card close to the sensing area. If a card is detected, the card will be selected and card UID will be acquired. Then enter the function scan_loop (). If command "quit" or "exit" is received, the program will exit.

```
        scanf("%s", command_buffer);
        if (strcmp(command_buffer, "scan") == 0) {
            puts("Scanning");
            while (1) {
                ret = MFRC522_Check(CardID);
                if (ret != MI_OK) {
                    printf(".");
                    fflush(stdout);
                    continue;
                }
                ret |= tag_select(CardID);
                if (ret == MI_OK) {
                    ret = scan_loop(CardID);
                    if (ret < 0) {
                        goto END_SCAN;
                    } else if (ret == 1) {
                        goto HALT;
                    }
                }
            }
            END_SCAN: printf("Card error...");
            HALT: puts("Halt");
```

```
        } else if (strcmp(command_buffer, "quit") == 0
                || strcmp(command_buffer, "exit") == 0) {
            return 0;
        } else {
            puts("Unknown command");
            puts("scan:scan card and dump");
            puts("quit:exit program");
        }
        /*Main Loop End*/
    }
```

The function scan_loop() will detect command read, write, clean, halt, dump and do the corresponding processing to each command. The functions of each command and the method have been introduced before.

```
int scan_loop(uint8_t *CardID) {

    while (1) {

        char input[32];
        int block_start;
        DISP_COMMANDLINE();
        printf("%02X%02X%02X%02X>", CardID[0], CardID[1], CardID[2], CardID[3]);
        scanf("%s", input);
        puts((char*)input);
        if (strcmp(input, "halt") == 0) {
            return 1;
        } else if (strcmp(input, "dump") == 0) {
            if (MFRC522_Debug_CardDump(CardID) < 0)
                return -1;
        } else if (strcmp(input, "read") == 0) {
            scanf("%d", &block_start);
            if (MFRC522_Debug_DumpSector(CardID, block_start) < 0) {
                return -1;
            }
        } else if(strcmp(input, "clean") == 0){
            char c;
            scanf("%d", &block_start);
            while ((c = getchar()) != '\n' && c != EOF)
                ;
            if (MFRC522_Debug_Clean(CardID, block_start)) {
                return -1;
            }

        } else if (strcmp(input, "write") == 0) {
            char write_buffer[256];
            size_t len = 0;
```

```
            scanf("%d", &block_start);
            scanf("%s",write_buffer);
            if (len >= 0) {
                if (MFRC522_Debug_Write(CardID, block_start, write_buffer,
                        strlen(write_buffer)) < 0) {
                    return -1;
                }
            }
        } else {

            printf(
                    "Usage:\r\n" "\tread <blockstart>\r\n" "\tdump\r\n" "\thalt\r\n"
                        "\tclean <blockaddr>\r\n" "\twrite <blockaddr> <data>\r\n");
            return 0;
        }
    }
    return 0;

}
```

The header file "mfrc522.h" contains the associated operation method for the MFRC522. You can open the file to view all the definitions and functions.

Python Code 24.1.1 RFID

There are two code files for this project. They are respectively under Python2 folder and Python3 folder. **Their functions are the same, but they are not compatible.** Code under Python2 folder can only run on Python2. And code under Python3 folder can only run on Python3.

First observe the project result, and then learn about the code in detail.

If you need any support, please contact us via: support@freenove.com

1. Use cd command to enter RFID directory of Python code.

   If you use Python2, it is needed to enter Python2 code folder.

```
cd ~/Freenove_Kit/Code/Python_Code/24.1.1_RFID/Python2
```

   If you use Python3, it is needed to enter Python3 code folder.

```
cd ~/Freenove_Kit/Code/Python_Code/24.1.1_RFID/Python3
```

2. Use python command to execute code "RFID.py".

```
python RFID.py
```

After the program is executed, the following contents will be displayed in the terminal:

```
pi@raspberrypi:~ $ cd ~/Freenove_Kit/Code/Python_Code/24.1.1_RFID/Python3
pi@raspberrypi:~/Freenove_Kit/Code/Python_Code/24.1.1_RFID/Python3 $ python RFID.py
Program is starting ...
Press Ctrl-C to exit.
RC522>
```

Here, type the command “quit” to exit the program.

Type command "scan", then the program begins to detect whether there is a card close to the sensing area of MFRC522 reader. Place a M1-S50 card in the sensing area. The following results indicate that the M1-S50 card has been detected, the UID of which is E6CF5C8EFB (HEX).

```
RC522> scan
scan
Scanning ...
Card detected
Card UID: ['0xe6', '0xcf', '0x5c', '0x8e', '0xfb']
Size: 8
RC522> E6CF5C8EFB>
```

When the Card is placed in the sensing area, you can read and write the card with the following command.

```
Usage:
        read <blockstart>
        dump
        halt
        clean <blockaddr>
        write <blockaddr> <data>
```

In the command read<blockstart>, the parameter blockstart is the address of the data block, and the range is 0-63. As is shown below:

In the command read<blockstart>, the parameter blockstart is the address of the data block, and the range is 0-63. This command is used to read the data of data block with address “blockstart”. For example, using command “read 0” can display the content of data block 0. Using the command “read 1” can display the content of data block 1. As is shown below:

```
RC522> E6CF5C8EFB> read 0
['read', '0']
Sector 0 :  e6 cf 5c 8e fb 8 4 0 62 63 64 65 66 67 68 69  | ??bcdefghi
RC522> E6CF5C8EFB> read 1
['read', '1']
Sector 1 :  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  |
RC522> E6CF5C8EFB>
```

Command “dump” is used to display the content of all data blocks in all sectors.

Command <address> <data> is used to write “data" to data block with address “address”, where the address range is 0-63 and the data length is 0-16. In the process of writing data to the data block, both the contents of data block before written and after written will be displayed. For example, if you want to write the string "Freenove" to the data block with address “1”, you can type the following command.

```
write 1 Freenove
```

```
RC522> E6CF5C8EFB> write 1 Freenove
['write', '1', 'Freenove']
Before writing , The data in block 1  is:
Sector 1 :  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  |
4 backdata &0x0F == 0x0A 10
Data written
After written , The data in block 1  is:
Sector 1 :  46 72 65 65 6e 6f 76 65 0 0 0 0 0 0 0 0  | Freenove
```

Command “clean <address>” is used remove the contents of the data block with address "address". For example, if you want to clear the contents of the data block 1 that has just been written, you can type the following command.

```
clean 1
```

```
RC522> E6CF5C8EFB> clean 1
['clean', '1']
Before cleaning , The data in block 1  is:
Sector 1 :  46 72 65 65 6e 6f 76 65 0 0 0 0 0 0 0 0  | Freenove
4 backdata &0x0F == 0x0A 10
Data written
After cleaned , The data in block 1  is:
Sector 1 :  0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0  |
```

Command “halt” is used to quit the selection state of the card.

```
RC522> E6CF5C8EFB> halt
['halt']
RC522>
```

The following is the program code (python2 code):

```
1  import RPi.GPIO as GPIO
2  import MFRC522
3
4  # Create an object of the class MFRC522
5  mfrc = MFRC522.MFRC522()
6
7  def dis_CommandLine():
8      print "RC522>",
9  def dis_CardID(cardID):
10     print "%2X%2X%2X%2X%2X>"%(cardID[0],cardID[1],cardID[2],cardID[3],cardID[4]),
11 def setup():
```

```
    print "Program is starting ... "
    print "Press Ctrl-C to exit."
    pass

def loop():
    while(True):
        dis_ConmandLine()
        inCmd = raw_input()
        print inCmd
        if (inCmd == "scan"):
            print "Scanning ... "
            isScan = True
            while isScan:
                # Scan for cards
                (status,TagType) = mfrc.MFRC522_Request(mfrc.PICC_REQIDL)
                # If a card is found
                if status == mfrc.MI_OK:
                    print "Card detected"
                # Get the UID of the card
                (status,uid) = mfrc.MFRC522_Anticoll()
                # If we have the UID, continue
                if status == mfrc.MI_OK:
                    print "Card UID: "+ str(map(hex,uid))
                    # Select the scanned tag
                    if mfrc.MFRC522_SelectTag(uid) == 0:
                        print "MFRC522_SelectTag Failed!"
                    if cmdloop(uid) < 1 :
                        isScan = False
        elif inCmd == "quit":
            destroy()
            exit(0)
        else :
            print "\tUnknown command\n"+"\tscan:scan card and dump\n"+"\tquit:exit
program\n"

def cmdloop(cardID):
    pass
    while(True):
        dis_ConmandLine()
        dis_CardID(cardID)
        inCmd = raw_input()
        cmd = inCmd.split(" ")
        print cmd
        if(cmd[0] == "read"):
```

```
            blockAddr = int(cmd[1])
            if((blockAddr<0) or (blockAddr>63)):
                print "Invalid Address!"
            # This is the default key for authentication
            key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
            # Authenticate
            status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
            # Check if authenticated
            if status == mfrc.MI_OK:
                mfrc.MFRC522_Readstr(blockAddr)
            else:
                print "Authentication error"
                return 0

        elif cmd[0] == "dump":
            # This is the default key for authentication
            key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
            mfrc.MFRC522_Dump_Str(key,cardID)

        elif cmd[0] == "write":
            blockAddr = int(cmd[1])
            if((blockAddr<0) or (blockAddr>63)):
                print "Invalid Address!"
            data = [0]*16
            if(len(cmd)<2):
                data = [0]*16
            else:
                data = cmd[2][0:17]
                data = map(ord,data)
                if len(data)<16:
                    data+=[0]*(16-len(data))
            # This is the default key for authentication
            key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
            # Authenticate
            status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
            # Check if authenticated
            if status == mfrc.MI_OK:
                print "Before writing , The data in block %d  is: "%(blockAddr)
                mfrc.MFRC522_Readstr(blockAddr)
                mfrc.MFRC522_Write(blockAddr, data)
                print "After written , The data in block %d  is: "%(blockAddr)
                mfrc.MFRC522_Readstr(blockAddr)
            else:
                print "Authentication error"
```

```
100                 return 0
101
102         elif cmd[0] == "clean":
103             blockAddr = int(cmd[1])
104             if((blockAddr<0) or (blockAddr>63)):
105                 print "Invalid Address!"
106             data = [0]*16
107             # This is the default key for authentication
108             key = [0xFF,0xFF,0xFF,0xFF,0xFF,0xFF]
109             # Authenticate
110             status = mfrc.MFRC522_Auth(mfrc.PICC_AUTHENT1A, blockAddr, key, cardID)
111             # Check if authenticated
112             if status == mfrc.MI_OK:
113                 print "Before cleaning , The data in block %d  is: "%(blockAddr)
114                 mfrc.MFRC522_Readstr(blockAddr)
115                 mfrc.MFRC522_Write(blockAddr, data)
116                 print "After cleaned , The data in block %d  is: "%(blockAddr)
117                 mfrc.MFRC522_Readstr(blockAddr)
118             else:
119                 print "Authentication error"
120                 return 0
121         elif cmd[0] == "halt":
122             return 0
123         else :
124             print "Usage:\r\n" "\tread <blockstart>\r\n" "\tdump\r\n" "\thalt\r\n"
125 "\tclean <blockaddr>\r\n" "\twrite <blockaddr> <data>\r\n"
126
127 def destroy():
128     GPIO.cleanup()
129
130 if __name__ == "__main__":
131     setup()
132     try:
133         loop()
134     except KeyboardInterrupt:  # Ctrl+C captured, exit
135         destroy()
```

In the code, first create an MFRC522 class object.

```
mfrc = MFRC522.MFRC522()
```

In the function loop, wait for the command input. If command "scan" is received, the function will begin to detect whether there is a card close to the sensing area. If a card is detected, the card will be selected and card UID will be acquired. Then enter the function scan_loop (). If command "quit" or "exit" is received, the program will exit.

```
        if (inCmd == "scan"):
            print "Scanning ... "
```

```
            isScan = True
            while isScan:
                ............
                    if cmdloop(uid) < 1 :
                        isScan = False
        elif inCmd == "quit":
            destroy()
            exit(0)
        else :
            print "\tUnknown command\n"+"\tscan:scan card and dump\n"+"\tquit:exit
program\n"
```

The function cmdloop() will detect command read, write, clean, halt, dump and do the corresponding processing to each command. The functions of each command and the method have been introduced before.

```
def cmdloop(cardID):
    pass
    while(True):
        dis_ConmandLine()
        dis_CardID(cardID)
        inCmd = raw_input()
        cmd = inCmd.split(" ")
        print cmd
        if(cmd[0] == "read"):
            ............
        elif cmd[0] == "dump":
            ............
        elif cmd[0] == "write":
            ............
        elif cmd[0] == "clean":
            ............
        elif cmd[0] == "halt":
            return 0
        else :
            print "Usage:\r\n" "\tread <blockstart>\r\n" "\tdump\r\n" "\thalt\r\n"
"\tclean <blockaddr>\r\n" "\twrite <blockaddr> <data>\r\n"
```

The file "MFRC522.py" contains the associated operation method for the MFRC522. You can open the file to view all the definitions and functions.

Chapter 25 Web IoT

In this chapter, we will learn how to use GPIO to control the RPi remotely via a network and how to build a WebIO service on the RPi.
This concept is known as “IoT” or Internet of Things. The development of IoT will greatly change our habits and make our lives more convenient and efficient

Project 25.1 Remote LED

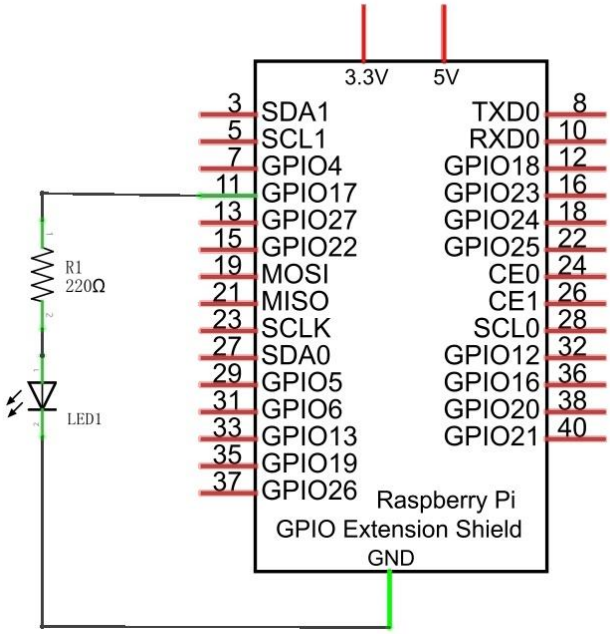
In this project, we need to build a WebIOPi service, and then use the RPi GPIO to control an LED through the web browser of phone or PC.

Component List

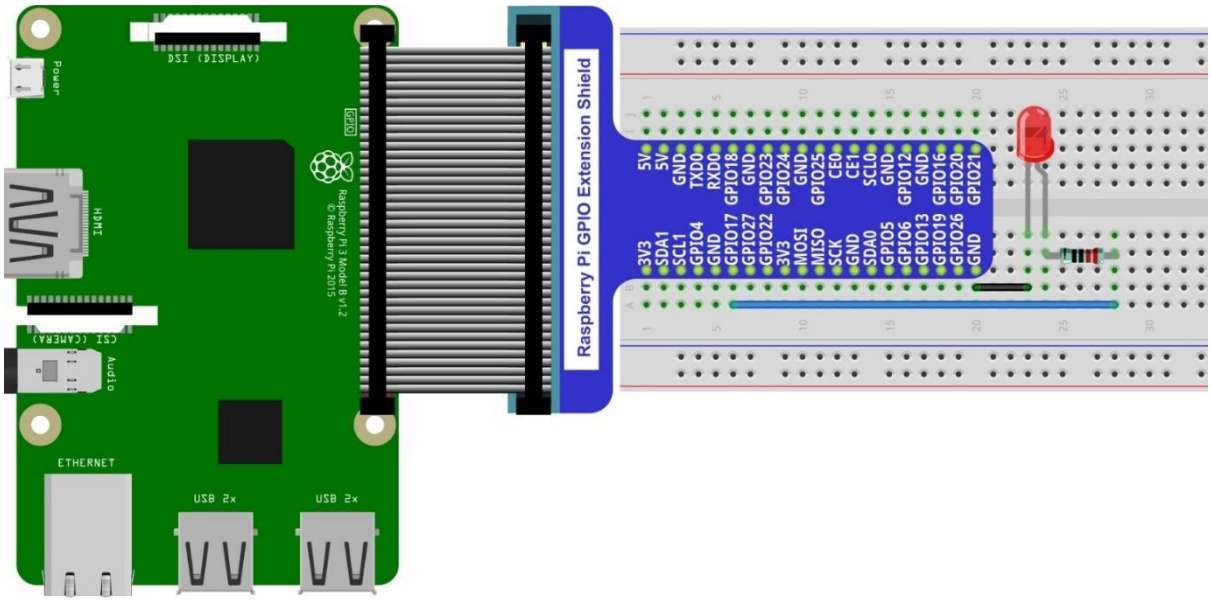
Raspberry Pi (with 40 GPIO) x1 GPIO Extension Board & Ribbon Cable x1 Breadboard x1	LED x1 	Resistor 220Ω x1 
Jumper M/M x2 		

Circuit

Schematic diagram



Hardware connection. If you need any support, please feel free to contact us via: support@freenove.com



Solution from E-Tinkers

Here is a solution fromblog E-Tinkers, author Henry Cheung. For more details, please refer to link below:
https://www.e-tinkers.com/2018/04/how-to-control-raspberry-pi-gpio-via-http-web-server/

1, Make sure you have set python3 as default python. Then run following command in terminal to install http.server in your Raspberry Pi.

```
sudo apt-get install http.server
```

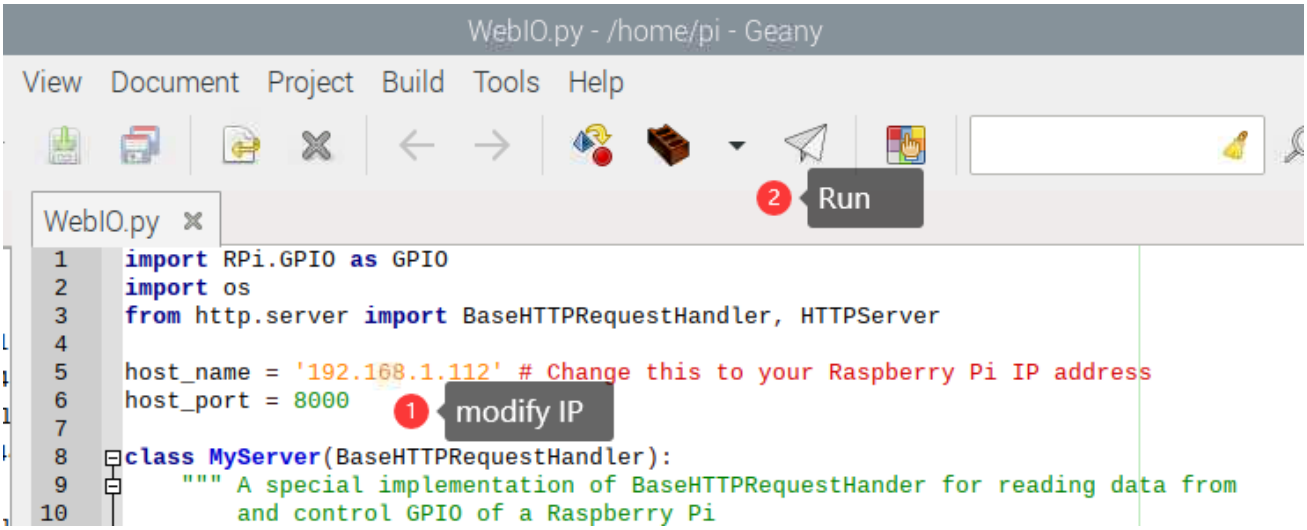
2, Open WebIO.py

```
cd ~/Freenove_Kit/Code/Python_Code/25.1.1_WebIO
geany WebIO.py
```

3, Change the host_name into your Raspberry Pi IP address.

```
host_name = '192.168.1.112'     # Change this to your Raspberry Pi IP address
```

Then run the code WebIO.py



3, Visit http://192.168.1.112:8000/ in web brower on compter under local area networks. **Change IP to your Raspberry Pi IP address.**

WebIOPi Service Framework

Note: **If you have a Raspberry Pi 4B, you may have some trouble. The reason for changing the file in the configuration process is that the newer generation models of the RPi CPUs are different form the older ones and you may not be able to access the GPIO Header at the end of this tutorial.** A solution to this is given in an online tutorial by from E-Tinkers blogger Henry Cheung. For more details, please refer to previouse section.

The following is the key part of this chapter. The installation steps refer to WebIOPi official. And you also can directly refer to the official installation steps. The latest version (in 2016-6-27) of WebIOPi is 0.7.1. So, you may encounter some issues in using it. We will explain these issues and provide the solution in the following installation steps.

Here are the steps to build a WebIOPi:

Installation

1. Get the installation package. You can use the following command to obtain.

```
wget https://github.com/Freenove/WebIOPi/archive/master.zip -O WebIOPi.zip
```

2. Extract the package and generate a folder named "WebIOPi-master". Then enter the folder.

```
unzip WebIOPi.zip
cd WebIOPi-master/WebIOPi-0.7.1
```

3. Patch for Raspberry Pi B+, 2B, 3B, 3B+.

```
patch -p1 -i webiopi-pi2bplus.patch
```

4. Run setup.sh to start the installation, the process takes a while and you will need to be patient.

```
sudo ./setup.sh
```

5. If setup.sh does not have permission to execute, execute the following command

```
sudo sh ./setup.sh
```

Run

After the installation is completed, you can use the webiopi command to start running.

```
$ sudo webiopi [-h] [-c config] [-l log] [-s script] [-d] [port]

    Options:
      -h, --help            Display this help
      -c, --config  file    Load config from file
      -l, --log     file    Log to file
      -s, --script  file    Load script from file
      -d, --debug           Enable DEBUG

    Arguments:
      port                  Port to bind the HTTP Server
```

Run webiopi with verbose output and the default config file:

```
sudo webiopi -d -c /etc/webiopi/config
```

The Port is 8000 in default. Now WebIOPi has been launched. Keep it running.

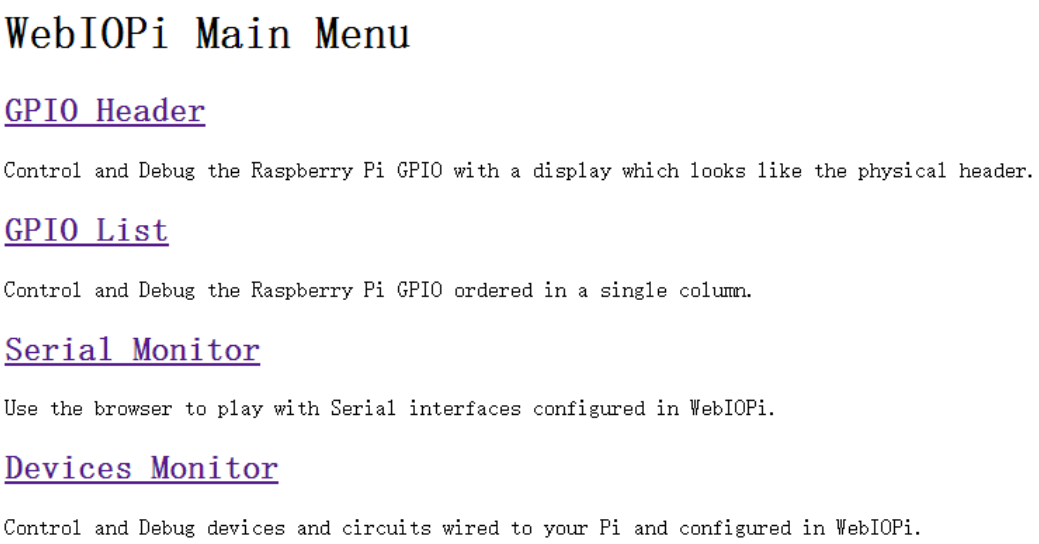
Access WebIOPi over local network

Under the same network, use a mobile phone or PC browser to open your RPi IP address, and add a port

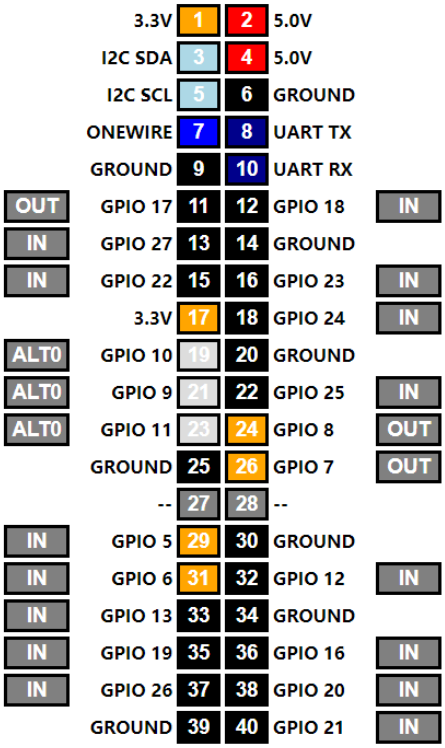
number like 8000. For example, my personal Raspberry Pi IP address is 192.168.1.109. Then, in the browser, I then should input: http://192.168.1.109:8000/

Default user is "webiopi" and password is "raspberry"。

Then, enter the main control interface:



Click on GPIO Header to enter the GPIO control interface.



Control methods:

- Click/Tap the OUT/IN button to change GPIO direction.
- Click/Tap pins to change the GPIO output state.

Completed

According to the circuit we build, set GPIO17 to OUT, then click Header11 to control the LED.

You can end the webioPi in the terminal by "Ctr+C".

What's Next?

THANK YOU for participating in this learning experience! If you have completed all of the projects successfully you can consider yourself a Raspberry Pi Master.

We have reached the end of this Tutorial. If you find errors, omissions or you have suggestions and/or questions about the Tutorial or component contents of this Kit, please feel free to contact us: **support@freenove.com**
We will make every effort to make changes and correct errors as soon as feasibly possible and publish a revised version.

If you are interesting in processing, you can study the Processing.pdf in the unzipped folder.

If you want to learn more about Arduino, Raspberry Pi, Smart Cars, Robotics and other interesting products in science and technology, please continue to visit our website. We will continue to launch fun, cost-effective, innovative and exciting products.
http://www.freenove.com/

Thank you again for choosing Freenove products.