anticlockwise, four-step as a cycle. Where parameter "dir" indicates the rotation direction, if "dir" is 1, the servo will rotate clockwise, otherwise it rotates to anticlockwise. Parameter "ms" indicates the time between each two steps. The "ms" of Stepper Motor used in this project is 3ms (the shortest time period), a value of less than 3ms will exceed the limits of the Stepper Motor with a result that it does not rotate.

```
void moveOnePeriod(int dir,int ms){
    int i=0,j=0;
    for (j=0;j<4;j++){  //cycle according to power supply order
        for (i=0;i<4;i++){  //assign to each pin, a total of 4 pins
            if(dir == 1)    //power supply order clockwise
                digitalWrite(motorPins[i],(CCWStep[j] == (1<<i)) ? HIGH : LOW);
            else        //power supply order anticlockwise
                digitalWrite(motorPins[i],(CWStep[j] == (1<<i)) ? HIGH : LOW);
            printf("motorPin %d, %d \n",motorPins[i],digitalRead(motorPins[i]));
        }
        printf("Step cycle!\n");
        if(ms<3)        //the delay can not be less than 3ms, otherwise it will exceed
speed limit of the motor
            ms=3;
        delay(ms);
    }
}
```

Subfunction **moveSteps** (int dir, int ms, int steps) is used to specific cycle number of Stepper Motor.

```
void moveSteps(int dir, int ms, int steps){
    int i;
    for(i=0;i<steps;i++){
        moveOnePeriod(dir,ms);
    }
}
```

Subfunction **motorStop** () is used to stop the Stepper Motor.

```
void motorStop(){   //function used to stop rotating
    int i;
    for(i=0;i<4;i++){
        digitalWrite(motorPins[i],LOW);
    }
}
```

Finally, in the while loop of main function, rotate one revolution clockwise, and then one revolution anticlockwise. According to the previous material covered, the Stepper Motor one revolution requires 2048 steps, that is, 2048/4=512 cycle.

```
    while(1){
        moveSteps(1,3,512);     //rotating  360°  clockwise, a total of 2048 steps in a
circle, namely, this function(four steps) will be called 512 times.
        delay(500);
        moveSteps(0,3,512);     //rotating 360°  anticlockwise
        delay(500);
```

```
    }
```

### Python Code 16.1.1 SteppingMotor

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 16.1.1_SteppingMotor directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/16.1.1_SteppingMotor
```

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

```
python SteppingMotor.py
```

After the program is executed, the Stepper Motor will rotate 360° clockwise and then 360° anticlockwise and repeat this action in an endless loop.

The following is the program code:

```
import RPi.GPIO as GPIO
import time

motorPins = (12, 16, 18, 22)    # define pins connected to four phase ABCD of stepper motor
CCWStep = (0x01,0x02,0x04,0x08) # define power supply order for rotating anticlockwise
CWStep = (0x08,0x04,0x02,0x01)  # define power supply order for rotating clockwise

def setup():
    GPIO.setmode(GPIO.BOARD)       # use PHYSICAL GPIO Numbering
    for pin in motorPins:
        GPIO.setup(pin,GPIO.OUT)

# as for four phase Stepper Motor, four steps is a cycle. the function is used to drive the
Stepper Motor clockwise or anticlockwise to take four steps
def moveOnePeriod(direction,ms):
    for j in range(0,4,1):      # cycle for power supply order
        for i in range(0,4,1):  # assign to each pin
            if (direction == 1):# power supply order clockwise
                GPIO.output(motorPins[i],((CCWStep[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
            else :              # power supply order anticlockwise
                GPIO.output(motorPins[i],((CWStep[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
        if(ms<3):       # the delay can not be less than 3ms, otherwise it will exceed speed
limit of the motor
            ms = 3
        time.sleep(ms*0.001)

# continuous rotation function, the parameter steps specify the rotation cycles, every four
steps is a cycle
def moveSteps(direction, ms, steps):
    for i in range(steps):
        moveOnePeriod(direction, ms)

```

```
# function used to stop motor
def motorStop():
    for i in range(0,4,1):
        GPIO.output(motorPins[i],GPIO.LOW)

def loop():
    while True:
        moveSteps(1,3,512)  # rotating 360 deg clockwise, a total of 2048 steps in a circle,
512 cycles
        time.sleep(0.5)
        moveSteps(0,3,512)  # rotating 360 deg anticlockwise
        time.sleep(0.5)

def destroy():
    GPIO.cleanup()             # Release resource

if __name__ == '__main__':     # Program entrance
    print ('Program is starting...')
    setup()
    try:
        loop()
    except KeyboardInterrupt:  # Press ctrl-c to end the program.
        destroy()
```

In the code we define the four pins of the Stepper Motor and the order to supply power to the coils for a four-step rotation mode.

```
motorPins = (12, 16, 18, 22)    #define pins connected to four phase ABCD of stepper
motor

CCWStep = (0x01,0x02,0x04,0x08) #define power supply order for coil for rotating
anticlockwise
CWStep = (0x08,0x04,0x02,0x01)  #define power supply order for coil for rotating
clockwise
```

Subfunction **moveOnePeriod** ((int dir, int ms) will drive the Stepper Motor rotating four-step clockwise or anticlockwise, four-step as a cycle. Where parameter "dir" indicates the rotation direction, if "dir" is 1, the servo will rotate clockwise, otherwise it rotates to anticlockwise. Parameter "ms" indicates the time between each two steps. The "ms" of Stepper Motor used in this project is 3ms (the shortest time period), a value of less than 3ms will exceed the limits of the Stepper Motor with a result that it does not rotate.

```
def moveOnePeriod(direction,ms):
    for j in range(0,4,1):      #cycle for power supply order
        for i in range(0,4,1):  #assign to each pin, a total of 4 pins
            if (direction == 1):#power supply order clockwise
                GPIO.output(motorPins[i],((CCWStep[j] == 1<<i) and GPIO.HIGH orGPIO.LOW))
            else :              #power supply order anticlockwise
                GPIO.output(motorPins[i],((CWStep[j] == 1<<i) and GPIO.HIGH or GPIO.LOW))
```

```
        if(ms<3):       #the delay can not be less than 3ms, otherwise it will exceed
speed limit of the motor
            ms = 3
        time.sleep(ms*0.001)
```

Subfunction **moveSteps** (direction, ms, steps) is used to specify the cycle number of Stepper Motor.

```
def moveSteps(direction, ms, steps):
    for i in range(steps):
        moveOnePeriod(direction, ms)
```

Subfunction **motorStop** () is used to stop the Stepper Motor.

```
def motorStop():
    for i in range(0,4,1):
        GPIO.output(motorPins[i],GPIO.LOW)
```

Finally, in the while loop of main function, rotate one revolution clockwise, and then one revolution anticlockwise. According to the previous material covered, the Stepper Motor one revolution requires 2048 steps, that is, 2048/4=512 cycle.

```
    while True:
        moveSteps(1,3,512)  #rotating  360°  clockwise, a total of 2048 steps in a
circle, namely, 512 cycles.
        time.sleep(0.5)
        moveSteps(0,3,512)  #rotating 360°  anticlockwise
        time.sleep(0.5)
```

# Chapter 17 74HC595 & Bar Graph LED

We have used LED Bar Graph to make a flowing water light, in which 10 GPIO ports of RPi are occupied. More GPIO ports mean that more peripherals can be connected to RPi, so GPIO resource is very precious. Can we make flowing water light with less GPIO ports? In this chapter, we will learn a component, 74HC595, which can achieve the target.

## Project 17.1 Flowing Water Light

Now let us learn how to use the 74HC595 IC Chip to make a flowing water light using less GPIO.

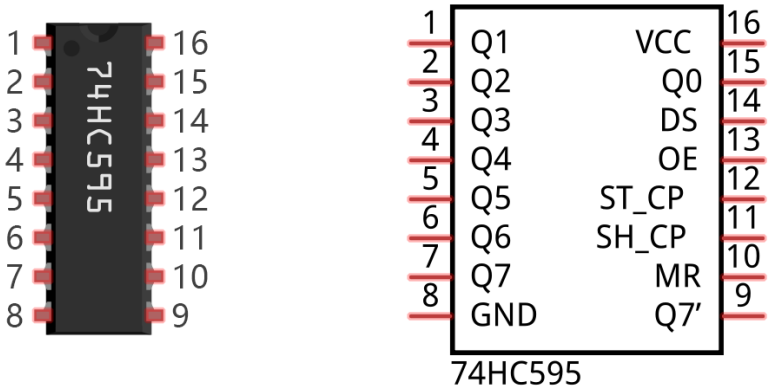
### Component List

<table>
<tr><td colspan="2">Raspberry Pi (with 40 GPIO) x1<br>GPIO Extension Board & Ribbon Cable x1<br>Breadboard x1</td><td>Jumper x17<br>  </td></tr>
<tr><td>74HC595 x1<br>  </td><td>Bar Graph LED x1<br>  </td><td>Resistor 220Ω x8<br>  </td></tr>
</table>

# Component knowledge

## 74HC595

A 74HC595 chip is used to convert serial data into parallel data. A 74HC595 chip can convert the serial data of one byte into 8 bits, and send its corresponding level to each of the 8 ports correspondingly. With this characteristic, the 74HC595 chip can be used to expand the IO ports of a Raspberry Pi. At least 3 ports on the RPI board are required to control the 8 ports of the 74HC595 chip.



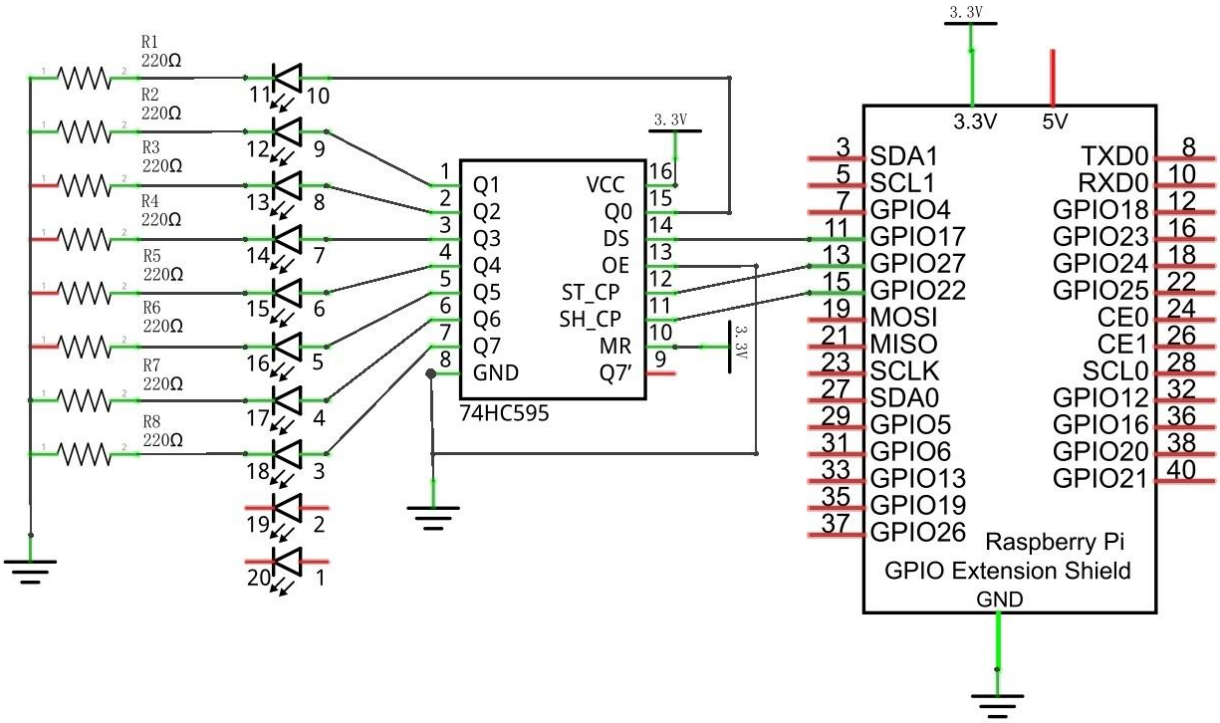

The ports of the 74HC595 chip are described as follows:

| Pin name | Pin number | Description |
|---|---|---|
| Q0-Q7 | 15, 1-7 | Parallel Data Output |
| VCC | 16 | The Positive Electrode of the Power Supply, the Voltage is 2~6V |
| GND | 8 | The Negative Electrode of Power Supply |
| DS | 14 | Serial Data Input |
| OE | 13 | Enable Output,<br>When this pin is in high level, Q0-Q7 is in high resistance state<br>When this pin is in low level, Q0-Q7 is in output mode |
| ST_CP | 12 | Parallel Update Output: when its electrical level is rising, it will update the parallel data output. |
| SH_CP | 11 | Serial Shift Clock: when its electrical level is rising, serial data input register will do a shift. |
| MR | 10 | Remove Shift Register: When this pin is in low level, the content in shift register will be cleared. |
| Q7' | 9 | Serial Data Output: it can be connected to more 74HC595 chips in series. |

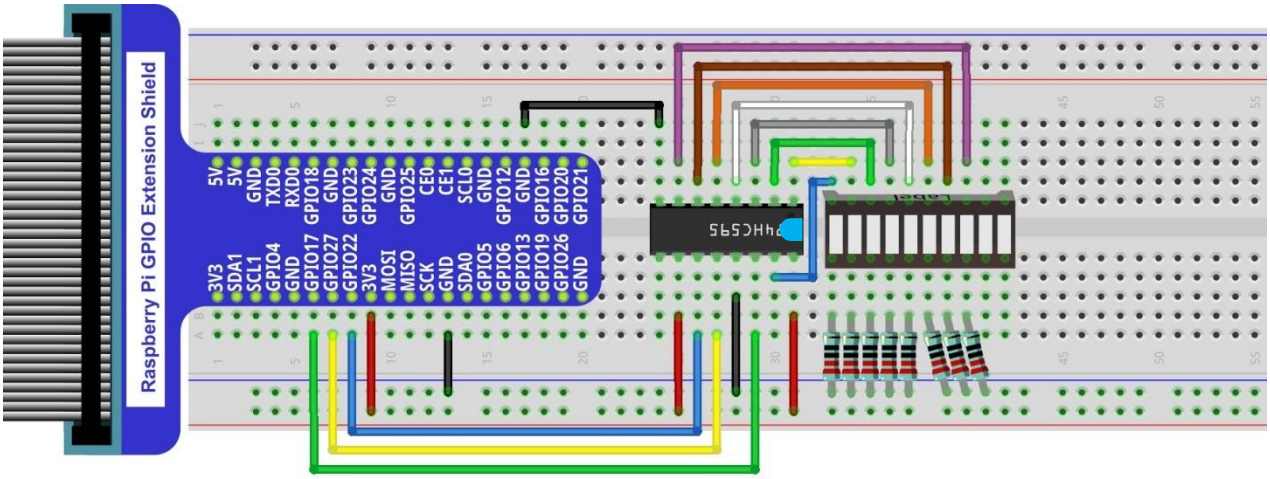
For more details, please refer to the datasheet on the 74HC595 chip.

# Circuit

Schematic diagram




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



# Code

In this project we will make a flowing water light with a 74HC595 chip to learn about its functions.

C Code 17.1.1 LightWater02

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 17.1.1_LightWater02 directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/17.1.1_LightWater02
```

2. Use following command to compile "LightWater02.c" and generate executable file "LightWater02".

```
gcc LightWater02.c -o LightWater02 -lwiringPi
```

3. Then run the generated file "LightWater02".

```
sudo ./LightWater02
```

After the program is executed, you will see that Bar Graph LED starts with the flowing water pattern flashing from left to right and then back from right to left.

The following is the program code:

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

#define   dataPin   0   //DS Pin of 74HC595(Pin14)
#define   latchPin  2   //ST_CP Pin of 74HC595(Pin12)
#define   clockPin 3    //CH_CP Pin of 74HC595(Pin11)

void _shiftOut(int dPin,int cPin,int order,int val){
    int i;
    for(i = 0; i < 8; i++){
        digitalWrite(cPin,LOW);
        if(order == LSBFIRST){
            digitalWrite(dPin,((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
            delayMicroseconds(10);
         }
        else {//if(order == MSBFIRST){
            digitalWrite(dPin,((0x80&(val<<i)) == 0x80) ? HIGH : LOW);
            delayMicroseconds(10);
         }
        digitalWrite(cPin,HIGH);
        delayMicroseconds(10);
    }
}

int main(void)
{
    int i;
```

```
29      unsigned char x;
30
31      printf("Program is starting ...\n");
32
33      wiringPiSetup();
34
35      pinMode(dataPin,OUTPUT);
36      pinMode(latchPin,OUTPUT);
37      pinMode(clockPin,OUTPUT);
38      while(1){
39          x=0x01;
40          for(i=0;i<8;i++){
41              digitalWrite(latchPin,LOW);         // Output low level to latchPin
42              _shiftOut(dataPin,clockPin,LSBFIRST,x);// Send serial data to 74HC595
43              digitalWrite(latchPin,HIGH);    //Output high level to latchPin, and 74HC595 will
44  update the data to the parallel output port.
45              x<<=1;      //make the variable move one bit to left once, then the bright LED
46  move one step to the left once.
47              delay(100);
48          }
49          x=0x80;
50          for(i=0;i<8;i++){
51              digitalWrite(latchPin,LOW);
52              _shiftOut(dataPin,clockPin,LSBFIRST,x);
53              digitalWrite(latchPin,HIGH);
54              x>>=1;
55              delay(100);
56          }
57      }
58      return 0;
59  }
```

In the code, we configure three pins to control the 74HC595 chip and define a one-byte variable to control the state of the 8 LEDs (in the Bar Graph LED Module) through the 8 bits of the variable. The LEDs light ON when the corresponding bit is 1. If the variable is assigned to 0x01, that is 00000001 in binary, there will be only one LED ON.

```
x=0x01;
```

In the “while” cycle of main function, use two cycles to send x to 74HC595 output pin to control the LED. In one cycle, x will shift one bit to the LEFT in one cycle, then when data of x is sent to 74HC595, the LED that is turned ON will move one bit to the LEFT once.

```
        for(i=0;i<8;i++){
            digitalWrite(latchPin,LOW);      // Output low level to latchPin
            _shiftOut(dataPin,clockPin,LSBFIRST,x);// Send serial data to 74HC595
            digitalWrite(latchPin,HIGH); // Output high level to latchPin, and 74HC595
will update the data to the parallel output port.
```

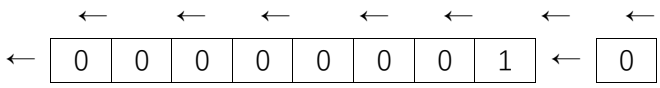
```
            x<<=1; // make the variable move one bit to left once, then the bright LED
move one step to the left once.
            delay(100);
        }
```

In second cycle, the situation is the same. The difference is that x is shift from 0x80 to the RIGHT in order.

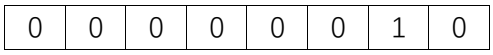
**<< operator**

"<<" is the left shift operator, which can make all bits of 1 byte shift by several bits to the left (high) direction and add 0 on the right (low). For example, shift binary 00000001 by 1 bit to left:

byte x = 1 << 1;

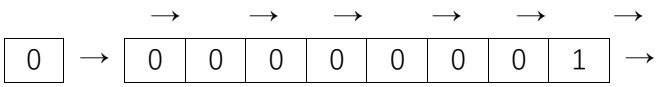



The result of x is 2 (binary 00000010).

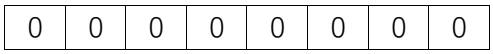



There is another similar operator" >>". For example, shift binary 00000001 by 1 bit to right:

byte x = 1 >> 1;




The result of x is 0 (00000000).




X <<= 1 is equivalent to x = x << 1 and x >>= 1 is equivalent to x = x >> 1

About shift function

**uint8_t shiftIn (uint8_t dPin, uint8_t cPin, uint8_t order) ;**

This is used to shift an 8-bit data value in with the data appearing on the dPin and the clock being sent out on the cPin. Order is either LSBFIRST or MSBFIRST. The data is sampled after the cPin goes high. (So cPin high, sample data, cPin low, repeat for 8 bits) The 8-bit value is returned by the function.

**void shiftOut (uint8_t dPin, uint8_t cPin, uint8_t order, uint8_t val) ;**

**void _shiftOut (uint8_t dPin, uint8_t cPin, uint8_t order, uint8_t val) ;**

This is used to shift an 8-bit data value out with the data being sent out on dPin and the clock being sent out on the cPin. order is as above. Data is clocked out on the rising or falling edge - ie. dPin is set, then cPin is taken high then low - repeated for the 8 bits.

For more details about shift function, please refer to: http://wiringpi.com/reference/shift-library/

### Python Code 17.1.1 LightWater02

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 17.1.1_LightWater02 directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/17.1.1_LightWater02
```

2. Use python command to execute Python code "LightWater02.py".

```
python LightWater02.py
```

After the program is executed, you will see that Bar Graph LED starts with the flowing water pattern flashing from left to right and then back from right to left.

The following is the program code:

```
import RPi.GPIO as GPIO
import time
# Defines the data bit that is transmitted preferentially in the shiftOut function.
LSBFIRST = 1
MSBFIRST = 2
# define the pins for 74HC595
dataPin   = 11      # DS Pin of 74HC595(Pin14)
latchPin  = 13      # ST_CP Pin of 74HC595(Pin12)
clockPin = 15     # CH_CP Pin of 74HC595(Pin11)

def setup():
    GPIO.setmode(GPIO.BOARD)     # use PHYSICAL GPIO Numbering
    GPIO.setup(dataPin, GPIO.OUT) # set pin to OUTPUT mode
    GPIO.setup(latchPin, GPIO.OUT)
    GPIO.setup(clockPin, GPIO.OUT)

# shiftOut function, use bit serial transmission.
def shiftOut(dPin,cPin,order,val):
    for i in range(0,8):
        GPIO.output(cPin,GPIO.LOW);
        if(order == LSBFIRST):
            GPIO.output(dPin,(0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
        elif(order == MSBFIRST):
            GPIO.output(dPin,(0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH);

def loop():
    while True:
        x=0x01
        for i in range(0,8):
            GPIO.output(latchPin,GPIO.LOW) # Output low level to latchPin
            shiftOut(dataPin,clockPin,LSBFIRST,x) # Send serial data to 74HC595
            GPIO.output(latchPin,GPIO.HIGH) # Output high level to latchPin, and 74HC595 will
update the data to the parallel output port.
```

```
35              x<<=1 # make the variable move one bit to left once, then the bright LED move one
36      step to the left once.
37              time.sleep(0.1)
38          x=0x80
39          for i in range(0,8):
40              GPIO.output(latchPin,GPIO.LOW)
41              shiftOut(dataPin,clockPin,LSBFIRST,x)
42              GPIO.output(latchPin,GPIO.HIGH)
43              x>>=1
44              time.sleep(0.1)
45
46  def destroy():
47      GPIO.cleanup()
48
49  if __name__ == '__main__': # Program entrance
50      print ('Program is starting...')
51      setup()
52      try:
53          loop()
54      except KeyboardInterrupt:  # Press ctrl-c to end the program.
55          destroy()
```

In the code, we define a shiftOut() function, which is used to output values with bits in order, where the dPin for the data pin, cPin for the clock and order for the priority bit flag (high or low). This function conforms to the operational modes of the 74HC595. LSBFIRST and MSBFIRST are two different flow directions.

```
def shiftOut(dPin,cPin,order,val):
    for i in range(0,8):
        GPIO.output(cPin,GPIO.LOW);
        if(order == LSBFIRST):
            GPIO.output(dPin,(0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
        elif(order == MSBFIRST):
            GPIO.output(dPin,(0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH);
```

In the loop() function, we use two cycles to achieve the action goal. First, define a variable x=0x01, binary 00000001. When it is transferred to the output port of 74HC595, the low bit outputs high level, then an LED turns ON. Next, x is shifted one bit, when x is transferred to the output port of 74HC595 once again, the LED that turns ON will be shifted. Repeat the operation, over and over and the effect of a flowing water light will be visible. If the direction of the shift operation for x is different, the flowing direction is different.

```
def loop():
    while True:
        x=0x01
        for i in range(0,8):
            GPIO.output(latchPin,GPIO.LOW)  #Output low level to latchPin
```

```
            shiftOut(dataPin,clockPin,LSBFIRST,x)#Send serial data to 74HC595
            GPIO.output(latchPin,GPIO.HIGH)#Output high level to latchPin, and 74HC595
will update the data to the parallel output port.
            x<<=1# make the variable move one bit to left once, then the bright LED move
one step to the left once.
            time.sleep(0.1)
        x=0x80
        for i in range(0,8):
            GPIO.output(latchPin,GPIO.LOW)
            shiftOut(dataPin,clockPin,LSBFIRST,x)
            GPIO.output(latchPin,GPIO.HIGH)
            x>>=1
            time.sleep(0.1)
```

# Chapter 18 74HC595 & 7-Segment Display

In this chapter, we will introduce the 7-Segment Display.

## Project 18.1 7-Segment Display

We will use a 74HC595 IC Chip to control a 7-Segment Display and make it display sixteen decimal characters "0" to "F".

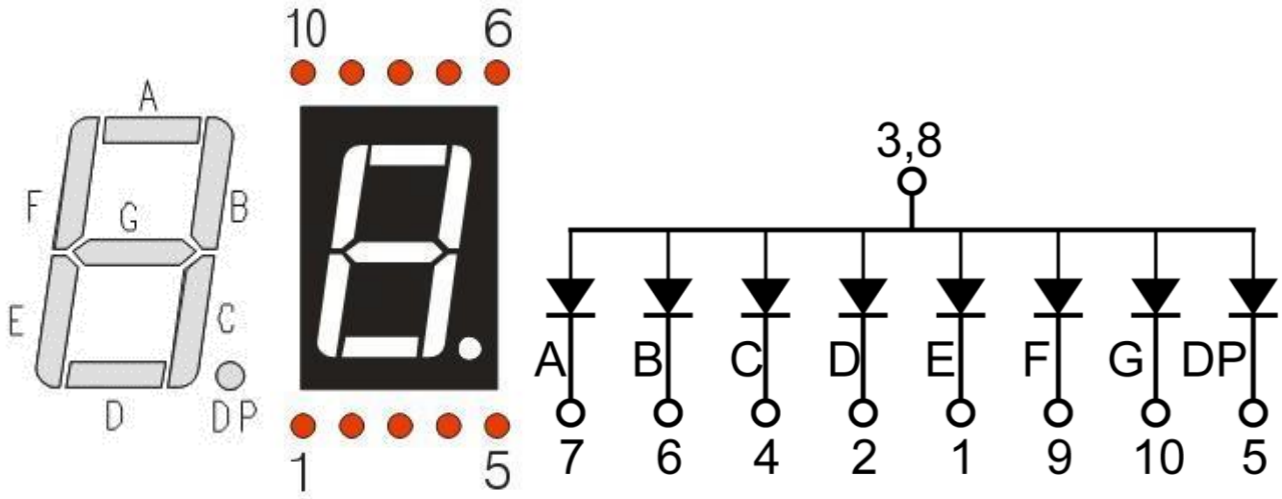
### Component List

| | | |
|---|---|---|
| Raspberry Pi (with 40 GPIO) x1<br>GPIO Extension Board & Ribbon Cable x1<br>Breadboard x1 | Jumper Wire x18<br>  | |
| 74HC595 x1<br>  | 7-Segment Display x1<br>  | Resistor 220Ω x8<br>  |

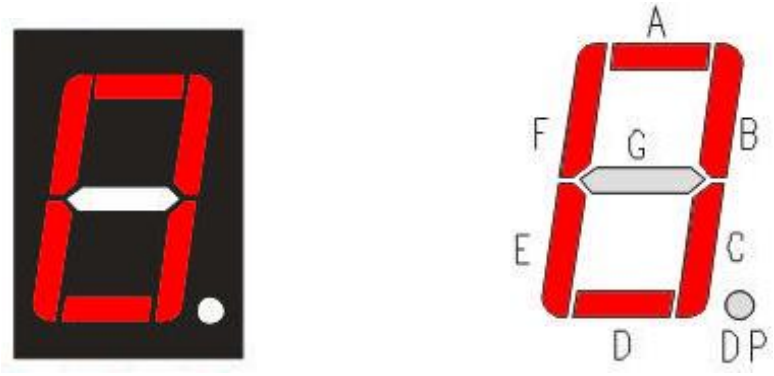
# Component knowledge

### 7-segment display

A 7-Segment Display is a digital electronic display device. There is a figure "8" and a decimal point represented, which consists of 8 LEDs. The LEDs have a Common Anode and individual Cathodes. Its internal structure and pin designation diagram is shown below:



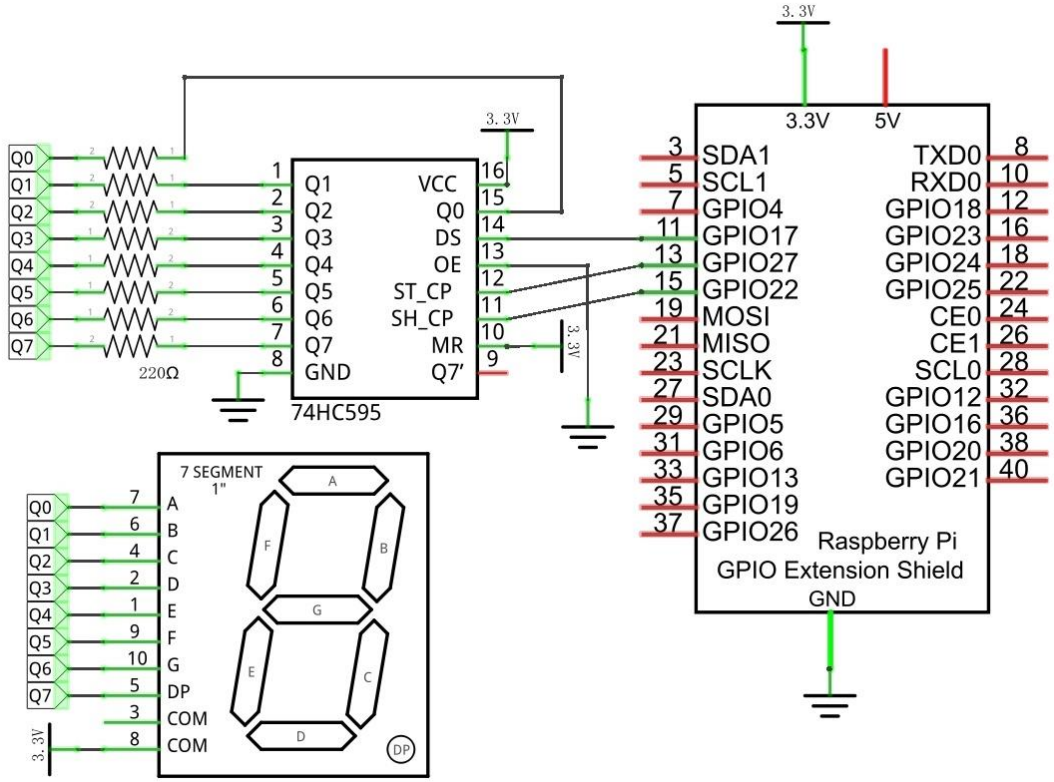

As we can see in the above circuit diagram, we can control the state of each LED separately. Also, by combining LEDs with different states of ON and OFF, we can display different characters (Numbers and Letters). For example, to display a “0”: we need to turn ON LED segments A, B, C, D, E and F, and turn OFF LED segments G and DP.



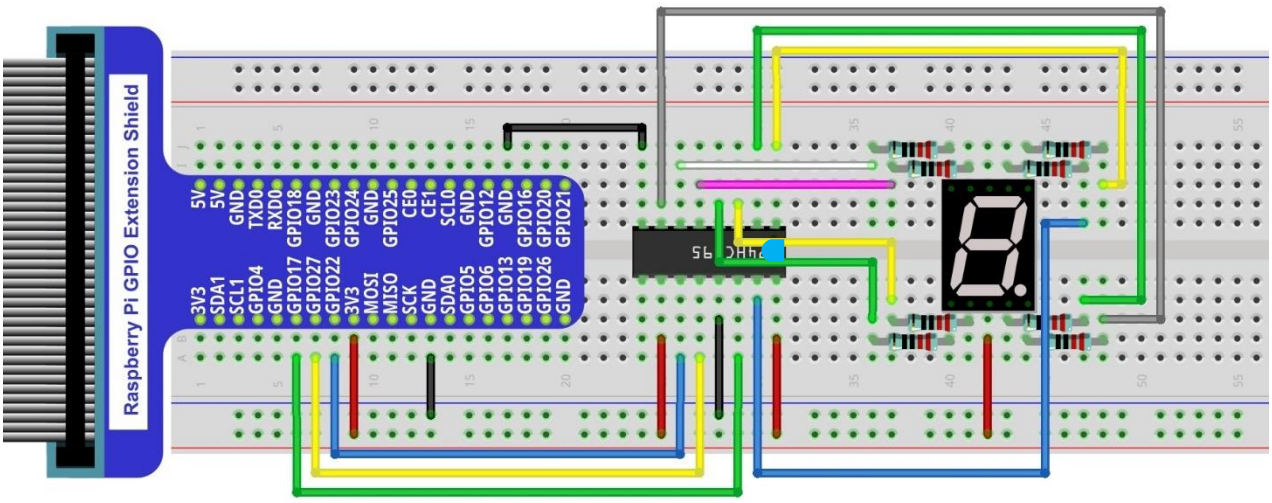

In this project, we will use a 7-Segment Display with a Common Anode. Therefore, when there is an input low level to an LED segment the LED will turn ON. Defining segment “A” as the lowest level and segment “DP” as the highest level, from high to low would look like this: “DP”, “G”, “F”, “E”, “D”, “C”, “B”, “A”. Character "0" corresponds to the code: 1100 0000b=0xc0.

# Circuit

Schematic diagram




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




# Code

This code uses a 74HC595 IC Chip to control the 7-Segment Display. The use of the 74HC595 IC Chip is

generally the same throughout this Tutorial. We need code to display the characters “0” to “F” one character at a time, and then output to display them with the 74HC595 IC Chip.

### C Code 18.1.1 SevenSegmentDisplay

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 18.1.1_SevenSegmentDisplay directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/18.1.1_SevenSegmentDisplay
```

2. Use following command to compile “SevenSegmentDisplay.c” and generate executable file “SevenSegmentDisplay”.

```
gcc SevenSegmentDisplay.c -o SevenSegmentDisplay -lwiringPi
```

3. Then run the generated file “SevenSegmentDisplay”.

```
sudo ./SevenSegmentDisplay
```

After the program is executed, the 7-Segment Display starts to display the characters “0” to “F” in succession.

The following is the program code:

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

#define   dataPin   0   //DS Pin of 74HC595(Pin14)
#define   latchPin  2   //ST_CP Pin of 74HC595(Pin12)
#define   clockPin 3    //CH_CP Pin of 74HC595(Pin11)
//encoding for character 0-F of common anode SevenSegmentDisplay.
unsigned char
num[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e};

void _shiftOut(int dPin,int cPin,int order,int val){
    int i;
    for(i = 0; i < 8; i++){
        digitalWrite(cPin,LOW);
        if(order == LSBFIRST){
            digitalWrite(dPin,((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
            delayMicroseconds(10);
        }
        else {//if(order == MSBFIRST){
            digitalWrite(dPin,((0x80&(val<<i)) == 0x80) ? HIGH : LOW);
            delayMicroseconds(10);
        }
        digitalWrite(cPin,HIGH);
        delayMicroseconds(10);
    }
}

int main(void)
{
```

```
31      int i;
32
33      printf("Program is starting ...\n");
34
35      wiringPiSetup();
36
37      pinMode(dataPin,OUTPUT);
38      pinMode(latchPin,OUTPUT);
39      pinMode(clockPin,OUTPUT);
40      while(1){
41          for(i=0;i<sizeof(num);i++){
42              digitalWrite(latchPin,LOW);
43              _shiftOut(dataPin,clockPin,MSBFIRST,num[i]);//Output the figures and the highest
44  level is transfered preferentially.
45              digitalWrite(latchPin,HIGH);
46              delay(500);
47          }
48          for(i=0;i<sizeof(num);i++){
49              digitalWrite(latchPin,LOW);
50              _shiftOut(dataPin,clockPin,MSBFIRST,num[i] & 0x7f);//Use the "&0x7f" to display
51  the decimal point.
52              digitalWrite(latchPin,HIGH);
53              delay(500);
54          }
55      }
56      return 0;
57  }
```

First, we need to create encoding for characters “0” to “F” in the array.

```
unsigned char
num[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90,0x88,0x83,0xc6,0xa1,0x86,0x8e};
```

In the “for” loop of loop() function, use the 74HC595 IC Chip to output contents of array “num” successively. SevenSegmentDisplay can then correctly display the corresponding characters. Pay attention to this in regard to shiftOut function, the transmission bit, flag bit and highest bit will be transmitted preferentially.

```
for(i=0;i<sizeof(num);i++){
            digitalWrite(latchPin,LOW);
            _shiftOut(dataPin,clockPin,MSBFIRST,num[i]);//Output the figures and the
highest level is transfered preferentially.
            digitalWrite(latchPin,HIGH);
            delay(500);
        }
```

If you want to display the decimal point, make the highest bit of each array “0”, which can be implemented easily by num[i]&0x7f.

```
_shiftOut(dataPin,clockPin,MSBFIRST,num[i] & 0x7f);
```

### Python Code 18.1.1 SevenSegmentDisplay

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 18.1.1_SevenSegmentDisplay directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/18.1.1_SevenSegmentDisplay
```

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

```
python SevenSegmentDisplay.py
```

After the program is executed, the 7-Segment Display starts to display the characters "0" to "F" in succession.

The following is the program code:

```
import RPi.GPIO as GPIO
import time

LSBFIRST = 1
MSBFIRST = 2
#define the pins connect to 74HC595
dataPin   = 11      #DS Pin of 74HC595(Pin14)
latchPin  = 13      #ST_CP Pin of 74HC595(Pin12)
clockPin = 15       #CH_CP Pin of 74HC595(Pin11)
#SevenSegmentDisplay display the character "0"- "F"successively
num = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]
def setup():
    GPIO.setmode(GPIO.BOARD)    # Number GPIOs by its physical location
    GPIO.setup(dataPin, GPIO.OUT)
    GPIO.setup(latchPin, GPIO.OUT)
    GPIO.setup(clockPin, GPIO.OUT)

def shiftOut(dPin,cPin,order,val):
    for i in range(0,8):
        GPIO.output(cPin,GPIO.LOW);
        if(order == LSBFIRST):
            GPIO.output(dPin,(0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
        elif(order == MSBFIRST):
            GPIO.output(dPin,(0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH);

def loop():
    while True:
        for i in range(0,len(num)):
            GPIO.output(latchPin,GPIO.LOW)
            shiftOut(dataPin,clockPin,MSBFIRST,num[i])#Output the figures and the highest
level is transfered preferentially.
            GPIO.output(latchPin,GPIO.HIGH)
            time.sleep(0.5)
        for i in range(0,len(num)):
```

```
            GPIO.output(latchPin,GPIO.LOW)
            shiftOut(dataPin,clockPin,MSBFIRST,num[i]&0x7f)#Use "&0x7f"to display the
decimal point.
            GPIO.output(latchPin,GPIO.HIGH)
            time.sleep(0.5)

def destroy():
    GPIO.cleanup()

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

First, we need to create encoding for characters "0" to "F" in the array.

```
num = [0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90, 0x88, 0x83, 0xc6, 0xa1, 0x86, 0x8e]
```

In the "for" loop of loop() function, use the 74HC595 IC Chip to output contents of array "num" successively. SevenSegmentDisplay can then correctly display the corresponding characters. Pay attention to this in regard to shiftOut function, the transmission bit, flag bit amd highest bit will be transmitted preferentially.

```
        for i in range(0,len(num)):
            GPIO.output(latchPin,GPIO.LOW)
            shiftOut(dataPin,clockPin,MSBFIRST,num[i])#Output the figures and the highest
level is transfered preferentially.
            GPIO.output(latchPin,GPIO.HIGH)
            time.sleep(0.5)
```

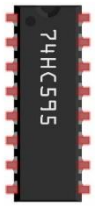
If you want to display the decimal point, make the highest bit of each array "0", which can be implemented easily by num[i]&0x7f.

```
shiftOut(dataPin,clockPin,MSBFIRST,num[i]&0x7f)# Use  "&0x7f" to display the decimal
point.
```

# Project 18.2 4-Digit 7-Segment Display

Now, let's try to control more-than-one digit displays by using a Four 7-Segment Display in one project.

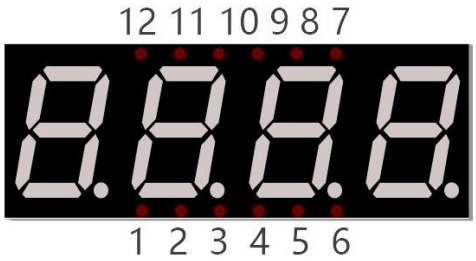
## Component List

<table>
<tr><td colspan="3">Raspberry Pi (with 40 GPIO) x1<br>GPIO Expansion Board & Wire x1<br>Breadboard x1</td><td colspan="2">Jumper Wire x30<br>  </td></tr>
<tr><td>74HC595 x1<br>  </td><td>PNP transistor x4<br>  </td><td>4-Digit 7-Segment Display x1<br>  </td><td>Resistor 220Ω x8<br>  </td><td>Resistor 1KΩ x4<br>  </td></tr>
</table>

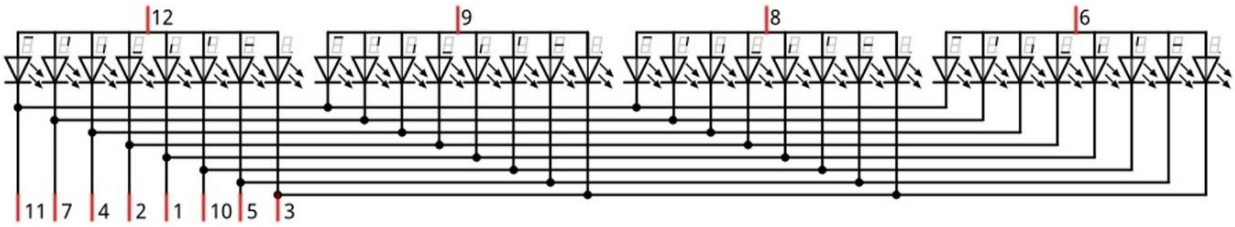
# Component knowledge

### 4 Digit 7-Segment Display

A 4 Digit 7-segment display integrates four 7-Segment Displays into one module, therefore it can display more characters. All of the LEDs contained have a Common Anode and individual Cathodes. Its internal structure and pin designation diagram is shown below:




The internal electronic circuit is shown below, and all 8 LED cathode pins of each 7-Segment Display are connected together.

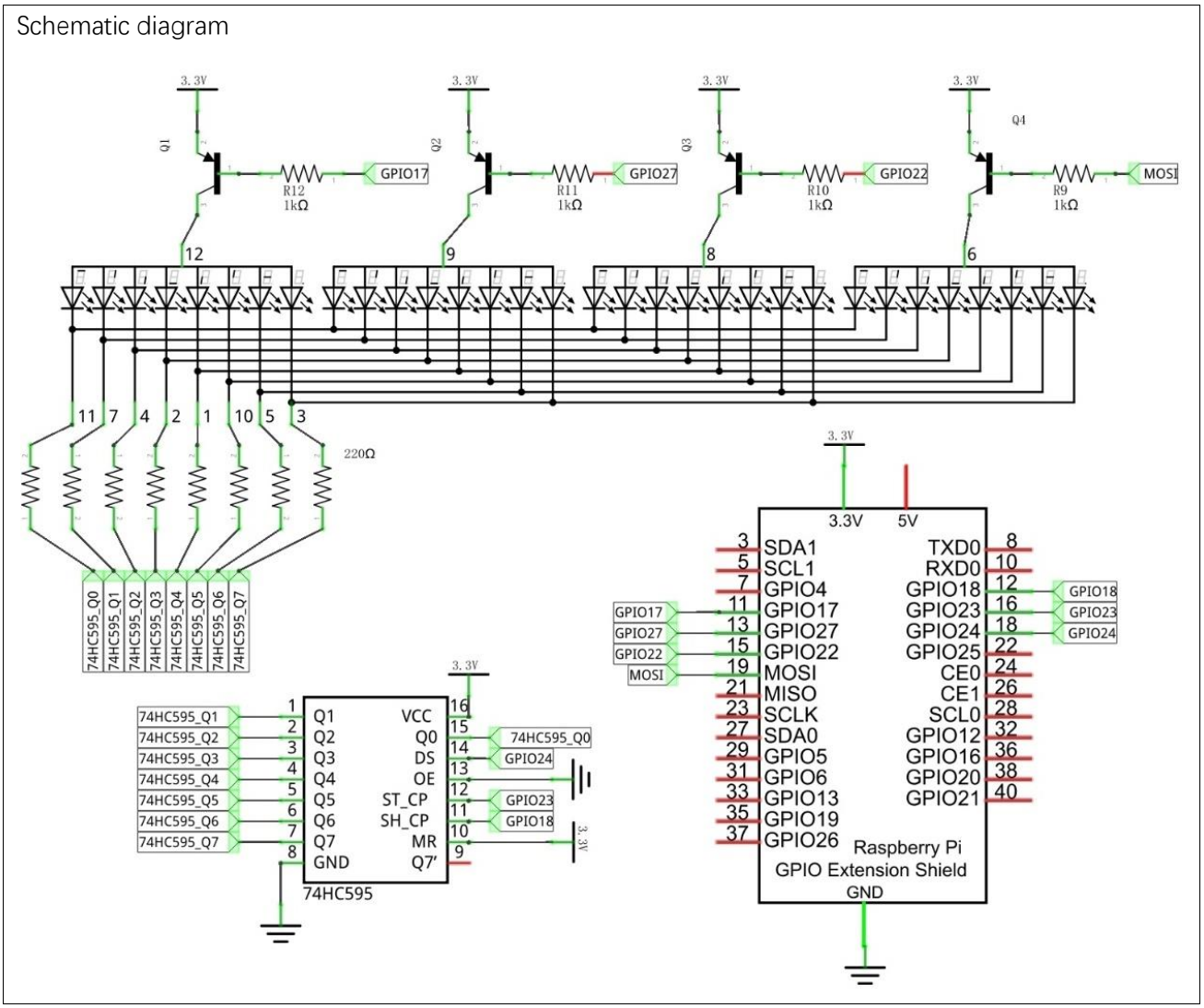



Display method of 4 Digit 7-segment display is similar to 1 Digit 7-segment display. The difference between them is that the 4-Digit displays each Digit is visible in turn, one by one and not together. We need to first send high level to the common end of the first Digit Display, and send low level to the remaining three common ends, and then send content to 8 LED cathode pins of the first Digit Display. At this time, the first 7-Segment Display will show visible content and the remaining three will be OFF.

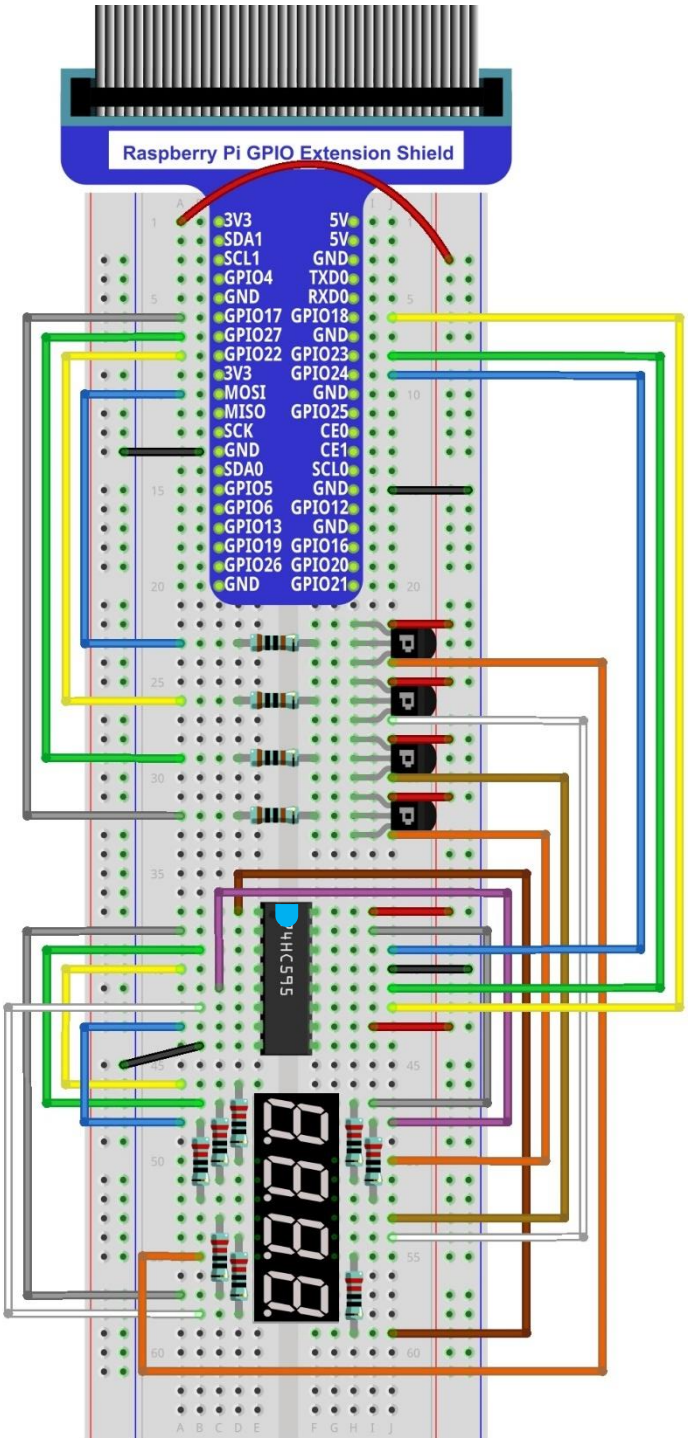
Similarly, the second, third and fourth 7-Segment Displays will show visible content in turn by scanning the display. Although the four number characters are displayed in turn separately, this process is so very fast that it is unperceivable to the naked eye. This is due to the principle of optical afterglow effect and the vision persistence effect in human sight. This is how we can see all 4 number characters at the same time. However, if each number character is displayed for a longer period, you will be able to see that the number characters are displayed separately.

# Circuit

Schematic diagram



Hardware connection



# Code

In this code, we use the 74HC595 IC Chip to control the 4-Digit 7-Segment Display, and use the dynamic scanning method to show the changing number characters.

C Code 18.2.1 StopWatch

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 16.1.1_SteppingMotor directory of C code.

```
cd ~/Freenove_Kit/Code/C_Code/18.2.1_StopWatch
```

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

```
gcc StopWatch.c -o StopWatch -lwiringPi
```

3. Run the generated file "SteppingMotor".

```
sudo ./StopWatch
```

After the program is executed, the 4-Digit 7-Segment Display starts displaying a four-digit number dynamically, and the numeric value of this number will increase by plus 1 each second thereafter.

The following is the program code:

```
#include <wiringPi.h>
#include <stdio.h>
#include <wiringShift.h>
#include <signal.h>
#include <unistd.h>
#define     dataPin     5   //DS Pin of 74HC595(Pin14)
#define     latchPin    4   //ST_CP Pin of 74HC595(Pin12)
#define     clockPin    1    //CH_CP Pin of 74HC595(Pin11)
const int digitPin[]={0,2,3,12};        // Define 7-segment display common pin
// character 0-9 code of common anode 7-segment display
unsigned char num[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
int counter = 0;    //variable counter,the number will be displayed by 7-segment display
//Open one of the 7-segment display and close the remaining three, the parameter digit is
optional for 1,2,4,8
void selectDigit(int digit){
    digitalWrite(digitPin[0],((digit&0x08) == 0x08) ? LOW : HIGH);
    digitalWrite(digitPin[1],((digit&0x04) == 0x04) ? LOW : HIGH);
    digitalWrite(digitPin[2],((digit&0x02) == 0x02) ? LOW : HIGH);
    digitalWrite(digitPin[3],((digit&0x01) == 0x01) ? LOW : HIGH);
}
void _shiftOut(int dPin,int cPin,int order,int val){
    int i;
    for(i = 0; i < 8; i++){
        digitalWrite(cPin,LOW);
        if(order == LSBFIRST){
            digitalWrite(dPin,((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
            delayMicroseconds(1);
```

```
28              }
29          else {//if(order == MSBFIRST){
30              digitalWrite(dPin,((0x80&(val<<i)) == 0x80) ? HIGH : LOW);
31              delayMicroseconds(1);
32              }
33          digitalWrite(cPin,HIGH);
34          delayMicroseconds(1);
35      }
36  }
37  void outData(int8_t data){      //function used to output data for 74HC595
38      digitalWrite(latchPin,LOW);
39      _shiftOut(dataPin,clockPin,MSBFIRST,data);
40      digitalWrite(latchPin,HIGH);
41  }
42  void display(int dec){  //display function for 7-segment display
43      int delays = 1;
44      outData(0xff);
45      selectDigit(0x01);      //select the first, and display the single digit
46      outData(num[dec%10]);
47      delay(delays);          //display duration
48
49      outData(0xff);
50      selectDigit(0x02);      //select the second, and display the tens digit
51      outData(num[dec%100/10]);
52      delay(delays);
53
54      outData(0xff);
55      selectDigit(0x04);      //select the third, and display the hundreds digit
56      outData(num[dec%1000/100]);
57      delay(delays);
58
59      outData(0xff);
60      selectDigit(0x08);      //select the fourth, and display the thousands digit
61      outData(num[dec%10000/1000]);
62      delay(delays);
63  }
64  void timer(int  sig){       //Timer function
65      if(sig == SIGALRM){   //If the signal is SIGALRM, the value of counter plus 1, and update
66  the number displayed by 7-segment display
67          counter ++;
68          alarm(1);           //set the next timer time
69          printf("counter : %d \n",counter);
70      }
71  }
```

```
72  int main(void)
73  {
74      int i;
75
76      printf("Program is starting ...\n");
77
78      wiringPiSetup();
79
80      pinMode(dataPin,OUTPUT);        //set the pin connected to74HC595 for output mode
81      pinMode(latchPin,OUTPUT);
82      pinMode(clockPin,OUTPUT);
83      //set the pin connected to 7-segment display common end to output mode
84      for(i=0;i<4;i++){
85          pinMode(digitPin[i],OUTPUT);
86          digitalWrite(digitPin[i],HIGH);
87      }
88      signal(SIGALRM,timer);  //configure the timer
89      alarm(1);               //set the time of timer to 1s
90      while(1){
91          display(counter);   //display the number counter
92      }
93      return 0;
94  }
```

First, we define the pin of the 74HC595 IC Chip and the 7-Segment Display Common Anode, use character encoding and a variable "counter" to enable the counter to be visible on the 7-Segment Display.

```
#define     dataPin     5   //DS Pin of 74HC595(Pin14)
#define     latchPin    4   //ST_CP Pin of 74HC595(Pin12)
#define     clockPin    1    //CH_CP Pin of 74HC595(Pin11)
const int digitPin[]={0,2,3,12};   //Define the pin of 7-segment display common end
// character 0-9 code of common anode 7-segment display
unsigned char num[]={0xc0,0xf9,0xa4,0xb0,0x99,0x92,0x82,0xf8,0x80,0x90};
int counter = 0;    //variable counter, the number will be displayed by 7-segment display
```

Subfunction **selectDigit** (int digit) function is used to open one of the 7-Segment Displays while closing the other 7-Segment Displays, where the parameter digit value can be 1,2,4,8. Using "|" can open a number of a 7-Segment Display.

```
void selectDigit(int digit){
    digitalWrite(digitPin[0],((digit&0x08) == 0x08) ? LOW : HIGH);
    digitalWrite(digitPin[1],((digit&0x04) == 0x04) ? LOW : HIGH);
    digitalWrite(digitPin[2],((digit&0x02) == 0x02) ? LOW : HIGH);
    digitalWrite(digitPin[3],((digit&0x01) == 0x01) ? LOW : HIGH);
}
```

Subfunction **outData** (int8_t data) is used to make the 74HC595 IC Chip output an 8-bit data immediately.

```
void outData(int8_t data){      // function used to output data for 74HC595
    digitalWrite(latchPin,LOW);
    shiftOut(dataPin,clockPin,MSBFIRST,data);
    digitalWrite(latchPin,HIGH);
}
```

Subfunction **display** (int dec) is used to make a 4-Digit 7-Segment Display a 4-bit integer. First open the common end of first 7-Segment Display Digit and turn OFF the other three Digits, now it can be used as 1-Digit 7-Segment Display. The first Digit is used for displaying single digits of "dec", the second Digit is for tens, the third for hundreds and fourth for thousands respectively. Each digit will be displayed for a period by using **delay** (). The time in this code is very brief, so you will see digits all together. If the time is set long enough, you will see that every digit is displayed independently.

```
void display(int dec){  //display function for 7-segment display
    selectDigit(0x01);      //select the first, and display the single digit
    outData(num[dec%10]);
    delay(1);               //display duration
    selectDigit(0x02);      //Select the second, and display the tens digit
    outData(num[dec%100/10]);
    delay(1);
    selectDigit(0x04);      //Select the third, and display the hundreds digit
    outData(num[dec%1000/100]);
    delay(1);
    selectDigit(0x08);      //Select the fourth, and display the thousands digit
    outData(num[dec%10000/1000]);
    delay(1);
}
```

Subfunction **timer** (int sig) is the timer function, which will set an alarm to signal. This function will be executed once at set time intervals. Accompanied by the execution, “1” will be added as the variable counter and then reset the time of timer to 1s.

```
void timer(int  sig){       //timer function
    if(sig == SIGALRM){   //If the signal is SIGALRM, the value of counter plus 1, and
update the number displayed by 7-segment display
        counter ++;
        alarm(1);           //set the next timer time
    }
}
```

Finally, in the main function, configure the GPIO, and set the timer function.

```
    pinMode(dataPin,OUTPUT);        //set the pin connected to74HC595 for output mode
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    //set the pin connected to 7-segment display common end to output mode
    for(i=0;i<4;i++){
        pinMode(digitPin[i],OUTPUT);
        digitalWrite(digitPin[i],LOW);
    }
    signal(SIGALRM,timer);  //configure the timer
    alarm(1);               //set the time of timer to 1s
```

In the while loop, make the digital display variable counter value “1”. The value will change in function timer (), so the content displayed by the 7-Segment Display will change accordingly.

```
    while(1){
        display(counter);   //display number counter
    }
```

### Python Code 18.2.1 StopWatch

This code uses the four step four pat mode to drive the Stepper Motor clockwise and reverse direction.

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

1. Use cd command to enter 16.1.1_SteppingMotor directory of Python code.

```
cd ~/Freenove_Kit/Code/Python_Code/18.2.1_StopWatch
```

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

```
python StopWatch.py
```

After the program is executed, 4-Digit 7-segment start displaying a four-digit number dynamically, and the will plus 1 in each successive second.

The following is the program code:

```
import RPi.GPIO as GPIO
import time
import threading

LSBFIRST = 1
MSBFIRST = 2
#define the pins connect to 74HC595
dataPin   = 18      #DS Pin of 74HC595(Pin14)
latchPin  = 16      #ST_CP Pin of 74HC595(Pin12)
clockPin = 12       #SH_CP Pin of 74HC595(Pin11)
num = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)
digitPin = (11, 13, 15, 19)    # Define the pin of 7-segment display common end
counter = 0         # Variable counter, the number will be dislayed by 7-segment display
t = 0               # define the Timer object
def setup():
    GPIO.setmode(GPIO.BOARD)    # Number GPIOs by its physical location
    GPIO.setup(dataPin, GPIO.OUT)       # Set pin mode to output
    GPIO.setup(latchPin, GPIO.OUT)
    GPIO.setup(clockPin, GPIO.OUT)
    for pin in digitPin:
        GPIO.setup(pin,GPIO.OUT)

def shiftOut(dPin,cPin,order,val):
    for i in range(0,8):
        GPIO.output
(cPin,GPIO.LOW);
        if(order == LSBFIRST):
            GPIO.output(dPin,(0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
        elif(order == MSBFIRST):
            GPIO.output(dPin,(0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH)

def outData(data):      #function used to output data for 74HC595
    GPIO.output(latchPin,GPIO.LOW)
```

```
    shiftOut(dataPin,clockPin,MSBFIRST,data)
    GPIO.output(latchPin,GPIO.HIGH)

def selectDigit(digit): # Open one of the 7-segment display and close the remaining
three, the parameter digit is optional for 1,2,4,8
    GPIO.output(digitPin[0],GPIO.LOW if ((digit&0x08) == 0x08) else GPIO.HIGH)
    GPIO.output(digitPin[1],GPIO.LOW if ((digit&0x04) == 0x04) else GPIO.HIGH)
    GPIO.output(digitPin[2],GPIO.LOW if ((digit&0x02) == 0x02) else GPIO.HIGH)
    GPIO.output(digitPin[3],GPIO.LOW if ((digit&0x01) == 0x01) else GPIO.HIGH)

def display(dec):   #display function for 7-segment display
    outData(0xff)   #eliminate residual display
    selectDigit(0x01)   #Select the first, and display the single digit
    outData(num[dec%10])
    time.sleep(0.003)   #display duration
    outData(0xff)
    selectDigit(0x02)   # Select the second, and display the tens digit
    outData(num[dec%100//10])
    time.sleep(0.003)
    outData(0xff)
    selectDigit(0x04)   # Select the third, and display the hundreds digit
    outData(num[dec%1000//100])
    time.sleep(0.003)
    outData(0xff)
    selectDigit(0x08)   # Select the fourth, and display the thousands digit
    outData(num[dec%10000//1000])
    time.sleep(0.003)
def timer():        #timer function
    global counter
    global t
    t = threading.Timer(1.0,timer)      #reset time of timer to 1s
    t.start()                           #Start timing
    counter+=1
    print ("counter : %d"%counter)

def loop():
    global t
    global counter
    t = threading.Timer(1.0,timer)      #set the timer
    t.start()                           # Start timing
    while True:
        display(counter)                # display the number counter

def destroy():
```

```
79      global t
80      GPIO.cleanup()
81      t.cancel()      #cancel the timer
82
83  if __name__ == '__main__':
84      print ('Program is starting...' )
85      setup()
86      try:
87          loop()
89      except KeyboardInterrupt:
            destroy()
```

First, define the pin of 74HC595 and 7-segment display common end, character encoding and a variable "counter" to be displayed counter.

```
dataPin   = 18      #DS Pin of 74HC595(Pin14)
latchPin  = 16      #ST_CP Pin of 74HC595(Pin12)
clockPin = 12       #CH_CP Pin of 74HC595(Pin11)
num = (0xc0, 0xf9, 0xa4, 0xb0, 0x99, 0x92, 0x82, 0xf8, 0x80, 0x90)
digitPin = (11, 13, 15, 19)     # Define the pin of 7-segment display common end
counter = 0         # Variable counter, the number will be displayed by 7-segment display
```

Subfunction **selectDigit** (digit) function is used to open one of the 7-segment display and close the other 7-segment display, where the parameter digit value can be 1,2,4,8. Using "|" can open a number of 7-segment display.

```
def selectDigit(digit): #Open one of the 7-segment display and close the remaining three,
the parameter digit is optional for 1,2,4,8
    GPIO.output(digitPin[0],GPIO.LOW if ((digit&0x08) == 0x08) else GPIO.HIGH)
    GPIO.output(digitPin[1],GPIO.LOW if ((digit&0x04) == 0x04) else GPIO.HIGH)
    GPIO.output(digitPin[2],GPIO.LOW if ((digit&0x02) == 0x02) else GPIO.HIGH)
    GPIO.output(digitPin[3],GPIO.LOW if ((digit&0x01) == 0x01) else GPIO.HIGH)
```

Subfunction **outData** (data) is used to make the 74HC595 output an 8-bit data immediately.

```
def outData(data):      #function used to output data for 74HC595
    GPIO.output(latchPin,GPIO.LOW)
    shiftOut(dataPin,clockPin,MSBFIRST,data)
    GPIO.output(latchPin,GPIO.HIGH)
```

Subfunction **display** (int dec) is used to make a 4-Digit 7-Segment Display a 4-bit integer. First open the common end of first 7-Segment Display Digit and turn OFF the other three Digits, now it can be used as 1-Digit 7-Segment Display. The first Digit is used for displaying single digits of "dec", the second Digit is for tens, the third for hundreds and fourth for thousands respectively. Each digit will be displayed for a period by using **delay** (). The time in this code is very brief, so you will a mess of Digits. If the time is set long enough, you will see that every digit is displayed independently.

```
def display(dec):   #display function for 7-segment display
    outData(0xff)   #eliminate residual display
    selectDigit(0x01)   #Select the first, and display the single digit
    outData(num[dec%10])
    time.sleep(0.003)   #display duration
```

```
    outData(0xff)
    selectDigit(0x02)    #Select the second, and display the tens digit
    outData(num[dec%100/10])
    time.sleep(0.003)
    outData(0xff)
    selectDigit(0x04)    #Select the third, and display the hundreds digit
    outData(num[dec%1000/100])
    time.sleep(0.003)
    outData(0xff)
    selectDigit(0x08)    #Select the fourth, and display the thousands digit
    outData(num[dec%10000/1000])
    time.sleep(0.003)
```

Subfunction **timer** () is the timer callback function. When the time is up, this function will be executed. Accompanied by the execution, the variable counter will be added 1, and then reset the time of timer to 1s. 1s later, the function will be executed again.

```
def timer():        #timer function
    global counter
    global t
    t = threading.Timer(1.0,timer)      #reset time of timer to 1s
    t.start()                           #Start timing
    counter+=1
    print ("counter : %d"%counter)
```

Subfunction **setup**(), configure all input output modes for the GPIO pin used.

Finally, in loop function, make the digital tube display variable counter value in the while loop. The value will change in function **timer** (), so the content displayed by 7-segment display will change accordingly.

```
def loop():
    global t
    global counter
    t = threading.Timer(1.0,timer)      # set the timer
    t.start()                           #Start timing
    while True:
        display(counter)                #display the number counter
```

After the program is executed, press "Ctrl+C", then subfunction destroy() will be executed, and GPIO resources and timers will be released in this subfunction.

```
def destroy():   # When 'Ctrl+C' is pressed, the function is executed.
    global t
    GPIO.cleanup()
    t.cancel()      # cancel the timer
```

# Chapter 19 74HC595 & LED Matrix

Thus far we have learned how to use the 74HC595 IC Chip to control the Bar Graph LED and the 7-Segment Display. We will now use 74HC595 IC Chips to control an LED Matrix.

## Project 19.1 LED Matrix

In this project, we will use two 74HC595 IC chips to control a monochrome (one color) (8X8) LED Matrix to make it display both simple graphics and characters.

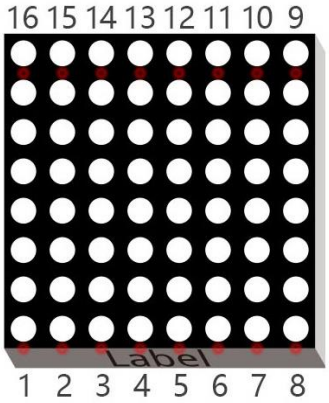
### Component List

<table>
<tr><td colspan="2">Raspberry Pi (with 40 GPIO) x1<br>GPIO Extension Board & Ribbon Cable x1<br>Breadboard x1</td><td>Jumper x36<br>  </td></tr>
<tr><td>74HC595 x2<br>  </td><td>8X8 LEDMatrix x1<br>  </td><td>Resistor 220Ω x8<br>  </td></tr>
</table>

# Component knowledge

## LED matrix

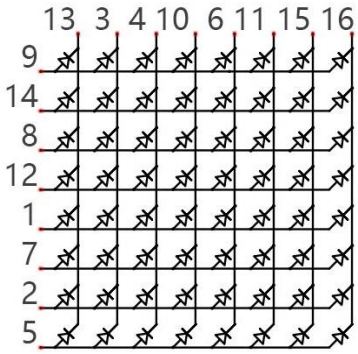
An LED Matrix is a rectangular display module that consists of a uniform grid of LEDs. The following is an 8X8 monochrome (one color) LED Matrix containing 64 LEDs (8 rows by 8 columns).



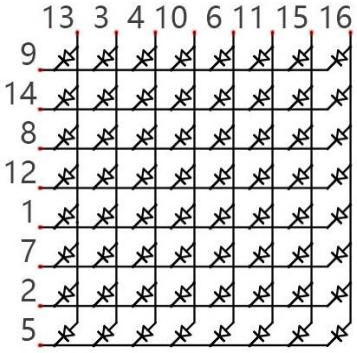

In order to facilitate the operation and reduce the number of ports required to drive this component, the Positive Poles of the LEDs in each row and Negative Poles of the LEDs in each column are respectively connected together inside the LED Matrix module, which is called a Common Anode. There is another arrangement type. Negative Poles of the LEDs in each row and the Positive Poles of the LEDs in each column are respectively connected together, which is called a Common Cathode.

The LED Matrix that we use in this project is a Common Anode LED Matrix.

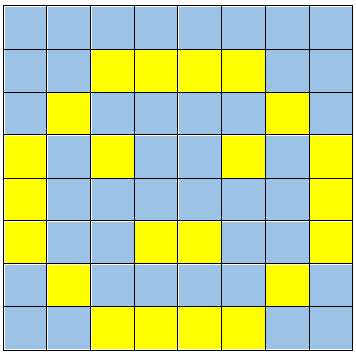
Connection mode of Common Anode




Connection mode of Common Cathode



Here is how a Common Anode LED Matrix works. First, choose 16 ports on RPI board to connect to the 16 ports of LED Matrix. Configure one port in columns for low level, which makes that column the selected port. Then configure the eight port in the row to display content in the selected column. Add a delay value and then select the next column that outputs the corresponding content. This kind of operation by column is called Scan. If you want to display the following image of a smiling face, you can display it in 8 columns, and each column is represented by one byte.



| 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 |

| Column | Binary | Hexadecimal |
|---|---|---|
| 1 | 0001 1100 | 0x1c |
| 2 | 0010 0010 | 0x22 |
| 3 | 0101 0001 | 0x51 |
| 4 | 0100 0101 | 0x45 |
| 5 | 0100 0101 | 0x45 |
| 6 | 0101 0001 | 0x51 |
| 7 | 0010 0010 | 0x22 |
| 8 | 0001 1100 | 0x1c |

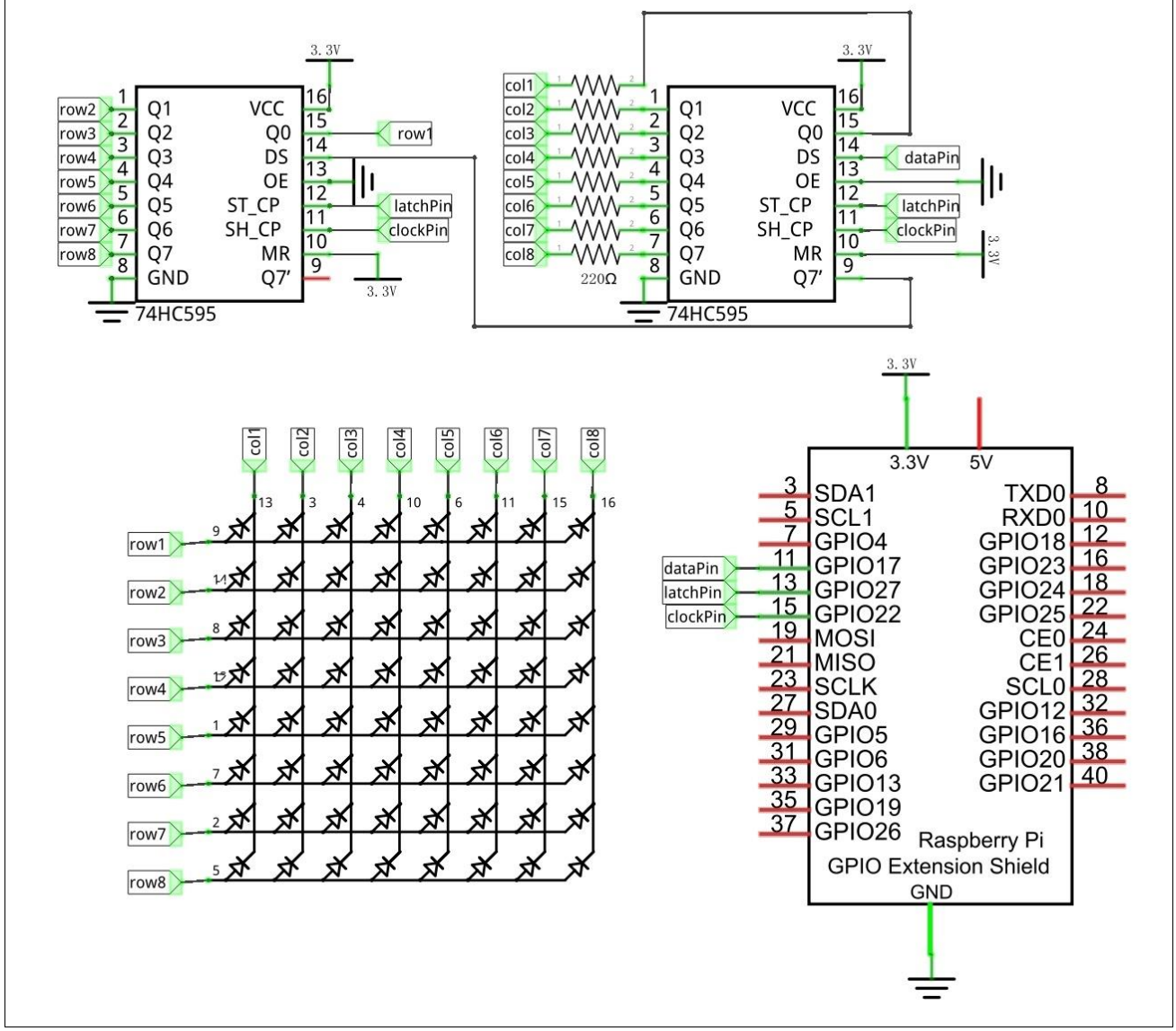
To begin, display the first column, then turn off the first column and display the second column. (and so on) .... turn off the seventh column and display the 8th column, and then start the process over from the first column again like the control of LED Bar Graph project. The whole process will be repeated rapidly in a loop. Due to the principle of optical afterglow effect and the vision persistence effect in human sight, we will see a picture of a smiling face directly rather than individual columns of LEDs turned ON one column at a time (although in fact this is the reality we cannot perceive).

Scanning rows is another option to display on an LED Matrix (dot matrix grid). Whether scanning by row or column, 16 GPIO is required. In order to save GPIO ports of control board, two 74HC595 IC Chips are used in the circuit. Every 74HC595 IC Chip has eight parallel output ports, so two of these have a combined total of 16 ports, which is just enough for our project. The control lines and data lines of the two 74HC595 IC Chips are not all connected to the RPi, but connect to the Q7 pin of first stage 74HC595 IC Chip and to the data pin of second IC Chip. The two 74HC595 IC Chips are connected in series, which is the same as using one "74HC595 IC Chip" with 16 parallel output ports.

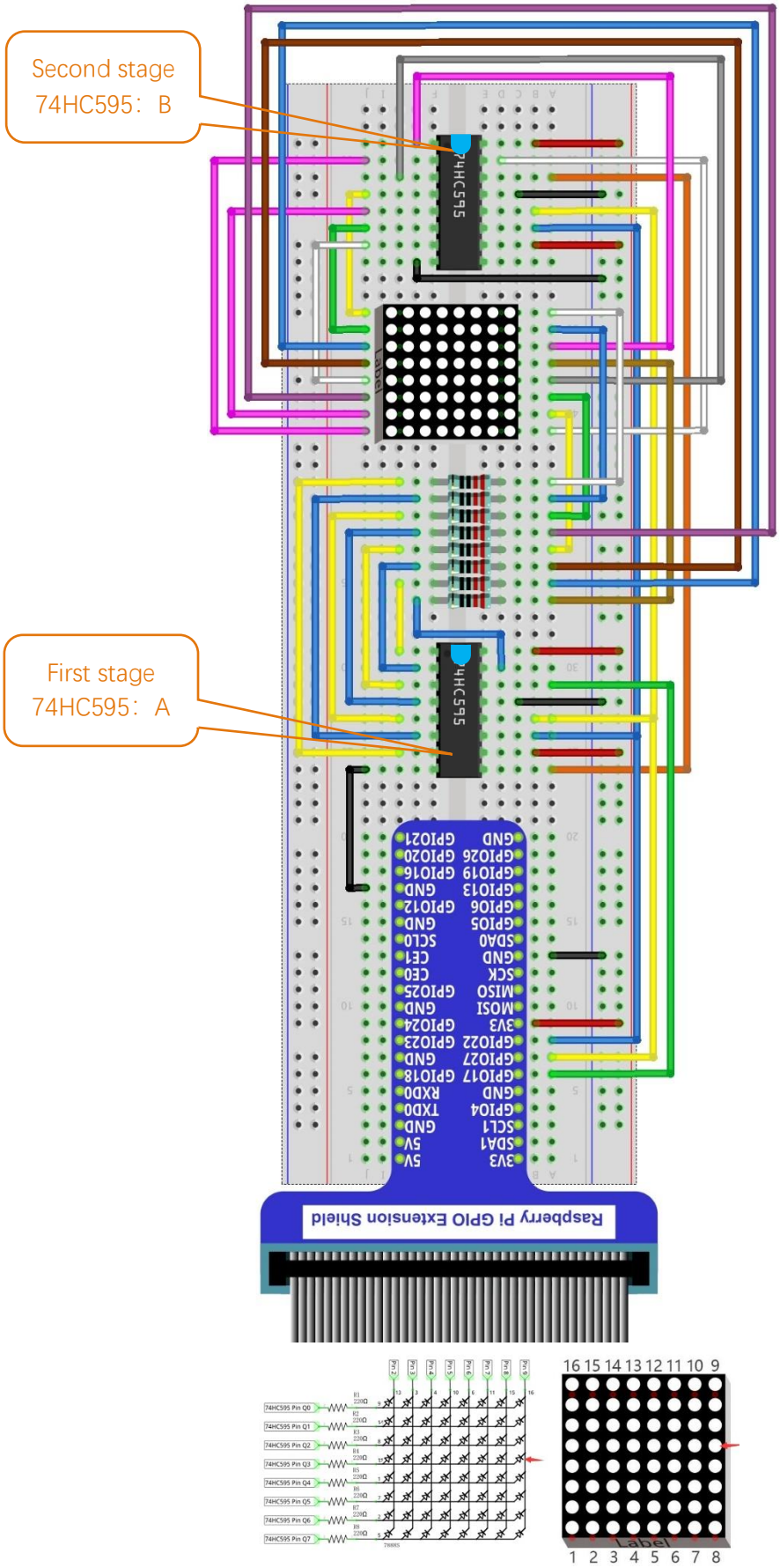
# Circuit

In circuit of this project, the power pin of the 74HC595 IC Chip is connected to 3.3V. It can also be connected to 5V to make LED Matrix brighter.

Schematic diagram



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



# Code

Two 74HC595 IC Chips are used in this project, one for controlling the LED Matrix's columns and the other for controlling the rows. According to the circuit connection, row data should be sent first, then column data. The following code will make the LED Matrix display a smiling face, and then display characters "0 to F" scrolling in a loop on the LED Matrix.

### C Code 19.1.1 LEDMatrix

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 19.1.1_LEDMatrix directory of C language.

```
cd ~/Freenove_Kit/Code/C_Code/19.1.1_LEDMatrix
```

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

```
gcc LEDMatrix.c -o LEDMatrix -lwiringPi
```

3. Then run the generated file "LEDMatrix".

```
sudo ./LEDMatrix
```

After the program is executed, the LED Matrix display a smiling face, and then display characters "0 to F" scrolling in a loop on the LED Matrix.

The following is the program code:

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

#define   dataPin   0   //DS Pin of 74HC595(Pin14)
#define   latchPin  2   //ST_CP Pin of 74HC595(Pin12)
#define   clockPin 3    //SH_CP Pin of 74HC595(Pin11)
// data of smile face
unsigned char pic[]={0x1c,0x22,0x51,0x45,0x45,0x51,0x22,0x1c};
unsigned char data[]={  // data of "0-F"
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // " "
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, // "0"
    0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, // "1"
    0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, // "2"
    0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, // "3"
    0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, // "4"
    0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, // "5"
    0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, // "6"
    0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, // "7"
    0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, // "8"
    0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, // "9"
    0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, // "A"
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, // "B"
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, // "C"
    0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, // "D"
```

```
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, // "E"
    0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00, // "F"
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, // " "
};
void _shiftOut(int dPin,int cPin,int order,int val){
    int i;
    for(i = 0; i < 8; i++){
        digitalWrite(cPin,LOW);
        if(order == LSBFIRST){
            digitalWrite(dPin,((0x01&(val>>i)) == 0x01) ? HIGH : LOW);
            delayMicroseconds(10);
        }
        else {//if(order == MSBFIRST){
            digitalWrite(dPin,((0x80&(val<<i)) == 0x80) ? HIGH : LOW);
            delayMicroseconds(10);
        }
        digitalWrite(cPin,HIGH);
        delayMicroseconds(10);
    }
}
int main(void)
{
    int i,j,k;
    unsigned char x;

    printf("Program is starting ...\n");

    wiringPiSetup();

    pinMode(dataPin,OUTPUT);
    pinMode(latchPin,OUTPUT);
    pinMode(clockPin,OUTPUT);
    while(1){
        for(j=0;j<500;j++){  //Repeat enough times to display the smiling face a period of
time
            x=0x80;
            for(i=0;i<8;i++){
                digitalWrite(latchPin,LOW);
                _shiftOut(dataPin,clockPin,MSBFIRST,pic[i]);// first shift data of line
information to the first stage 74HC959
                _shiftOut(dataPin,clockPin,MSBFIRST,~x);//then shift data of column
information to the second stage 74HC959

                digitalWrite(latchPin,HIGH);//Output data of two stage 74HC595 at the same
```

```
70  time
71                  x>>=1;   //display the next column
72                  delay(1);
73              }
74          }
76          for(k=0;k<sizeof(data)-8;k++){  //sizeof(data) total number of "0-F" columns
77              for(j=0;j<20;j++){  //times of repeated displaying LEDMatrix in every frame, the
78  bigger the "j", the longer the display time
79                  x=0x80;          //Set the column information to start from the first column
80                  for(i=k;i<8+k;i++){
81                      digitalWrite(latchPin,LOW);
82                      _shiftOut(dataPin,clockPin,MSBFIRST,data[i]);
83                      _shiftOut(dataPin,clockPin,MSBFIRST,~x);
84                      digitalWrite(latchPin,HIGH);
85                      x>>=1;
86                      delay(1);
87                  }
88              }
89          }
90      }
91      return 0;
92  }
```

The first "for" loop in the "while" loop is used to display a static smile. Displaying column information from left to right, one column at a time with a total of 8 columns. This repeats 500 times to ensure sufficient display time.

```
            for(j=0;j<500;j++){// Repeat enough times to display the smiling face a period
of time
              x=0x80;
              for(i=0;i<8;i++){
                  digitalWrite(latchPin,LOW);
                  shiftOut(dataPin,clockPin,MSBFIRST,pic[i]);
                  shiftOut(dataPin,clockPin,MSBFIRST,~x);
                  digitalWrite(latchPin,HIGH);
                  x>>=1;
                  delay(1);
              }
          }
```

The second “for” loop is used to display scrolling characters "0 to F", for a total of 18 X 8 = 144 columns. Displaying the 0-8 column, then the 1-9 column, then the 2-10 column..... and so on…138-144 column in consecutively to achieve the scrolling effect. The display of each frame is repeated a certain number of times and the more repetitions, the longer the single frame display will be and the slower the scrolling movement.

```
        for(k=0;k<sizeof(data)-8;k++){  //sizeof(data) total number of "0-F" columns
            for(j=0;j<20;j++){// times of repeated displaying LEDMatrix in every frame,
the bigger the “j”, the longer the display time
                x=0x80;      // Set the column information to start from the first column
                for(i=k;i<8+k;i++){
                    digitalWrite(latchPin,LOW);
                    shiftOut(dataPin,clockPin,MSBFIRST,data[i]);
                    shiftOut(dataPin,clockPin,MSBFIRST,~x);
                    digitalWrite(latchPin,HIGH);
                    x>>=1;
                    delay(1);
                }
            }
        }
```

### Python Code 19.1.1 LEDMatrix

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 19.1.1_LEDMatrix directory of Python language.

```
cd ~/Freenove_Kit/Code/Python_Code/19.1.1_LEDMatrix
```

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

```
python LEDMatrix.py
```

After the program is executed, the LED Matrix display a smiling face, and then display characters "0 to F" scrolling in a loop on the LED Matrix.

The following is the program code:

```
import RPi.GPIO as GPIO
import time

LSBFIRST = 1
MSBFIRST = 2
#define the pins connect to 74HC595
dataPin   = 11      #DS Pin of 74HC595(Pin14)
latchPin  = 13      #ST_CP Pin of 74HC595(Pin12)
clockPin = 15       #SH_CP Pin of 74HC595(Pin11)
pic = [0x1c,0x22,0x51,0x45,0x45,0x51,0x22,0x1c]# data of smiling face
data = [#data of "0-F"
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, # " "
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x3E, 0x00, 0x00, # "0"
    0x00, 0x00, 0x21, 0x7F, 0x01, 0x00, 0x00, 0x00, # "1"
    0x00, 0x00, 0x23, 0x45, 0x49, 0x31, 0x00, 0x00, # "2"
    0x00, 0x00, 0x22, 0x49, 0x49, 0x36, 0x00, 0x00, # "3"
    0x00, 0x00, 0x0E, 0x32, 0x7F, 0x02, 0x00, 0x00, # "4"
    0x00, 0x00, 0x79, 0x49, 0x49, 0x46, 0x00, 0x00, # "5"
    0x00, 0x00, 0x3E, 0x49, 0x49, 0x26, 0x00, 0x00, # "6"
    0x00, 0x00, 0x60, 0x47, 0x48, 0x70, 0x00, 0x00, # "7"
    0x00, 0x00, 0x36, 0x49, 0x49, 0x36, 0x00, 0x00, # "8"
    0x00, 0x00, 0x32, 0x49, 0x49, 0x3E, 0x00, 0x00, # "9"
    0x00, 0x00, 0x3F, 0x44, 0x44, 0x3F, 0x00, 0x00, # "A"
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x36, 0x00, 0x00, # "B"
    0x00, 0x00, 0x3E, 0x41, 0x41, 0x22, 0x00, 0x00, # "C"
    0x00, 0x00, 0x7F, 0x41, 0x41, 0x3E, 0x00, 0x00, # "D"
    0x00, 0x00, 0x7F, 0x49, 0x49, 0x41, 0x00, 0x00, # "E"
    0x00, 0x00, 0x7F, 0x48, 0x48, 0x40, 0x00, 0x00, # "F"
    0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, # " "
]
def setup():
    GPIO.setmode(GPIO.BOARD)     # Number GPIOs by its physical location
    GPIO.setup(dataPin, GPIO.OUT)
    GPIO.setup(latchPin, GPIO.OUT)
```

```
    GPIO.setup(clockPin, GPIO.OUT)

def shiftOut(dPin,cPin,order,val):
    for i in range(0,8):
        GPIO.output(cPin,GPIO.LOW);
        if(order == LSBFIRST):
            GPIO.output(dPin,(0x01&(val>>i)==0x01) and GPIO.HIGH or GPIO.LOW)
        elif(order == MSBFIRST):
            GPIO.output(dPin,(0x80&(val<<i)==0x80) and GPIO.HIGH or GPIO.LOW)
        GPIO.output(cPin,GPIO.HIGH);

def loop():
    while True:
        for j in range(0,500):# Repeat enough times to display the smiling face a period
of time
            x=0x80
            for i in range(0,8):
                GPIO.output(latchPin,GPIO.LOW)
                shiftOut(dataPin,clockPin,MSBFIRST,pic[i]) #first shift data of line
information to first stage 74HC959

                shiftOut(dataPin,clockPin,MSBFIRST,~x) #then shift data of column
information to second stage 74HC959
                GPIO.output(latchPin,GPIO.HIGH)# Output data of two stage 74HC595 at the
same time
                time.sleep(0.001)# display the next column
                x>>=1
        for k in range(0,len(data)-8):#len(data) total number of "0-F" columns
            for j in range(0,20):# times of repeated displaying LEDMatrix in every frame,
the bigger the "j", the longer the display time.
                x=0x80      # Set the column information to start from the first column
                for i in range(k,k+8):
                    GPIO.output(latchPin,GPIO.LOW)
                    shiftOut(dataPin,clockPin,MSBFIRST,data[i])
                    shiftOut(dataPin,clockPin,MSBFIRST,~x)
                    GPIO.output(latchPin,GPIO.HIGH)
                    time.sleep(0.001)
                    x>>=1
def destroy():
    GPIO.cleanup()
if __name__ == '__main__':
    print ('Program is starting...' )
    setup()
    try:
```

```
79          loop()
80      except KeyboardInterrupt:
81          destroy()
```

The first “for” loop in the “while” loop is used to display a static smile. Displaying column information from left to right, one column at a time with a total of 8 columns. This repeats 500 times to ensure sufficient display time.

```
        for j in range(0,500):# Repeat enough times to display the smiling face a period of time
            x=0x80
            for i in range(0,8):
                GPIO.output(latchPin,GPIO.LOW)
                shiftOut(dataPin,clockPin,MSBFIRST,pic[i])#first shift data of line information to first stage 74HC959
                shiftOut(dataPin,clockPin,MSBFIRST,~x)#then shift data of column information to first stage 74HC959

                GPIO.output(latchPin,GPIO.HIGH)# Output data of two stage 74HC595 at the same time.
                time.sleep(0.001)# display the next column
                x>>=1
```

The second “for” loop is used to display scrolling characters "0 to F", for a total of 18 X 8 = 144 columns. Displaying the 0-8 column, then the 1-9 column, then the 2-10 column..... and so on…138-144 column in consecutively to achieve the scrolling effect. The display of each frame is repeated a certain number of times and the more repetitions, the longer the single frame display will be and the slower the scrolling movement.

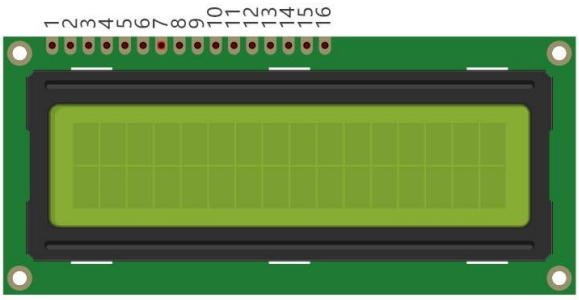
```
        for k in range(0,len(data)-8):#len(data) total number of “0-F” columns.
            for j in range(0,20):# times of repeated displaying LEDMatrix in every frame, the bigger the “j”, the longer the display time
                x=0x80      # Set the column information to start from the first column
                for i in range(k,k+8):
                    GPIO.output(latchPin,GPIO.LOW)
                    shiftOut(dataPin,clockPin,MSBFIRST,data[i])
                    shiftOut(dataPin,clockPin,MSBFIRST,~x)
                    GPIO.output(latchPin,GPIO.HIGH)
                    time.sleep(0.001)
                    x>>=1
```

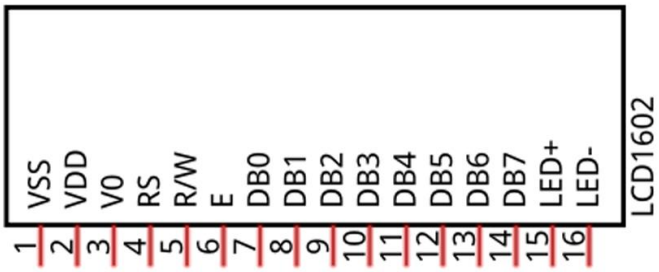
# Chapter 20 LCD1602

In this chapter, we will learn about the LCD1602 Display Screen,

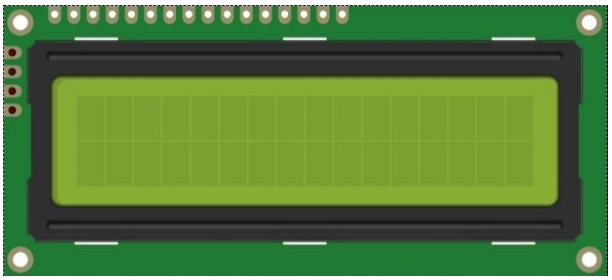
## Project 20.1 I2C LCD1602

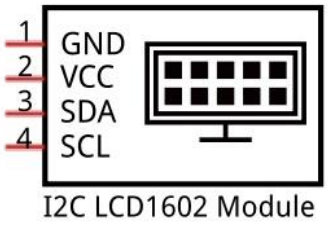
There are LCD1602 display screen and the I2C LCD. We will introduce both of them in this chapter. But what we use in this project is an I2C LCD1602 display screen. The LCD1602 Display Screen can display 2 lines of characters in 16 columns. It is capable of displaying numbers, letters, symbols, ASCII code and so on. As shown below is a monochrome LCD1602 Display Screen along with its circuit pin diagram






I2C LCD1602 Display Screen integrates a I2C interface, which connects the serial-input & parallel-output module to the LCD1602 Display Screen. This allows us to only use 4 lines to operate the LCD1602.

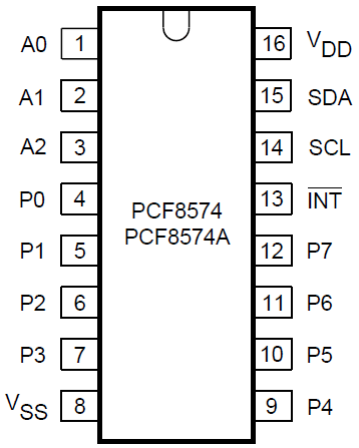




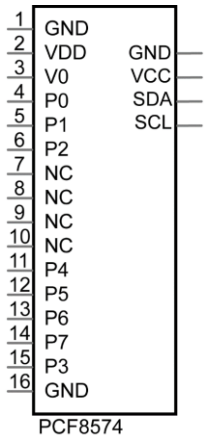

The serial-to-parallel IC chip used in this module is PCF8574T (PCF8574AT), and its default I2C address is 0x27(0x3F). You can also view the RPI bus on your I2C device address through command "i2cdetect -y 1" (refer to the "configuration I2C" section below).

Below is the PCF8574 chip pin diagram and its module pin diagram:

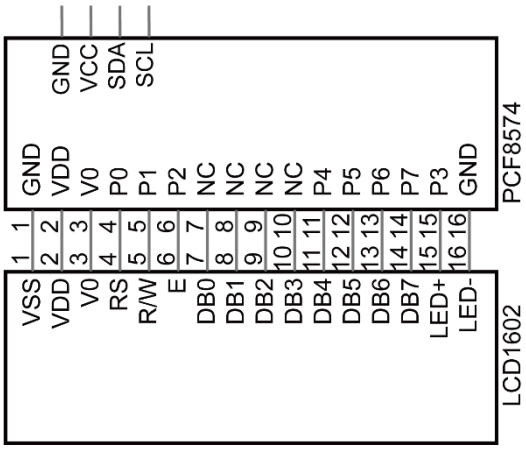
PCF8574 chip pin diagram:




PCF8574 module pin diagram



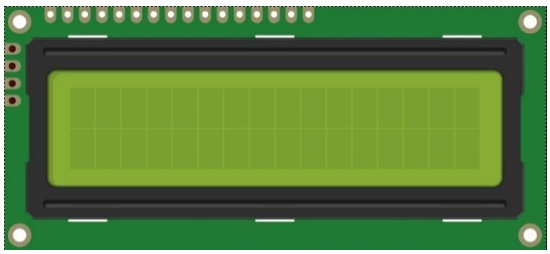

PCF8574 module pins and LCD1602 pins correspond to each other and connected to each other:




Because of this, as stated earlier, we only need 4 pins to control the16 pins of the LCD1602 Display Screen through the I2C interface.

In this project, we will use the I2C LCD1602 to display some static characters and dynamic variables.

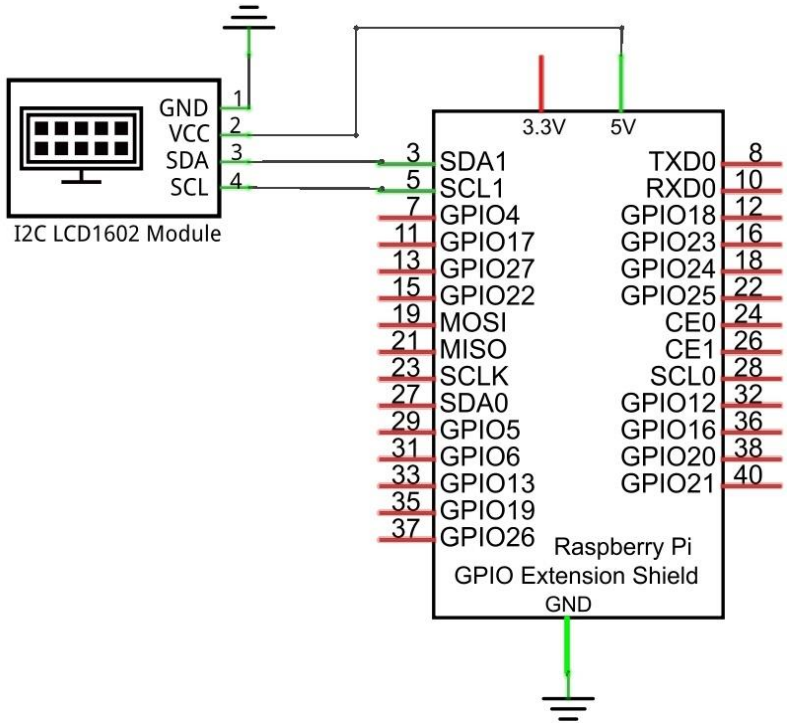
## Component List

| | |
|---|---|
| Raspberry Pi (with 40 GPIO) x1<br>GPIO Extension Board & Ribbon Cable x1<br>Breadboard x1 | Jumper Wire x4  |
| I2C LCD1602 Module x1  | |

# Circuit

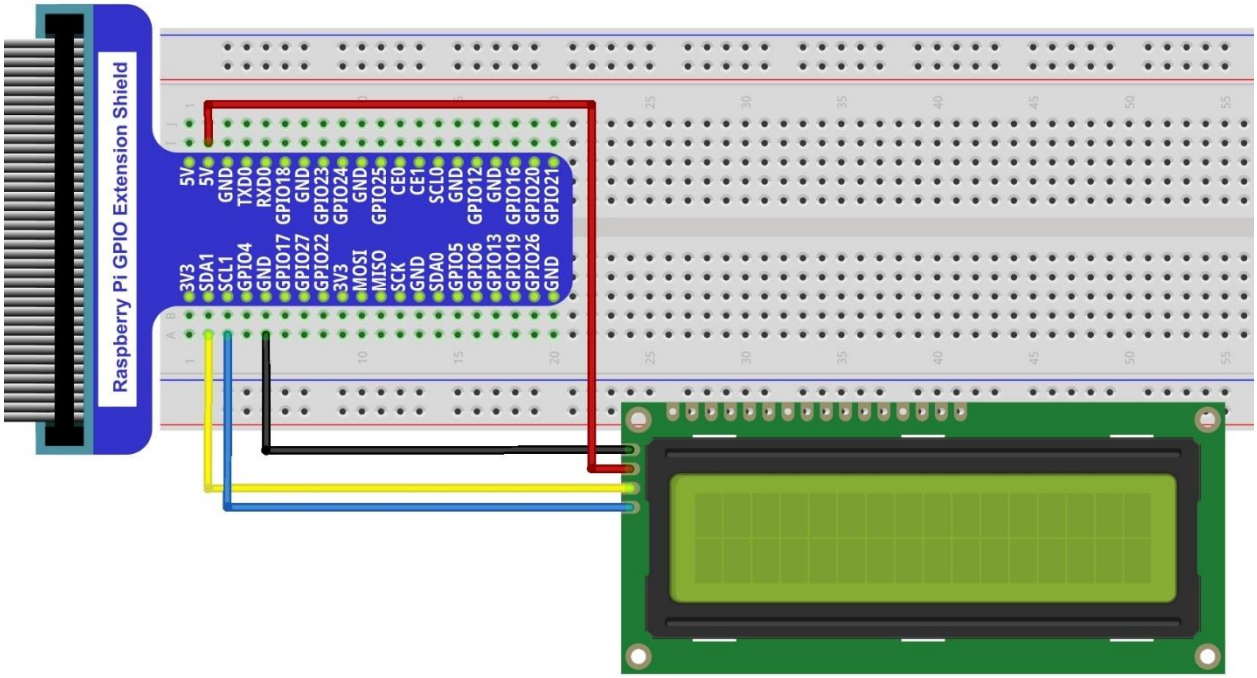
Note that the power supply for I2C LCD1602 in this circuit is 5V.

Schematic diagram




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

**NOTE: It is necessary to configure 12C and install Smbus first (see chapter 7 for details)**



# Code

This code will have your RPi's CPU temperature and System Time Displayed on the LCD1602.

C Code 20.1.1 I2CLCD1602

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

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

2. Use following command to compile "I2CLCD1602.c" and generate executable file "I2CLCD1602".

```
gcc I2CLCD1602.c -o I2CLCD1602 -lwiringPi -lwiringPiDev
```

3. Then run the generated file "I2CLCD1602".

```
sudo ./ I2CLCD1602
```

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:

```c
#include <stdlib.h>
#include <stdio.h>
#include <wiringPi.h>
#include <wiringPiI2C.h>
#include <pcf8574.h>
#include <lcd.h>
#include <time.h>

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

int lcdhd;// used to handle LCD
void printCPUTemperature(){// sub function 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);
}
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:%02d:%02d:%02d",timeinfo->tm_hour,timeinfo->tm_min,timeinfo->tm_sec);
//Display system time on LCD
}
int detectI2C(int addr){  //Used to detect i2c address of LCD
    int _fd = wiringPiI2CSetup (addr);
    if (_fd < 0){
        printf("Error address : 0x%x \n",addr);
        return 0 ;
    }
    else{
        if(wiringPiI2CWrite(_fd,0) < 0){
            printf("Not found device in address 0x%x \n",addr);
            return 0;
        }
        else{
            printf("Found device in address 0x%x \n",addr);
            return 1 ;
        }
```