7.7-Segment_Display

Introduction

In this lesson, we try to drive a 7-segment display to show a figure from 0 to 9 and A to F.

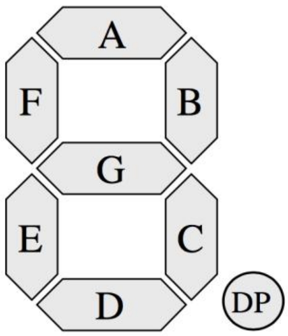
Hardware Required

- ✓ 1 * Raspberry Pi
- ✓ 1 * T-Extension Board
- ✓ 1 * Seven-segment display
- ✓ 1 * 40-pin Cable
- ✓ Several Jumper Wires
- ✓ 1 * Resistor(220Ω)
- ✓ 1 * Breadboard
- ✓ 1 * 74HC595

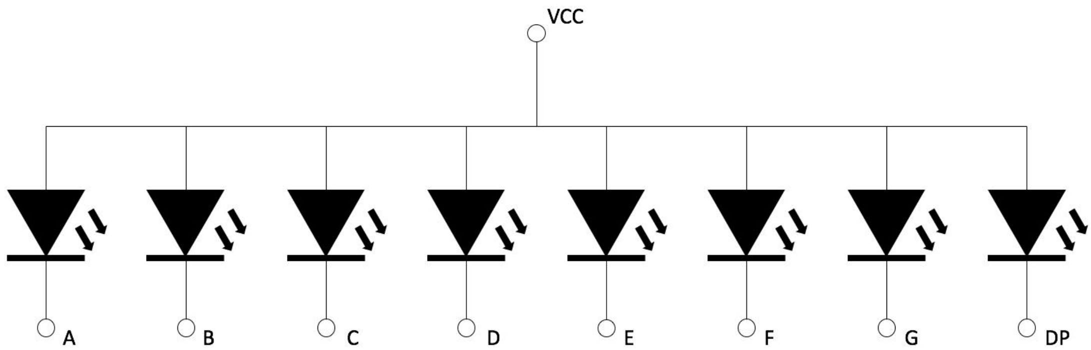
Principle

7-Segment Display

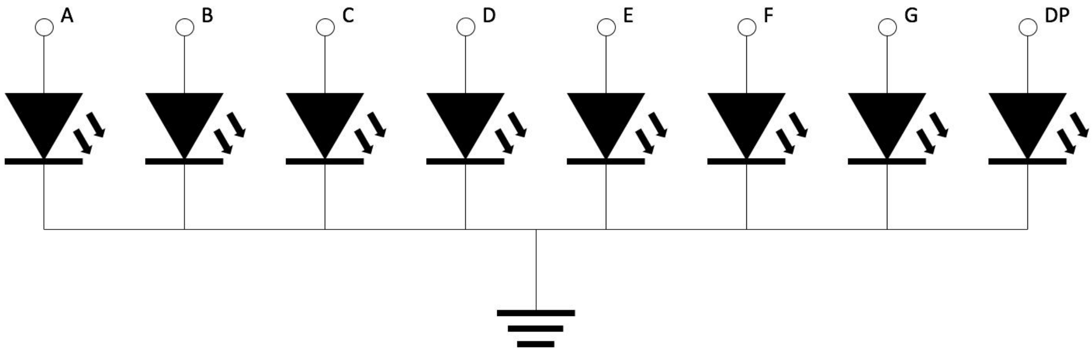
A 7-segment display is a LED module composed of 8 LEDs. 7 of the LEDs are for segments of one digit (shown as A to G below) and the other LED is for the decimal point (shown as DP below).



A common-anode seven-segment display



A common-cathode seven-segment display



For saving the pin number for controlling a seven-segment display, a shift register is used as a serial-to-parallel converter to send signals to the display. That is, we serially send 8 bits of data, which represents the way we want to turn on the display, by one signal pin into the shift register and the register can output the corresponding data pattern to its 8 output pins at once (parallel).

Display Codes

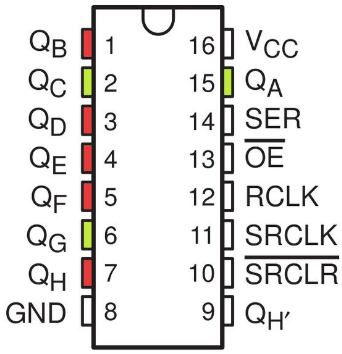
To help you get to know how 7-segment displays(Common Cathode) display Numbers, we have drawn the following table. Numbers are the number 0-F displayed on the 7-segment display; (DP) GFEDCBA refers to the corresponding LED set to 0 or 1, For example, 00111111 means that DP and G are set to 0, while others are set to 1.

Therefore, the number 0 is displayed on the 7-segment display, while HEX Code corresponds to hexadecimal number.

Numbers	Common Cathode		Numbers	Common Cathode	
	(DP)GFEDCBA	Hex Code		(DP)GFEDCBA	Hex Code
0	00111111	0x3f	A	01110111	0x77

1	00000110	0x06	B	01111100	0x7c
2	01011011	0x5b	C	00111001	0x39
3	01001111	0x4f	D	01011110	0x5e
4	01100110	0x66	E	01111001	0x79
5	01101101	0x6d	F	01110001	0x71
6	01111101	0x7d			
7	00000111	0x07			
8	01111111	0x7f			
9	01101111	0x6f			

74HC595



The 74HC595 consists of an 8−bit shift register and a storage register with three−state parallel outputs. It converts serial input into parallel output so that you can save IO ports of an MCU. The 74HC595 is widely used to indicate multipath LEDs and drive multi-bit segment displays. "Three-state" refers to the fact that you can set the output pins as either high, low or "high impedance." With data latching, the instant output will not be affected during the shifting; with data output, you can cascade 74HC595s more easily.

The chip contains eight pins that we can use for output, each of which is associated with a bit in the register. In the case of the 74HC595 IC, we refer to these as QA through to QH. In order to write to these outputs via the Arduino, we have to send a binary value to the shift register, and from that number the shift register can figure out which outputs to use. For example, if we sent the binary value 10100010, the pins highlighted in green in the image below would be active and the ones highlighted in red would be inactive.

Pins of 74HC595 and their functions:

Q0-Q7: 8-bit parallel data output pins, able to control 8 LEDs or 8 pins of 7-segment display directly.

Q7 ' : Series output pin, connected to DS of another 74HC595 to connect multiple 74HC595s in series

MR: Reset pin, active at low level;

SHcp: Time sequence input of shift register. On the rising edge, the data in shift register moves successively one bit, i.e. data in Q1 moves to Q2, and so forth. While on the falling edge, the data in shift register remain unchanged.

STcp: Time sequence input of storage register. On the rising edge, data in the shift register moves into memory register.

CE: Output enable pin, active at low level.

DS: Serial data input pin

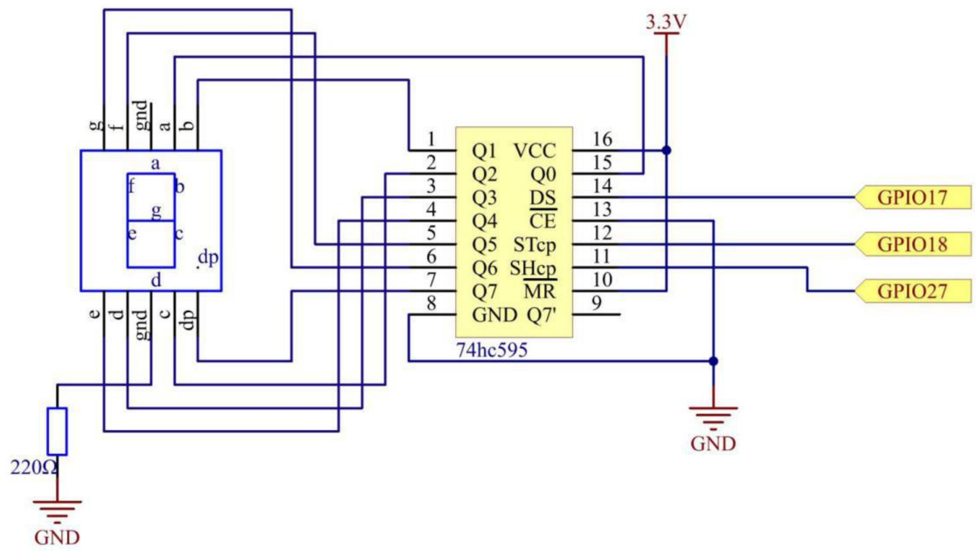
VCC: Positive supply voltage

GND: Ground

Schematic Diagram

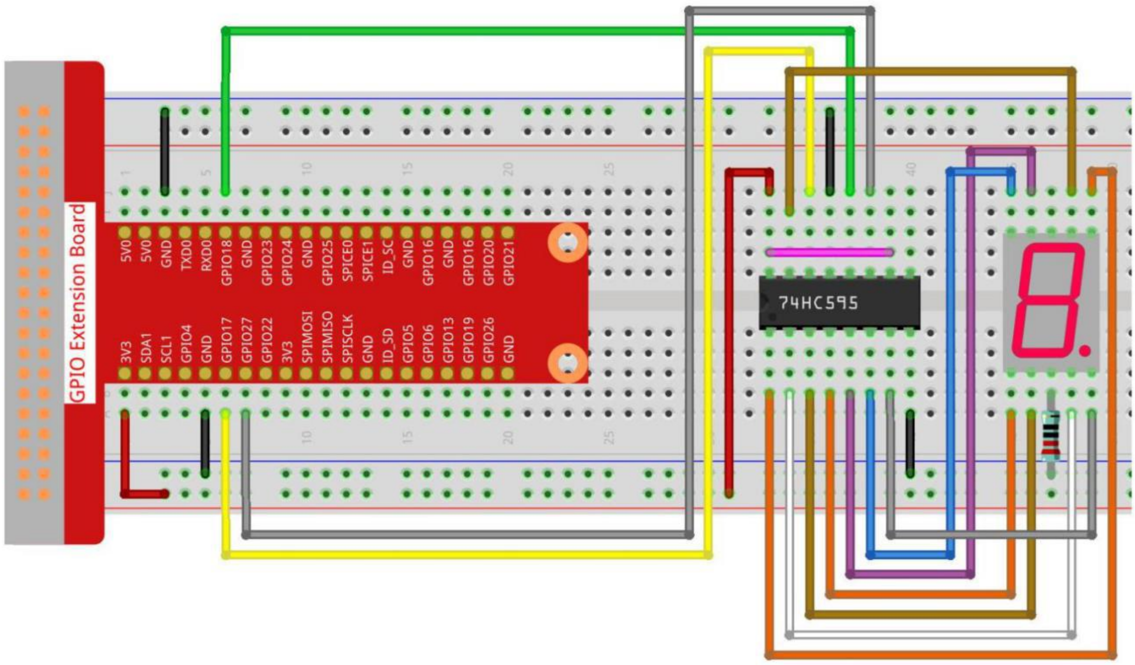
Connect pin ST_CP of 74HC595 to Raspberry Pi GPIO18, SH_CP to GPIO27, DS to GPIO17, parallel output ports to 8 segments of the LED segment display. Input data in DS pin to shift register when SH_CP (the clock input of the shift register) is at the rising edge, and to the memory register when ST_CP (the clock input of the memory) is at the rising edge. Then you can control the states of SH_CP and ST_CP via the Raspberry Pi GPIOs to transform serial data input into parallel data output so as to save Raspberry Pi GPIOs and drive the display.

T-Board Name	physical	wiringPi	BCM
GPIO17	Pin 11	0	17
GPIO18	Pin 12	1	18
GPIO27	Pin 13	2	27



Experimental Procedures

Step 1: Build the circuit.



For C Language Users

Step 2: Get into the folder of the code.

```
cd /home/pi/REXQualis_Raspberry_Pi_Complete_Starter_Kit/C/7.7-Segment_Display
```

Step 3: Compile.

```
gcc 7.7-Segment_Display.c -o 7-Segment_Display.out -lwiringPi
```

Step 4: Run the executable file above.

```
sudo ./7-Segment_Display.out
```

After the code runs, you'll see the 7-segment display display 0-9, A-F.

Code

```
#include <wiringPi.h>

#include <stdio.h>

#define SDI     0     //serial data input

#define RCLK 1     //memory clock input(STCP)

#define SRCLK 2     //shift register clock input(SHCP)

unsigned char SegCode[16] =

{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};

void init(void){

    pinMode(SDI, OUTPUT);

    pinMode(RCLK, OUTPUT);

    pinMode(SRCLK, OUTPUT);

    digitalWrite(SDI, 0);

    digitalWrite(RCLK, 0);

    digitalWrite(SRCLK, 0);

}

void hc595_shift(unsigned char dat){

    int i;

    for(i=0;i<8;i++){

        digitalWrite(SDI, 0x80 & (dat << i));

        digitalWrite(SRCLK, 1);

        delay(1);

        digitalWrite(SRCLK, 0);

    }

        digitalWrite(RCLK, 1);

        delay(1);
```

```
        digitalWrite(RCLK, 0);
}
int main(void){
    int i;
    if(wiringPiSetup() == -1){ //when initialize wiring failed, print messageto screen
        printf("setup wiringPi failed !");
        return 1;
    }
    init();
    while(1){
        for(i=0;i<16;i++){
            printf("Print %1X on Segment\n", i); // %X means hex output
            hc595_shift(SegCode[i]);
            delay(500);
        }
    }
    return 0;
}
```

Code Explanation

```
unsigned char SegCode[16] =
{0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71};
```

A segment code array from 0 to F in Hexadecimal (Common cathode).

```
void init(void){
    pinMode(SDI, OUTPUT);
    pinMode(RCLK, OUTPUT);
    pinMode(SRCLK, OUTPUT);
    digitalWrite(SDI, 0);
    digitalWrite(RCLK, 0);
    digitalWrite(SRCLK, 0);
}
```

Set ds, st_cp, sh_cp three pins to OUTPUT, and the initial state as 0.

```
void hc595_shift(unsigned char dat){}
```

To assign 8 bit value to 74HC595's shift register.

```
        digitalWrite(SDI, 0x80 & (dat << i));
```

Assign the dat data to SDI(DS) by bits. Here we assume dat=0x3f(0011 1111, when i=2, 0x3f will shift left(<<) 2 bits. 1111 1100 (0x3f << 2) & 1000 0000 (0x80) = 1000 0000, is true.

```
        digitalWrite(SRCLK, 1);
```

SRCLK's initial value was set to 0, and here it's set to 1, which is to generate a rising edge pulse, then shift the DS date to shift register.

```
        digitalWrite(RCLK, 1);
```

RCLK's initial value was set to 0, and here it's set to 1, which is to generate a rising edge, then shift data from shift register to storage register.

```
    while(1){
        for(i=0;i<16;i++){
            printf("Print %1X on Segment\n", i); // %X means hex output
            hc595_shift(SegCode[i]);
            delay(500);
        }
    }
```

In this for loop, we use "%1X" to output i as a hexadecimal number. Apply i to find the corresponding segment code in the SegCode[] array, and employ hc595_shift() to pass the SegCode into 74HC595's shift register.

For Python Language Users

Step 2: Get into the folder of the code.

```
cd /home/pi/REXQualis_Raspberry_Pi_Complete_Starter_Kit/Python
```

Step 3: Run.

```
sudo python3 7.7-Segment_Display.py
```

After the code runs, you'll see the 7-segment display display 0-9, A-F.

Code

```
import RPi.GPIO as GPIO

import time

        # Set up pins

SDI    = 17

RCLK   = 18

SRCLK = 27

        # Define a segment code from 0 to F in Hexadecimal

segCode =

[0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71]

def setup():

    GPIO.setmode(GPIO.BCM)

    GPIO.setup(SDI, GPIO.OUT, initial=GPIO.LOW)

    GPIO.setup(RCLK, GPIO.OUT, initial=GPIO.LOW)

    GPIO.setup(SRCLK, GPIO.OUT, initial=GPIO.LOW)

        # Shift the data to 74HC595

def hc595_shift(dat):

    for bit in range(0, 8):

        GPIO.output(SDI, 0x80 & (dat << bit))

        GPIO.output(SRCLK, GPIO.HIGH)

        time.sleep(0.001)

        GPIO.output(SRCLK, GPIO.LOW)

        GPIO.output(RCLK, GPIO.HIGH)

        time.sleep(0.001)

        GPIO.output(RCLK, GPIO.LOW)

def main():
```

```
    while True:
        # Shift the code one by one from segCode list
        for code in segCode:
            hc595_shift(code)
        print ("segCode[%s]: 0x%02X"%(segCode.index(code), code))
        # %02X means double digit HEX to print
            time.sleep(0.5)
def destroy():
    GPIO.cleanup()
if __name__ == '__main__':
    setup()
    try:
        main()
    except KeyboardInterrupt:
        destroy()
```

Code Explanation

```
segCode =
[0x3f,0x06,0x5b,0x4f,0x66,0x6d,0x7d,0x07,0x7f,0x6f,0x77,0x7c,0x39,0x5e,0x79,0x71]
```

A segment code array from 0 to F in Hexadecimal (Common cathode).

```
def setup():
    GPIO.setmode(GPIO.BCM)
    GPIO.setup(SDI, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(RCLK, GPIO.OUT, initial=GPIO.LOW)
    GPIO.setup(SRCLK, GPIO.OUT, initial=GPIO.LOW)
```

Set ds, st_cp, sh_cp three pins to output and the initial state as low level.

```
        GPIO.output(SDI, 0x80 & (dat << bit))
```

Assign the dat data to SDI(DS) by bits. Here we assume dat=0x3f(0011 1111, when bit=2, 0x3f will shift right(<<) 2 bits. 1111 1100 (0x3f << 2) & 1000 0000 (0x80) = 1000

0000, is true.

```
GGPIO.output(SRCLK, GPIO.HIGH)
```

SRCLK's initial value was set to LOW, and here it's set to HIGH, which is to generate a rising edge pulse, then shift the DS date to shift register.

```
GPIO.output(RCLK, GPIO.HIGH)
```

RCLK's initial value was set to LOW, and here it's set to HIGH, which is to generate a rising edge, then shift data from shift register to storage register.

Note: The hexadecimal format of number 0~15 are (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D,E, F)

Phenomenon Picture