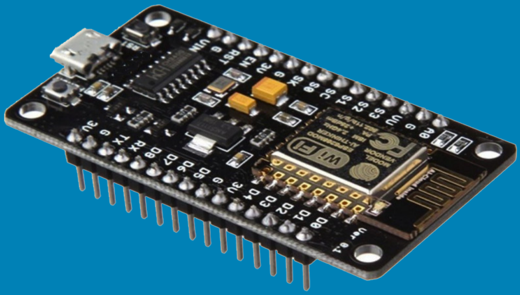


Erwin Ouyang

Hands-On ESP8266: Mastering Basic Peripherals

Developing with Arduino and C/C++ by Examples

Hands-On EMBEDDED

Hands-On ESP8266: Mastering Basic Peripherals

Developing with Arduino and C/C++ by Examples

Erwin Ouyang

Build your own dreams, or someone else will hire you to build theirs.

— FARRAH GRAY

Preface

Rapid advances in IoT technology demand a lot of devices to be connected to the internet. To design such devices, we usually need microcontrollers and network modules (Ethernet or WiFi). ESP8266 is a low-cost microcontroller that already has an on-chip WiFi module. This WiFi module makes the ESP8266 as a popular microcontroller for IoT device development. With ESP8266, we do not need external the WiFi module, so we can significantly reduce the Bill of Material (BOM) cost of the IoT product. This book is primarily written as a hands-on material rather than theory. This book covers the basic of ESP8266 peripherals and the popular sensors and actuators. In this book, the Arduino library is used for programming the ESP8266, and the NodeMCU development board is used.

Erwin Ouyang
October 2018

Contents

Listings

Chapter 1

Introduction

1.1 ESP8266

ESP8266 is a low-cost WiFi chip. ESP8266 chip is shown in Figure 1.1. This chip consists of a microcontroller and a full TCP/IP stack. ESP8266 is made by Espressif Systems, a Shanghai-based Chinese fabless semiconductor company. ESP8266 is a very popular chip for developing IoT devices. The ESP8266's specifications are listed as the following [1]:

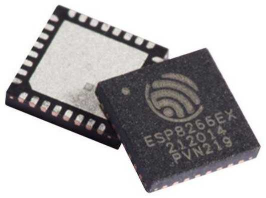


Figure 1.1. ESP8266 chip. Retrieved August 26, 2018, from gridconnect.com

- L106 32-bit RISC microprocessor core based on the Tensilica Xtensa Diamond Standard 106Micro running at 80 MHz
- Memory:
  - 32 KiB instruction RAM
  - 32 KiB instruction cache RAM
  - 80 KiB user-data RAM
  - 16 KiB ETS system-data RAM
- External QSPI flash: up to 16 MiB is supported (512 KiB to 4 MiB typically included)
- IEEE 802.11 b/g/n Wi-Fi
  - Integrated transmit/receive (TR) switch, balun, LNA, power amplifier, and matching network
  - WEP or WPA/WPA2 authentication, or open networks
- 16 GPIO pins (GPIO6–11 are used for communication with on-board flash memory).
- SPI
- I^{2}C (software implementation)
- I^{2}S interfaces with DMA (sharing pins with GPIO)
- UART on dedicated pins, plus a transmit-only UART can be enabled on GPIO2
- 10-bit ADC (successive approximation ADC)

With ESP8266, you can make IoT devices that are connected to the internet through WiFi network. You can also make embedded web server in this chip, so your PC or smartphone can connect to the ESP8266.

What is the difference between ESP8266 and ESP8266EX?

ESP8266 is the initial version. ESP8266EX is the updated version. Now ESP8266EX is the most commonly available.

1.2 ESP-12E

ESP-12E is a WiFi module that uses the ESP8266. ESP-12E is shown in Figure 1.2. ESP-12E is made by Ai-Thinker, a third-party manufacturer. There are other ESP modules made by this manufacturer. They are referred to as "ESP-xx modules". The disadvantage of ESP-12E is that it is not breadboard friendly.

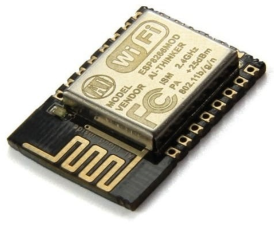


Figure 1.2. ESP-12E module. Retrieved August 26, 2018, from hackster.io

ESP-12E has an RF shield (metal enclosure) that covers the ESP8266 chip as shown in Figure 1.3. This shield is used for compliance with Federal Communications Commission (FCC) emissions rules. This shiled can minimize interference with other devices.

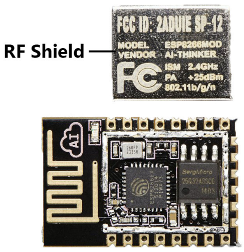


Figure 1.3. RF shield on ESP-12E. Retrieved August 26, 2018, from amicus.com.sg

1.3 NodeMCU

NodeMCU is a development board that uses the ESP-12E. NodeMCU is shown in Figure 1.4. This is NodeMCU V3 which is used in this book. At the time of writing, it is the latest NodeMCU generation. Compared to the ESP-12E, NodeMCU is breadboard friendly and includes USB to serial interface. NodeMCU can be programmed using Lua scipting, Arduino, or ESP8266 SDK. In this book, we will program NodeMCU using Arduino, which is the easiest and most popular method.

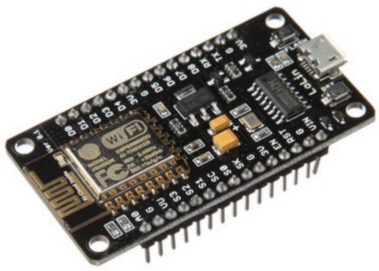


Figure 1.4. NodeMCU development board. Retrieved August 26, 2018, from root.cz

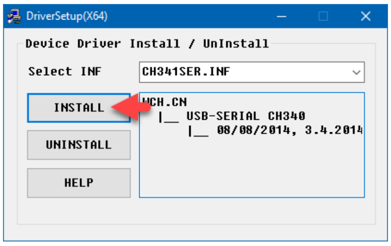
What are the differences between NodeMCU V1, V2, and V3?

NodeMCU V1 is the first generation, but now outdated. NodeMCU V2 and V3 are the second generation. V2 is made by Amica, while V3 is made by LoLin. V2 and V3 use a different USB to serial chip. V3's board size is significantly larger than the V2.

1.4 Driver Installation

Before we can program NodeMCU, we need to install the USB driver. The step-by-step how to install NodeMCU's USB driver is described as the following:

- Download NodeMCU V3 driver from this link: **`https://github.com/nodemcu/nodemcu-devkit/tree/master/Drivers`**
- Connect **NodeMCU** development board to the USB port.
- Run the driver installer file: **CH341SER.EXE**, and install the driver as shown in Figure 1.5.
- Open **Device Manager** in order to find NodeMCU's COM port as shown in Figure 1.6.



Select INF: CH341SER.INF

```
WCH.CN
  |__ USB-SERIAL CH340
       |__ 08/08/2014, 3.4.2014
```

Figure 1.5. NodeMCU V3 USB driver installation

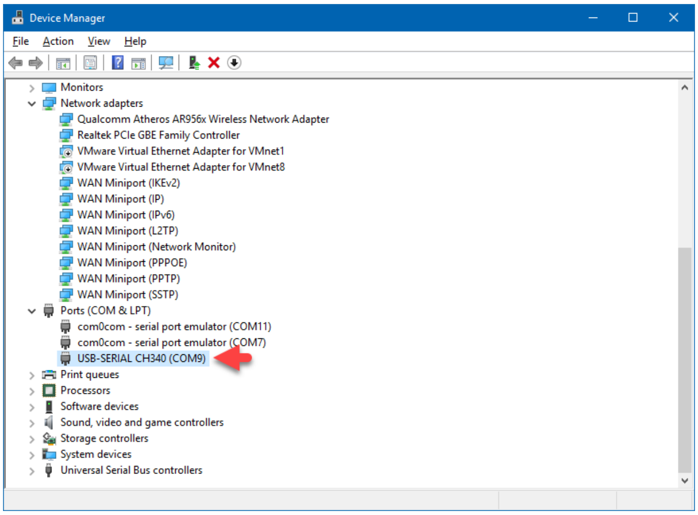

Device Manager
File Action View Help
Monitors
Network adapters
Qualcomm Atheros AR956x Wireless Network Adapter
Realtek PCIe GBE Family Controller
VMware Virtual Ethernet Adapter for VMnet1
VMware Virtual Ethernet Adapter for VMnet8
WAN Miniport (IKEv2)
WAN Miniport (IP)
WAN Miniport (IPv6)
WAN Miniport (L2TP)
WAN Miniport (Network Monitor)
WAN Miniport (PPPOE)
WAN Miniport (PPTP)
WAN Miniport (SSTP)
Ports (COM & LPT)
com0com - serial port emulator (COM11)
com0com - serial port emulator (COM7)
USB-SERIAL CH340 (COM9)
Print queues
Processors
Software devices
Sound, video and game controllers
Storage controllers
System devices
Universal Serial Bus controllers

Figure 1.6. NodeMCU's COM port in Device Manager

If you use NodeMCU V2, then the USB driver is different. NodeMCU V2 use CP210x USB to serial chip, while NodeMCU V3 use CH341 USB to serial chip. The installation process NodeMCU V2 USB driver is similar to NodeMCU V3 USB driver. You can download the NodeMCU V2 USB driver from `https://www.silabs.com/products/development-tools/software/usb-to-uart-bridge-vcp-drivers`.

You can also find NodeMCU V2 USB driver in Arduino software installation folder, under the `driver` folder, there is CP210x driver. You can also use this driver for NodeMCU V2, so you do not have to download the driver. You have to manually install the driver from **Device Manager** by right clicking on the NodeMCU's USB port, select **Update Driver Software** menu, and search for the driver in `arduino-x.x.x/driver` folder.

1.5 ESP8266 Library Installation

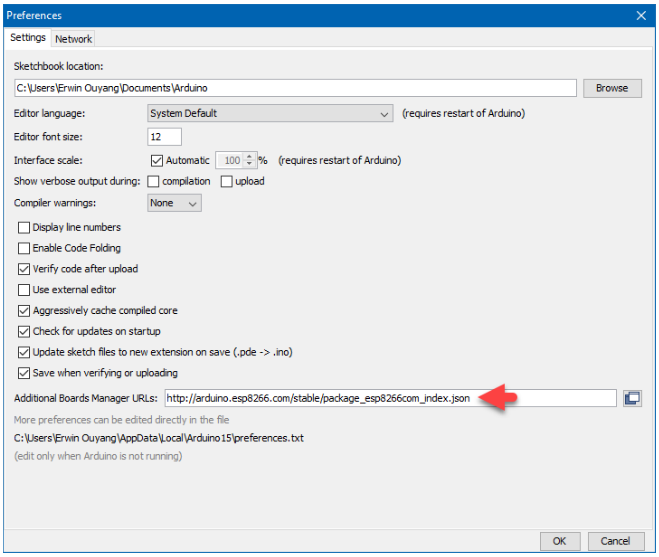
Arduino IDE will be used for programming ESP8266 in this book. At the time of writing, the latest Arduino IDE version is 1.8.5. You can download either the installed or the portable version. ESP8266 library is not included in Arduino IDE, so you must install it manually. The step-by-step how to install the ESP8266 library is described as the following:

- Start Arduino IDE, go to **File** menu, and open **Preferences** window.
- Enter **`http://arduino.esp8266.com/stable/package_esp8266com_index.json`** into **Additional Board Manager URLs** field as shown in Figure 1.7.
- Go to **Tools** → **Board** menu, open **Boards Manager**, and install **esp8266** platform as shown in Figure 1.8.
- After installation, select **NodeMCU 1.0 (ESP-12E Module)** board from **Tools** → **Board** menu.

The ESP8266 library files can be found in `C:\Users\<USER>\AppData\Local\Arduino15\packages\esp8266\hardware\esp8266\2.4.1\cores\esp8266`.

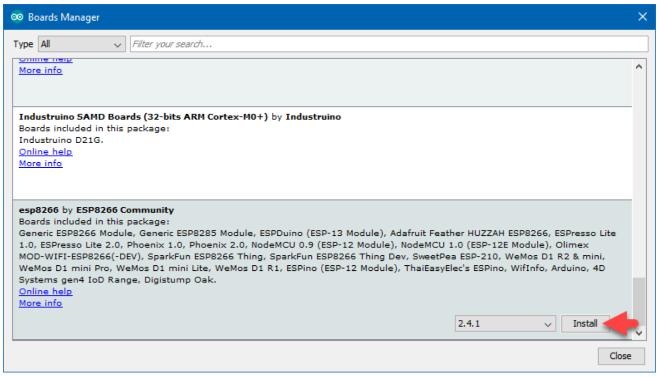
1.6 Create the First Program

For the very first program, we will create a simple program that turns on the ESP-12E's on-board LED. The code for turning on the on-board LED is shown in Listing 1.1. The on-board LED is connected to pin D4, and the circuit is active-low, so a logic `LOW` is needed to turn on the LED. The details about LED circuits will be explained later in chapter 2. The `digitalWrite` function is used for writing a digital value to a digital output pin.



Sketchbook location:
C:\Users\Erwin Ouyang\Documents\Arduino
Editor language: System Default (requires restart of Arduino)
Editor font size: 12
Interface scale: [✓] Automatic 100 % (requires restart of Arduino)
Show verbose output during: [ ] compilation [ ] upload
Compiler warnings: None
[ ] Display line numbers
[ ] Enable Code Folding
[✓] Verify code after upload
[ ] Use external editor
[✓] Aggressively cache compiled core
[✓] Check for updates on startup
[✓] Update sketch files to new extension on save (.pde -> .ino)
[✓] Save when verifying or uploading
Additional Boards Manager URLs: http://arduino.esp8266.com/stable/package_esp8266com_index.json
More preferences can be edited directly in the file
C:\Users\Erwin Ouyang\AppData\Local\Arduino15\preferences.txt
(edit only when Arduino is not running)

Figure 1.7. Enter ESP8266 package in Arduino preferences



Online help
More info

Industruino SAMD Boards (32-bits ARM Cortex-M0+) by **Industruino**
Boards included in this package:
Industruino D21G.
Online help
More info

esp8266 by **ESP8266 Community**
Boards included in this package:
Generic ESP8266 Module, Generic ESP8285 Module, ESPDuino (ESP-13 Module), Adafruit Feather HUZZAH ESP8266, ESPresso Lite 1.0, ESPresso Lite 2.0, Phoenix 1.0, Phoenix 2.0, NodeMCU 0.9 (ESP-12 Module), NodeMCU 1.0 (ESP-12E Module), Olimex MOD-WIFI-ESP8266(-DEV), SparkFun ESP8266 Thing, SparkFun ESP8266 Thing Dev, SweetPea ESP-210, WeMos D1 R2 & mini, WeMos D1 mini Pro, WeMos D1 mini Lite, WeMos D1 R1, ESPino (ESP-12 Module), ThaiEasyElec's ESPino, WifInfo, Arduino, 4D Systems gen4 IoD Range, Digistump Oak.
Online help
More info
2.4.1

Figure 1.8. Install ESP8266 library

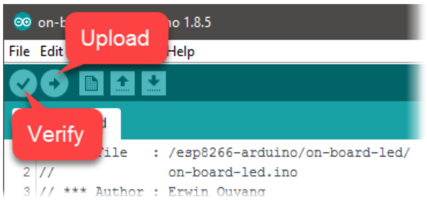
Listing 1.1. Turn on the on-board LED

```
// *** File   : /esp8266-arduino/on-board-led/
//              on-board-led.ino
// *** Author : Erwin Ouyang
// *** Date   : 17 Agt 2018

void setup()
{
    // Set on-board LED pin as output
    pinMode(D4, OUTPUT);
}

void loop()
{
    // Turn on the active-low LED
    digitalWrite(D4, LOW);
}
```

To compile the code, go to **Sketch** → **Verify/Compile** menu or click the checklist button in toolbar menu as shown in Figure 1.9. After the code is compiled without any error, you can connect NodeMCU to USB port, then go to **Tools** → **Port** menu, and select the correct COM port as you found in Device Manager. To upload the code to NodeMCU, you can go to **Sketch** → **Upload** menu or click the right arrow button in toolbar menu. The result of this program is shown in Figure 1.10.



```
ile     : /esp8266-arduino/on-board-led/
2 //              on-board-led.ino
3 // *** Author : Erwin Ouyang
```

Figure 1.9. Verify and Upload buttons

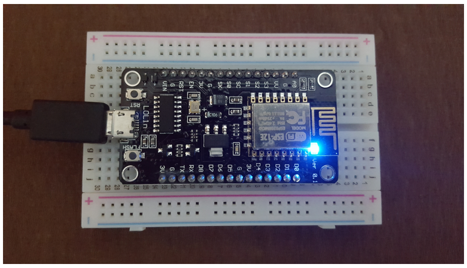


Figure 1.10. The on-board LED is on

1.7 Development Breadboard

All of the projects in this book will be done in a breadboard. The breadboard consists of NodeMCU, sensors, and actuators as shown in Figure 1.11. The list of components for this development breadboard is listed in Table 1.1. The circuit of this development breadboard is shown in Figure 1.12 and in Figure 1.13 for the schematic version.

Table 1.1. List of components for development breadboard

No.	**Component Name**	**Quantity**
1	NodeMCU board	1
2	Red LED	1
3	RGB LED common anode	1
4	0.96 Inch OLED I2C display 128x64	1
5	Push button	1
6	Trimpot 10kΩ	1
7	DS1307 RTC	1
8	32.768 kHz quartz crystal	1
9	DHT11 temperature and humidity sensor	1
10	100Ω resistor	3
11	150Ω resistor	1
12	4.7kΩ resistor	2

1.8 Summary

The development board used in this book is NodeMCU V3. Several sensors and actuators are required for example code in this book. The software tools required for programming NodeMCU are USB driver, Arduino IDE, and ESP8266 library.

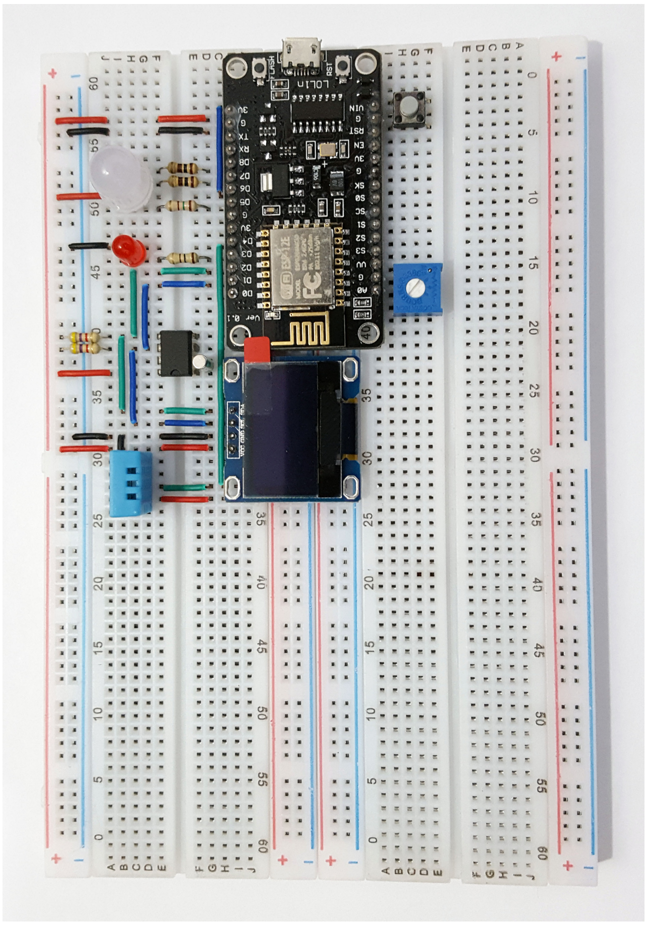


Figure 1.11. Development breadboard photograph

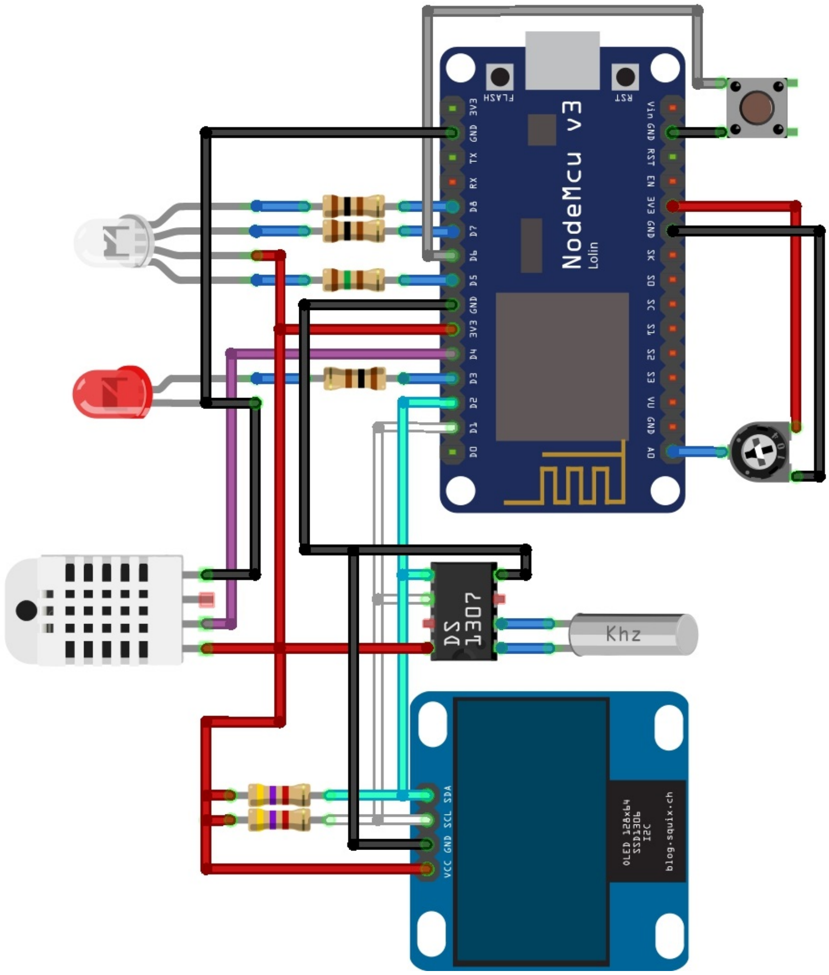

NodeMcu v3
Lolin
FLASH
RST
3V3 GND TX RX D8 D7 D6 D5 GND 3V3 D4 D3 D2 D1 D0
Vin GND RST EN 3V3 GND SK SO SC S1 S2 S3 VU GND A0
DS 1307
Khz
OLED 128x64
SSD1306
I2C
blog.squix.ch
SDA SCL GND VCC

Figure 1.12. Development breadboard circuit

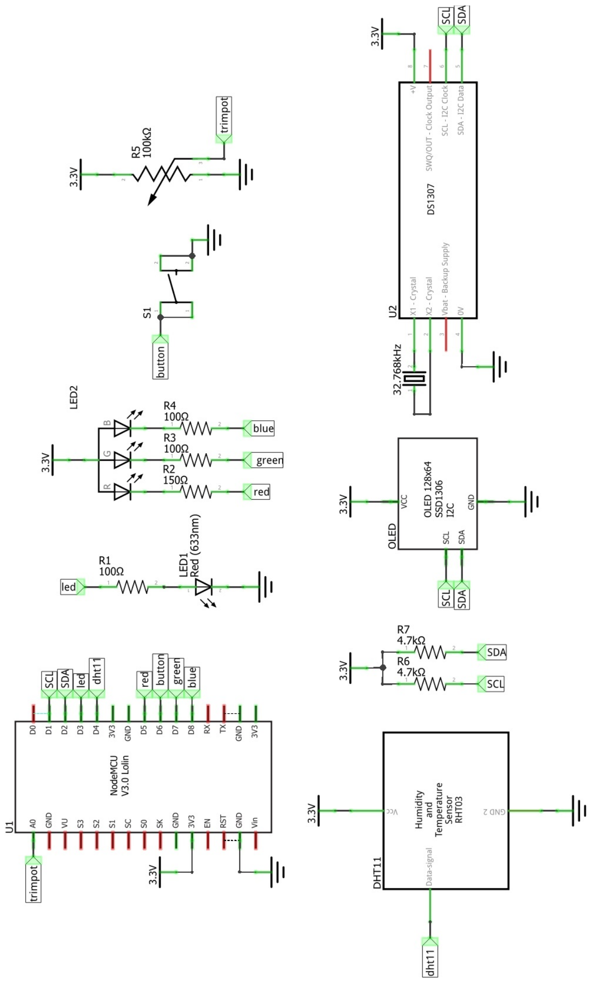


Figure 1.13. Development breadboard schematic

Chapter 2

Blink an LED

In chapter 1, the requirements for programming ESP8266 are explained. The very first program is created to make sure that everyting is properly installed. In this chapter, you will learn about General Purpose Input Output (GPIO) for blinking an LED.

What will you learn in this chapter?

- Build an LED circuit.
- Set a GPIO pin as output.
- Blink the LED using `digitalWrite` and `delay` function.

2.1 LED Circuit

In `on-board-led` program, when a logic `LOW` is written to the GPIO pin, then the LED will turn on. This is because the LED circuit is active-low. It is called active-low because a logic `LOW` is needed to activate (to turn on) the LED. The active-low LED circuit is shown in Figure 2.1. When a logic `LOW` is written to the GPIO pin, the current

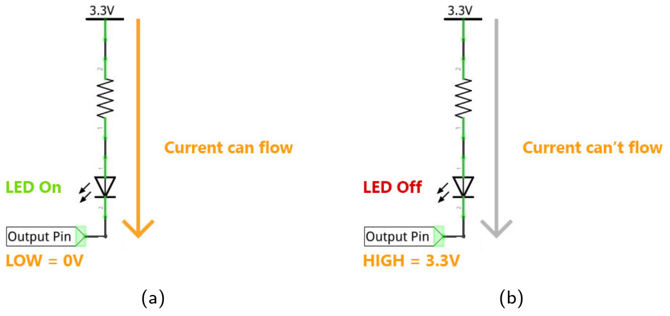


Figure 2.1. Active-low LED circuit: (a) A logic `LOW` turns on the LED; (b) A logic `HIGH` turns off the LED

can flow from the 3.3V through the resistor and LED, and then flows to the GND. Therefore the LED is on. On the other hand, when a logic `HIGH` is written to the GPIO pin, the current can not flow through the resistor and LED. Therefore the LED is off.

In this chapter, you will build the active-high LED circuit. It is called active-high because a logic `HIGH` is needed to activate (to turn on) the LED. The active-high circuit is shown in Figure 2.2. In active-high circuit, the LED will turn on, when a logic `HIGH` is written to the GPIO pin, and the LED will turn off, when a logic `LOW` is written to the GPIO pin. When a logic `HIGH` is written to the GPIO pin, the current can flow through the resistor and LED, and then flows to the GND. Therefore the LED is on. When a logic `LOW` is written, the current can not flow.

2.2 Digital Output

NodeMCU has 10 GPIO pins. The GPIO pins can be identified by ESP8266 pin numbering or NodeMCU pin numbering. NodeMCU pin is shown in Figure 2.3. The NodeMCU pin numbering starts from

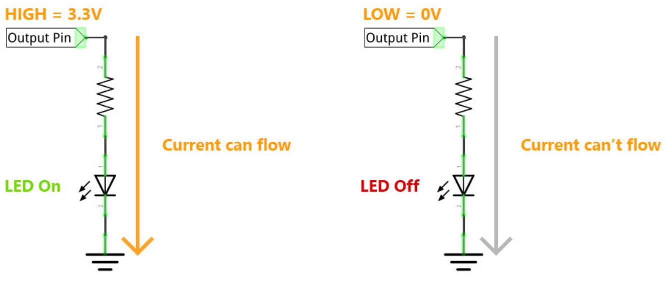


Figure 2.2. Active-high LED circuit: (a) A logic `HIGH` turns on the LED; (b) A logic `LOW` turns off the LED

D0–10, while the ESP8266 pin numbering starts from GPIO0–16.

To set a GPIO pin as output, you can use `pinMode` function. You can use either NodeMCU or ESP8266 pin numbering as its input parameter. The `pinMode` function is defined as

```
// pin : D0, D1, ... (NodeMCU pins) or
//       0, 1, ... (ESP8266 pins).
// mode: OUTPUT, INPUT, or INPUT_PULLUP.
void pinMode(pin, mode);
```

After the GPIO pin is initialized as output, you can write a value to the GPIO pin by using `digitalWrite` function. The `digitalWrite` function is defined as

```
// pin   : D0, D1, ... (NodeMCU pins) or
//         0, 1, ... (ESP8266 pins).
// value: LOW or HIGH.
void digitalWrite(pin, value);
```

The `pinMode` and `digitalWrite` function are defined in `core_esp8266_wiring_digital.c`. This is the ESP8266 library file. The ESP8266 library files can be found in `C:\Users\<USER>\AppData\`

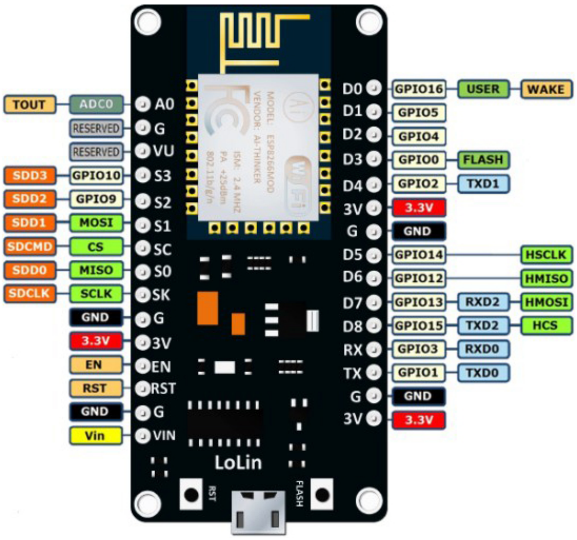


Figure 2.3. NodeMCU pinout. Retrieved August 26, 2018, from teachmemicro.com

`Local\Arduino15\packages\esp8266\hardware\esp8266\2.4.1\cores\esp8266`.

2.3 Example Program

In this example program, you will use a GPIO pin to blink the red LED. The code for the program is shown in Listing 2.1. The red LED is connected to pin D3. Before you can use a GPIO pin as output, you must configure it with `pinMode` function, as shown in line 8. In this line, the GPIO0 is configured as output by using NodeMCU pin numbering (D3).

LED blinking is created by turning on and off the LED every 1 second with `digitalWrite` and `delay` functions. The `digitalWrite` function in line 16 and 19 are used for writing a value to output pin. The `delay` function in line 17 and 20 are used for pausing the code execution for 1 second. The `delay` function is defined as

```
// value: Delay in milliseconds.
void delay(value);
```

The `delay` function is defined in `core_esp8266_wiring.c`.

What is the difference between `setup` and `loop` function?

The `setup` function runs once at the very beginning of the program, i.e. when you turn on the NodeMCU, reset the NodeMCU, or upload new code. The `loop` function runs over-and-over until the ESP8266 is reset. The `loop` function is just like the `while (1)` loop that runs forever.

Listing 2.1. Blink an LED

```
1  // *** File   : /esp8266-arduino/led/led.ino
2  // *** Author : Erwin Ouyang
3  // *** Date   : 17 Agt 2018
4
5  void setup()
6  {
7      // Set GPIO pin as output using NodeMCU pins,
8      pinMode(D3, OUTPUT);
9      // or use ESP8266 pins
10     //pinMode(0, OUTPUT);
11 }
12
13 void loop()
14 {
15     // Turn on LED
16     digitalWrite(D3, HIGH);
17     delay(1000);
18     // Turn off LED
```

```
    digitalWrite(D3, LOW);
    delay(1000);
}
```

2.4 Summary

There are two types of LED circuit, namely active-low and active-high. The `pinMode` function is used for configuring the GPIO pin. The `digitalWrite` function is used for writing a value to an output pin. The `delay` function is used for pausing the code execution.

Chapter 3

Dim an LED using PWM

In chapter 2, you have learned how to turn on and off the LED and blinking the LED. In this chapter, you will learn how to dim an LED. To dim an LED, you can use Pulse Width Modulation (PWM).

What will you learn in this chapter?

- Use `analogWrite` function to create PWM.
- Increase and decrease the LED brightness.

3.1 PWM

Pulse Width Modulation (PWM) is a square wave, a signal switched between `HIGH` and `LOW`, which the `HIGH` duration can be varied [2]. The `HIGH` duration is called pulse width, and can be varied from zero (fully `LOW`) up to the period of square wave (fully `HIGH`). The ratio of pulse width to the square wave period is called duty cycle as shown in Figure 3.1. The average voltage of PWM signal can simulate analog value between 0V to VDD (3.3V on ESP8266). The analog value is

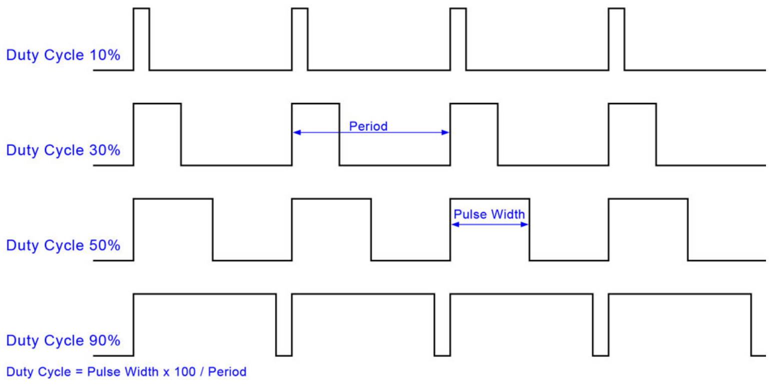


Figure 3.1. PWM duty cycle. Retrieved September 11, 2018, from caferacer-sjpg.com

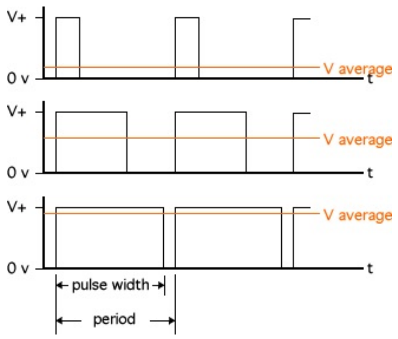


Figure 3.2. The average voltage of PWM signals. Retrieved September 11, 2018, from web.stanford.edu

actually the average voltage of the square wave that corresponds to duty cycle as shown in Figure 3.2. The larger the duty cycle, the larger the average voltage.

3.2 Analog Output

In Arduino, the PWM output is also called analog output. The PWM is implemented by software, and can be used on GPIO0–16 [3]. To write an analog value to a GPIO pin, you can use `analogWrite` function. The `analogWrite` function is defined as

```
// pin  : D0, D1, ... (NodeMCU pins) or
//        0, 1, ... (ESP8266 pins)
// value: 0 to PWMRANGE=1023 (default)
void analgWrite(pin, value);
```

The analog value can be in range from 0–`PWMRANGE`, which is defined as 1023 in `Arduino.h`. The range of analog value can be changed by using `analogWriteRange` function. The `analogWriteRange` function is defined as

```
// range: PWM maximum range
void analogWriteRange(range);
```

The PWM frequency is 1 kHz by default. The PWM frequency can be changed by using `analogWriteFreq` function. The `analogWriteFreq` function is defined as

```
// freq: PWM frequency.
void analogWriteFreq(freq);
```

The `analogWrite`, `analogWriteRange`, and `analogWriteFreq` functions are defined in `core_esp8266_wiring_pwm.c`.

3.3 Example Program

In this example program, you will use a GPIO pin to dim the red LED. The code for the program is shown in Listing 3.1. The red LED is connected to pin D3. When you use analog output, you do not have to initialize it using `pinMode` function, because it is already initialized by the Arduino library.

The code in the line 12–16 are used for increasing the LED brightness from 0–1023. The value is incremented by 20 every 25 milliseconds. The codes in the line 18–22 are used for decreasing the LED brightness from 1023–0. The value is decremented by 20 every 25 milliseconds.

Listing 3.1. Dim an LED using PWM

```
 1  // *** File   : /esp8266-arduino/led-pwm/led-pwm.ino
 2  // *** Author : Erwin Ouyang
 3  // *** Date   : 17 Agt 2018
 4
 5  void setup()
 6  {
 7  }
 8
 9  void loop()
10  {
11      // *** Increase brightness ***
12      for (int i = 0; i <= 1023; i += 20)
13      {
14          analogWrite(D3, i);
15          delay(25);
16      }
17      // *** Decrease brightness ***
18      for (int i = 1023; i >= 0; i -= 20)
19      {
20          analogWrite(D3, i);
21          delay(25);
22      }
23  }
```

3.4 Summary

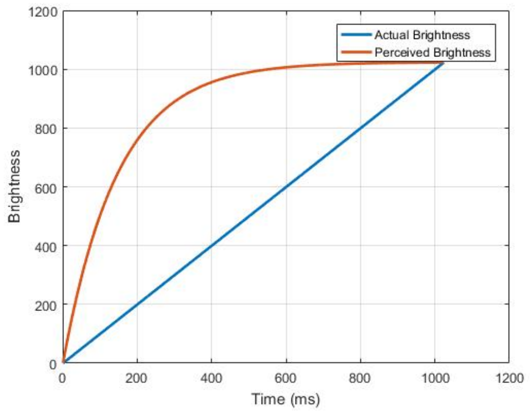
The analog output can be use to simulate an analog voltage, which can be varied from 0V–3.3V. The `analogWrite` function is used for writing an analog value to a GPIO pin.

3.5 Coding Challenge

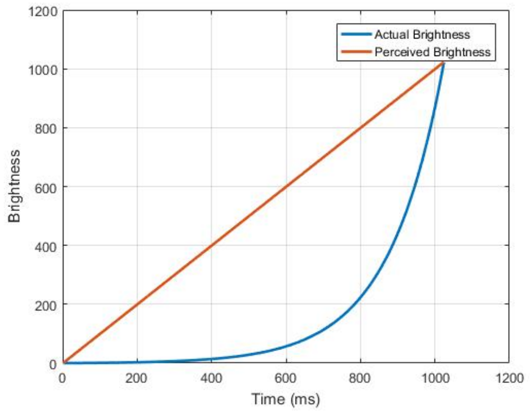
In the previous example, you may notice the red LED do not dim nicely. It appears to dim very quickly at first, and then spend a long time with almost full brightness. This happens because human eyes do not respond to light linearly. The led's actual brightness and eye's preceived brightness curve are shown in Figure 3.3(a).

To make the LED brightness to be appeared linearly to human eye, you can counteract the non-linear preceived brightness by using a curve as shown in Figure 3.3(b) (blue curve). You can create this curve by using equation 3.1 [4]. Where x is the linear curve (PWM value between 0–1023), and y is the counteract non-linear curve. In this coding challenge, you have to create a dimming program that use this counteract non-linear curve!

$$y = 2^{\frac{x}{102.3}} - 1 \tag{3.1}$$



(a)



(b)

Figure 3.3. LED dimming curve: (a) A linear LED dimming curve (blue) produces a non-linear preceived curve (orange); (b) A counteract non-linear LED dimming curve (blue) produces linear preceived curve (orange)

Chapter 4

Control an RGB LED

In chapter 3, you have learned how to use analog output to dim an LED. In this chapter, you will learn how to use analog output to control the color of RGB LED.

What will you learn in this chapter?

- Two different kinds of RGB LED.
- Control the color of the RGB LED using PWM.

4.1 RGB LED

RGB LED is just like the normal LED, but it has three LED with three different colors (red, green, and blue) within the package. With these primary colors of light, you can create other colors by varying the brightness of primary colors. To varying the brightness of RGB LED, you can use PWM.

There are two different kinds of RGB LED, namely common anode and common cathode. In common anode, the positive terminals of three

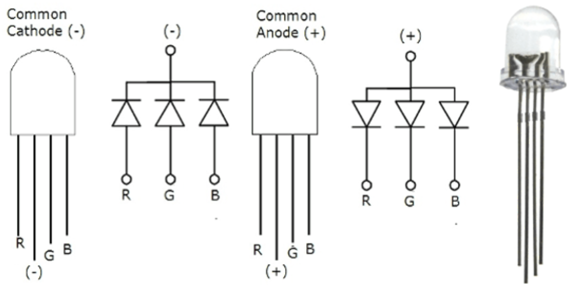


Figure 4.1. Common anode and common cathode RGB LED. Retrieved September 15, 2018, from www.hackster.io

LEDs are connected together, while in common cathode, the negative terminals of three LEDs are connected together as shown in Figure 4.1. In commom anode, the anode is connected to supply voltage, and the cathodes are connected to PWM output pins through resistors, so the LEDs are active-low. In common cathode, the cathode is connected to ground, and the anodes are connected to PWM output pins through resistors, so the LEDs are active-high.

4.2 Example Program

In this example program, you will use analog output to control the color of RGB LED. The code for the program is shown in Listing 4.1. In this example, a common anode RGB LED is used. The cathodes of red, green, and blue LEDs are connected to pin D5, D7, and D8, respectively.

In line 8, the `analogWriteRange` function is used for changing the PWM range from 0–255, so the range is equal to the 24-bit color. The 24-bit color uses 8-bit for each R, G, and B value. In line 39, there is a function called `rgbLedWrite`. This function is used for setting the color of RGB LED. The input parameters are the RGB value. In this function, the `analogWrite` function is called to write the RGB value to pin D5, D7, and D8. The values are inversed by subtracting the RGB value from 255 because the RGB LED is active-low. In main program, the color of RGB LED is changed from red to white with a delay of 1 second for every color.

Listing 4.1. Control an RGB LED

```
// *** File   : /esp8266-arduino/rgb-led/rgb-led.ino
// *** Author : Erwin Ouyang
// *** Date   : 17 Agt 2018

void setup()
{
    // Set PWM range from 0 to 255
    analogWriteRange(255);
}

void loop()
{
    // Red
    rgbLedWrite(255, 0, 0);
    delay(1000);
    // Orange
    rgbLedWrite(255, 50, 0);
    delay(1000);
    // Yellow
    rgbLedWrite(255, 150, 0);
    delay(1000);
    // Green
    rgbLedWrite(0, 255, 0);
    delay(1000);
    // Light Blue
    rgbLedWrite(0, 150, 255);
    delay(1000);
    // Blue
    rgbLedWrite(0, 0, 255);
    delay(1000);
    // Purple
    rgbLedWrite(150, 0, 255);
    delay(1000);
    // White
    rgbLedWrite(255, 255, 255);
    delay(1000);
}

void rgbLedWrite(byte red, byte green, byte blue)
{
    analogWrite(D5, 255-red);
    analogWrite(D7, 255-green);
    analogWrite(D8, 255-blue);
}
```

4.3 Summary

The `analogWriteRange` function is used for changing the PWM range to 8-bit. The `analogWrite` function is used for writing an analog value to the RGB pins of RGB LED.

4.4 Coding Challenge

In the previous example, you have created a program that changes the RGB LED color every 1 second. In this coding challenge, you have to create a program that gradually cycle the color spectrum, i.e start from red color, then the red color is gradually decremented, while at the same time the green color is also gradually incremented. Do this process until the RGB LED color is fully green. Through this process, the RGB LED color will gradually change from red, orange, yellow, and finally green. Do the same method for gradually cycling the entire color spectrum (except the white color)!

Chapter 5

Display a Message on OLED Display

In chapter 4, you have learned how to use RGB LED. In this chapter, you will learn how to use OLED display to display a message and a bitmap image.

What will you learn in this chapter?

- Install an external library on Arduino IDE.
- Use OLED SSD1306 display library.
- Display a message and a bitmap image on OLED display.

5.1 OLED Display

Organic LED (OLED) display is just like LCD display, but it uses organic component and no backlight. In this book, the popular 0.96 inch OLED with 128x64 pixels is used as shown in Figure 5.1. This OLED display uses SSD1306 chip as its controller. This OLED display

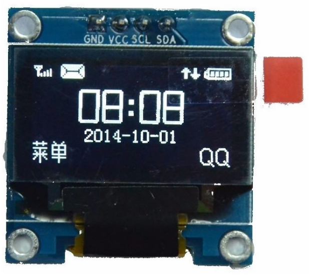


Figure 5.1. 0.96 inch 128x64 OLED display module. Retrieved September 15, 2018, from www.megaeshop.pk

has two types of interfaces, either I2C or SPI bus. In this book, the I2C bus is used. In general, the I2C bus is used for communication between ICs on a PCB. This bus only needs two wires which are for data and clock. Because of that, I2C bus is also called Two Wire Interface (TWI). In ESP8266, the I2C bus is handled by using `Wire.h` library.

5.2 OLED Library

To use OLED display with ESP8266, you need external libraries. There are two external libraries needed for the OLED display, namely Adafruit SSD1306 and Adafruit GFX Library. The step-by-step how to install these libraries is explained as the following:

- Download Adafruit SSD1306 from this link: **`https://github.com/adafruit/Adafruit_SSD1306`**
- Download Adafruit GFX Library from this link: **`https://github.com/adafruit/Adafruit-GFX-Library`**
- Go to **Sketch** → **Include Library** → **Add .ZIP Library...** menu, open the library file **`Adafruit_SSD1306-master.zip`**, and

then the **`Adafruit-GFX-Library-master.zip`**

After the libraries are installed, you have to setup the OLED display size in Adafruit SSD1306. The step-by-step how to setup the OLED display size is explained as the following:

- Go to Adafruit SSD1306 installation folder **`C:\Users\<USER>\Documents\Arduino\libraries\Adafruit_SSD1306`**.
- Open the **`Adafruit_SSD1306.h`** file.
- **Uncomment line 73** in order to set the OLED display size to 128x64, and then comment the other lines.

In Adafruit SSD1306 library, there is a memory buffer that holds the value of every OLED display pixel. The buffer is stored in ESP8266's RAM, so you should send the buffer to the OLED display periodically in order to update/refresh the display. This can be done by using `display` method[1]. You can use a method called `clearDisplay` to clear the buffer. After the buffer is cleared, you should call the `display` method.

5.3 Example Program

In the first example program, you will use the OLED display to display a message. The code for the program is shown in Listing 5.1. In line 16, the OLED's I2C address is defined. In line 19, the Adafruit OLED object is created, and it is initialized in line 24 by using `begin` method. The code in line 29–32, are used for setup the text size, color, location, and display the "Hello, world!" text. Finally, in line 35, you should send the buffer to the OLED display by using `display` method.

[1] A function that is created inside a class is called method. The term method is used almost exclusively in object-oriented programming.

Listing 5.1. Display a message on OLED display

```
// *** File   : /esp8266-arduino/oled-display/
//              oled-display.ino
// *** Author : Erwin Ouyang
// *** Date   : 17 Agt 2018

#include <Wire.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>

// *** Check library setting ***
#if (SSD1306_LCDHEIGHT != 64)    // 128 x 64 pixel display
#error("Height incorrect, please fix Adafruit_SSD1306.h!");
#endif

// OLED I2C address
#define OLED_ADDR 0x3C

// OLED object declaration
Adafruit_SSD1306 oled;

void setup()
{
    // *** Initialize and clear display ***
    oled.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR);
    // Clear OLED buffer
    oled.clearDisplay();

    // *** Display text ***
    oled.setTextSize(1);
    oled.setTextColor(WHITE);
    oled.setCursor(27, 30);
    oled.print("Hello, world!");

    // Show OLED buffer on the display
    oled.display();
}

void loop()
{
}
```

The `setTextSize` method is used for setting text size. The `setTextSize` method is defined as

```
// s: Text size, 1 is 6x8, 2 is 12x16, 3 is 18x24,
//    and so on
void Adafruit_GFX::setTextSize(uint8_t s)
```

The size 1 is the default size that has 6×8 pixels. The size 2 has 12×16 pixels. The size 3 has 24×32 pixels.

The `setTextColor` method has two definitions[2] which are defined as

```
// c: Text color. For monochrome OLED, the color
//    is only WHITE.
void Adafruit_GFX::setTextColor(uint16_t c)

// c: Text color. For monochrome OLED, the color
//    should be BLACK.
// b: Background color. For monochrome OLED,
//    the color should be WHITE.
void Adafruit_GFX::setTextColor(uint16_t c,
        uint16_t b)
```

The first `setTextColor` method is used for setting the text color with transparent background, while in the second `setTextColor` method, you can also set the text's background color. For example,

[2] Function/method overloading is a feature that allows you to define more than one function/method having the same name, but the input arguments or the return value are different.

you can call `setTextColor(BLACK, WHITE)` in order to set the text color to `BLACK` and the background color to `WHITE`.

The `setCursor` method is used for setting the X and Y coordinates in pixels. The `setCursor` method is defined as

```
// x: X coordinate in pixels
// y: Y coordinate in pixels
void Adafruit_GFX::setCursor(int16_t x,
        int16_t y)
```

The coordinate (0,0) is at the top left corner of the display.

In the second example program, you will use the OLED display to display bitmap images. The code for the program is shown in Listing 5.2. In line 19 and 39, there are two bitmap arrays for temperature and humidity symbols, respectively.

What do `static` and `PROGMEM` mean?

The `static` keyword has two cases. A static global variable or function is seen only in the file it is declared in. A static variable inside a function keeps the value when you call the function again, i.e. it behaves like a global variable. The `PROGMEM` keyword is used for storing data in program memory instead of RAM. This method is used for saving RAM space.

Listing 5.2. Display bitmap images on OLED display

```
1  // *** File   : /esp8266-arduino/oled-display-bitmap/
2  //              oled-display-bitmap.ino
3  // *** Author : Erwin Ouyang
4  // *** Date   : 17 Agt 2018
5
6  #include <Wire.h>
7  #include <Adafruit_SSD1306.h>
8  #include <Adafruit_GFX.h>
9
10 // *** Check library setting ***
```

```
#if (SSD1306_LCDHEIGHT != 64)     // 128 x 64 pixel display
#error("Height incorrect, please fix Adafruit_SSD1306.h!");
#endif

// OLED I2C address
#define OLED_ADDR 0x3C

// Temperature logo
static const unsigned char PROGMEM temperature_bmp[] =
{
    B00000011, B11001111,
    B00000110, B01100000,
    B00000100, B00101111,
    B00000100, B00100000,
    B00000100, B00101111,
    B00000101, B10100000,
    B00000101, B10101111,
    B00000101, B10100000,
    B00001101, B10110000,
    B00011011, B11011000,
    B00110111, B11101100,
    B00101111, B11110100,
    B00101111, B11110100,
    B00110111, B11101100,
    B00011000, B00011000,
    B00001111, B11110000
};
// Humidity logo
static const unsigned char PROGMEM humidity_bmp[] =
{
    B00000001, B10000000,
    B00000011, B11000000,
    B00000110, B01100000,
    B00000100, B00100000,
    B00001100, B00110000,
    B00001000, B00010000,
    B00011000, B00011000,
    B00010000, B00001000,
    B00110100, B00001100,
    B00101100, B00000100,
    B00101100, B00000100,
    B00101100, B00000100,
    B00110110, B00001100,
    B00011011, B00011000,
    B00001100, B00110000,
    B00000111, B11100000
};

// OLED object declaration
Adafruit_SSD1306 oled;

void setup()
{
    // *** Initialize and clear display ***
    oled.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR);
    // Clear OLED buffer
    oled.clearDisplay();

    // *** Display bitmap ***
    oled.drawBitmap(32, 16, temperature_bmp, 16, 16, 1);
    oled.drawBitmap(32, 36, humidity_bmp, 16, 16, 1);

    // *** Display text ***
    oled.setTextSize(2);
    oled.setTextColor(WHITE);
    oled.setCursor(55, 16);
    oled.printf("25%cC", (char)247);
    oled.setCursor(55, 36);
    oled.print("60%");

    // Show OLED buffer on the display
    oled.display();
}

```

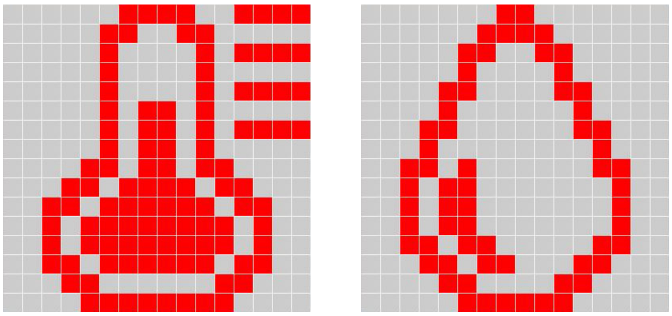


Figure 5.2. Bitmap symbols: (a) Temperature symbol; (b) Humidity symbol

```
85 void loop()
86 {
87 }
```

Every data bit in the bitmap array corresponds to every pixel of the symbol as shown in Figure 5.2. In line 70 and 71, the bitmap is drawn to the OLED buffer by using `drawBitmap` method. The `draw Bitmap` method is defined as

```
// x     : X coordinate in pixels
// y     : Y coordinate in pixels
// bitmap: Byte array with monochrome bitmap
// w     : Width of bitmap in pixels
// h     : Height of bitmap in pixels
void Adafruit_GFX::drawBitmap(int16_t x,
        int16_t y, uint8_t *bitmap, int16_t w,
        int16_t h, uint16_t color)
```

In line 74–79, the dummy value for temperature and humidity is printed by using `print` and `printf` methods.

5.4 Summary

In this chapter, you have learned how to install external libraries for OLED display. You have learned how to display a message and a bitmap image on OLED display. You can change the size, color, and location of the text by using `setTextSize`, `setTextColor`, and `setCursor` methods, respectively. The `drawBitmap` method is used for drawing a bitmap image.

Chapter 6

Read a Button

In chapter 1–5, you have learned how to use simple actuators, which are LED, RGB LED, and OLED display. In this chapter, you will learn how to read a button, which is a simplest sensor that produces digital output.

What will you learn in this chapter?

- Build a button circuit
- Set a GPIO pin as input.
- Read the button using `digitalRead` function.

6.1 Button Circuit

There are two types of button circuits, namely active-low and active-high. The active-low button circuit is shown in Figure 6.1. In this circuit, the VCC is connected to a GPIO pin through a resistor, so the current can flow to the GPIO pin. Therefore it receives a logic `HIGH`. When the button is pressed, the current flows through the button,

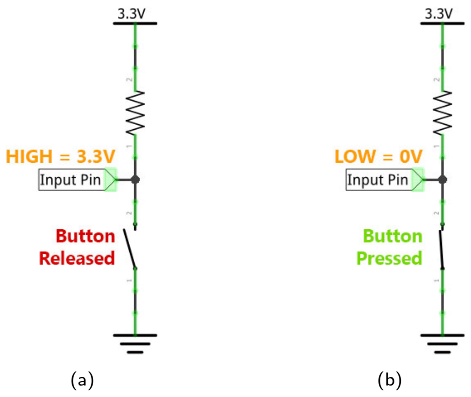


Figure 6.1. Active-low button circuit: (a) Button released, input pin receives a logic `HIGH`; (b) Button pressed, input pin receives a logic `LOW`

because the button resistance is almost zero, and the GPIO pin (when it is configured as input) has high impedance. Therefore the GPIO pin receives a logic `LOW`. The resistor in active-low button circuit is called pull-up resistor. The typical pull-up resistor value is 1–10kΩ.

On the other hand, the active-high button circuit is shown in Figure 6.2. In this circuit, the GND is connected to a GPIO pin through a resistor, so there is no current flows to the GPIO pin. Therefore it receives a logic `LOW`. When the button is pressed, the current flows from VCC to the GPIO pin and also to the resistor. Therefore the GPIO pin receives a logic `HIGH`. The resistor in active-high button circuit is called pull-down resistor. The typical pull-down resistor value is 1–10kΩ.

6.2 Digital Input

In order to read a button, you should configure a GPIO pin as input. You can configure the GPIO pin by using `pinMode` function. The

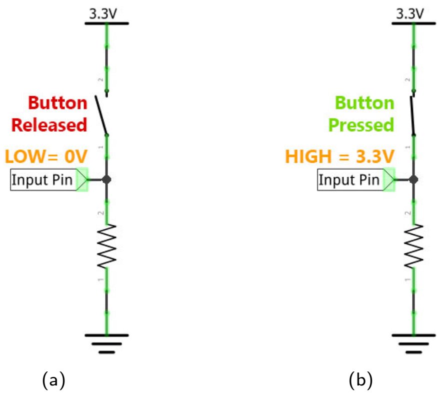


Figure 6.2. Active-high button circuit: (a) Button released, input pin receives a logic `LOW`; (b) Button pressed, input pin receives a logic `HIGH`

`pinMode` function is already defined in chapter 2 as

```
// pin : D0, D1, ... (NodeMCU pins) or
//       0, 1, ... (ESP8266 pins)
// mode: OUTPUT, INPUT, or INPUT_PULLUP
void pinMode(pin, mode);
```

After the GPIO pin is initialized as input, you can read the value of the GPIO pin by using `digitalRead` function. The `digitalRead` function is defined as

```
// pin    : D0, D1, ... (NodeMCU pins) or
//          0, 1, ... (ESP8266 pins)
// return: 0 or 1
int digitalRead(pin);
```

The `pinMode` and `digitalRead` functions are defined in `core_esp8266_wiring_digital.c`.

6.3 Example Program

In this example program, you will use the GPIO input to read the button state. The code for the program is shown in Listing 6.1. In line 8, pin D3 is initialized as output for LED. In line 10, pin D6 is initialized as input for button with internal pull-up resistor. The button circuit is active-low.

In main program, you can read the button state by using `digitalRead` function as shown in line 16. If the button is pressed, then the LED will be on, otherwise the LED will be off.

Listing 6.1. Read a button

```
 1  // *** File   : /esp8266-arduino/button/button.ino
 2  // *** Author : Erwin Ouyang
 3  // *** Date   : 17 Agt 2018
 4
 5  void setup()
 6  {
 7      // Set pin as output
 8      pinMode(D3, OUTPUT);
 9      // Set pin as input pull-up
10      pinMode(D6, INPUT_PULLUP);
11  }
12
13  void loop()
14  {
15      // *** Read button state ***
16      if (digitalRead(D6) == LOW)
17      {
18          // If button is pressed, then turn on
19          // the LED
20          digitalWrite(D3, HIGH);
21      }
22      else
23      {
        // If button is not pressed, then turn off
        // the LED
        digitalWrite(D3, LOW);
    }
}
```

6.4 Button Debouncing

Ideally, when you press the button, it should produce a clean transition from a logic `HIGH` to `LOW` (in case of an active-low button circuit), but in reality there are spurious transitions as shown in Figure 6.3. This happens because of the mechanical and physical issues. If the button is used as input to a counter, the counter will get multiple counts rather than the expected single count. [5]

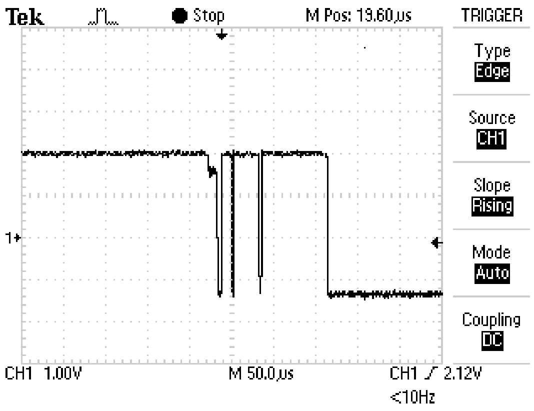


Figure 6.3. The output of an active-low button circuit when it is pressed. Retrieved October 14, 2018, from www.labbookpages.co.uk

To solve this bouncing problem, you can use either hardware or software solution. In this book, only the software solution is explained.

There are many software solutions to solve the bouncing problem. In this book, I use a debouncing counter. The button is read every 10 milliseconds. The debouncing counter counts from 0–4. The debouncing counter will reset the counts to 0 when there is a spurious transition. If the counts reach 4, then button state has reached the stable state.

The code for button debouncing is shown in Listing 6.2. This program use a button and an OLED display. The button is used as input to a counter `count` which is declared in line 24. The counter value is displayed on OLED display. You can compare the counter behaviour between the button that use debouncing and not use debouncing by commenting/uncommenting the line 16.

Listing 6.2. Debounce a button

```
 1  // *** File   : /esp8266-arduino/button-debouncing/
 2  //              button-debouncing.ino
 3  // *** Author : Erwin Ouyang
 4  // *** Date   : 17 Agt 2018
 5
 6  #include <Wire.h>
 7  #include <Adafruit_SSD1306.h>
 8  #include <Adafruit_GFX.h>
 9
10  // *** Check library setting ***
11  #if (SSD1306_LCDHEIGHT != 64)    // 128 x 64 pixel display
12  #error("Height incorrect, please fix Adafruit_SSD1306.h!");
13  #endif
14
15  // Comment this line to disable button debouncing
16  #define DEBOUNCE
17
18  // OLED I2C address
19  #define OLED_ADDR 0x3C
20
21  // OLED object declaration
22  Adafruit_SSD1306 oled;
```

```
// Counter
uint8_t count = 0;
// Debounced press
uint8_t deb_press = 0;

void setup()
{
    // Set pin as input pull-up
    pinMode(D6, INPUT_PULLUP);

    // *** Initialize and clear display ***
    oled.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR);
    // Clear OLED buffer
    oled.clearDisplay();

    // *** Display text ***
    oled.setTextSize(3);
    oled.setTextColor(WHITE);
    oled.setCursor(48, 24);
    oled.printf("%02d", count);
    oled.display();
}

void loop()
{
#ifndef DEBOUNCE
    if (digitalRead(D6) == LOW)
    {
        oled.clearDisplay();
        oled.setCursor(48, 24);
        oled.printf("%02d", ++count);
        oled.display();
    }
#else
    // *** Debouncing the button ***
    debounce();
    delay(10);

    if (deb_press == HIGH)
    {
        deb_press = LOW;
        oled.clearDisplay();
        oled.setCursor(48, 24);
        oled.printf("%02d", ++count);
        oled.display();
    }
#endif
}

void debounce()
{
    static uint8_t deb_count = 0;
    static uint8_t deb_state = 0;

    // Read and invert the active-low button state
    uint8_t read_state = !digitalRead(D6);

    // *** Debouncing the button state ***
    if (read_state != deb_state)
    {
        // If there is a transition then increment
        // the debouncing count
        deb_count++;
        if (deb_count >= 4)
        {
            // If the debouncing count reach 4
            // then update the debounced state
            deb_state = read_state;
            // If the debounced state is HIGH,
            // it means the button is pressed
            if (deb_state == HIGH)
                deb_press = HIGH;
            deb_count = 0;
        }
    }
    else
    {
        // Reset debouncing count
        deb_count = 0;
    }
}
```

In line 58 and 59, `debounce` function is called every 10 milliseconds. The `debounce` function is defined in line 72. In line 74 and 75, there are debouncing counter `deb_count` and debouncing button state `deb_state`. These variables are declared with static keyword, so the variables keep the value every time you call this function. In line 81–102, the debouncing counter is implemented. The debounced button press is stored in a global variable `deb_press`. In line 53 and 66, the counter value is printed to the OLED buffer. The counter value always has two characters, i.e if the counter value is only one digit (0–9), then a leading zero will be added. This is done by using `"%02d"` format.

What is the difference between `++count` and `count++`?

`++count` will increment the value, and then return the incremented value. `count++` will increment the value, but return the original value before being incremented.

6.5 Summary

There are two types of button circuits, namely active-low and active-high. On every GPIO pin, there is an internal pull-up resistor that can be configured for GPIO input. The `digitalRead` function is used for reading the state of a GPIO pin. Button debouncing is a method for avoiding multiple button press when you press the button.

Chapter 7

Read a Button using External Interrupt

In chapter 6, you have learned how to read a button. In this chapter, you will learn how to read a button using external interrupt.

What will you learn in this chapter?

- Interrupt concept.
- Use `attachInterrupt` function.
- Read a button inside an interrupt function.

7.1 Interrupt

Interrupt is a signal to the CPU emitted by an internal peripheral or an external device indicating that the internal peripheral or external device needs immediate attention [6]. There is an Interrupt Service Routine (ISR) function that is executed every Interrupt Request (IRQ). When an interrupt occurs, the CPU suspends the execution of main

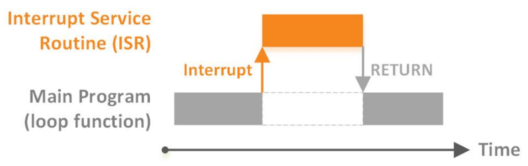


Figure 7.1. An Interrupt Service Routine (ISR) is called when interrupt occurs.

program (`loop` function), and executes the ISR function as shown in Figure 7.1. The concept of interupt can also be explained by a simple code as the following:

```
void loop()
{
    digitalWrite(D3, HIGH);
    delay(2000);
    digitalWrite(D3, LOW);
    delay(2000);

    if (digitalRead(D6) == LOW)
        // Do something
    else
        // Do something else
}
```

In this program, the LED blinks every 2 seconds, and then reads the button state. If you run this program, the button works, but the response time is slow. This happens because the CPU executes the `digitalRead` function every 4 seconds (because of the `delay` function).

To solve this problem, you can use an interrupt. You can place the `digitalRead` function in the ISR function, so when the interrupt is occured, the execution of main program is suspended in order to executes the ISR. After the ISR is executed, then the execution of main program is resumed.

There are two types of interrupts, namely internal interrupt and external interrupt. Internal interrupt comes from internal peripherals such as timer, UART, SPI, I2C, etc. External interrupt come from external devices such as button.

7.2 Example Program

In this example program, you will read the button by using external interrupt. The code for the program is shown in Listing 7.1. To use external interrupt, you should initialize a GPIO input with `attach Interrupt` function. The `attachInterrupt` function is defined as

```
// pin : Any GPIO pin, except D0 (GPIO16)
// ISR : The ISR to call when the interrupt occurs
// mode: RISING, FALLING, or CHANGE
void attachInterrupt(digitalPinToInterrupt(pin),
        ISR, mode);
```

There are three modes of external interrupts, namely rising, falling, and change, as shown in Figure 7.2. In rising mode, an interrupt occurs when the input value changes from a logic `LOW` to `HIGH`. In falling mode, an interrupt occurs when the input value changes from a logic `HIGH` to `LOW`. The change mode is the combination of rising and falling mode.

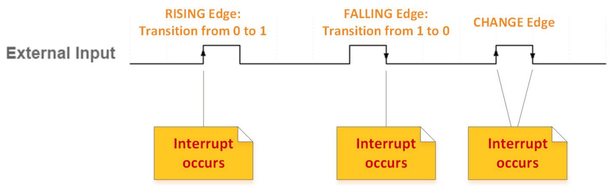


Figure 7.2. Three modes of external interrupts.

The external interrupt can be attached on any GPIO pin, except D0 (GPIO16). In line 11, pin D6 is initialized as input with internal pull-up resistor for the button. In line 14, an ISR function called `isr` is attached to pin D6 with change mode. The ISR function is created in line 20. The interrupt occurs when the input value goes from a logic `HIGH` to `LOW` or from a logic `LOW` to `HIGH`. The input value changes from a logic `HIGH` to `LOW` when the button is pressed, and it changes from a logic `LOW` to `HIGH` when the button is released.

Listing 7.1. Read a button using external interrupt

```
1  // *** File   : /esp8266-arduino/button-interrupt/
2  //              button-interrupt.ino
3  // *** Author : Erwin Ouyang
4  // *** Date   : 17 Agt 2018
5
6  void setup()
7  {
8      // Set pin as output
9      pinMode(D3, OUTPUT);
10     // Set pin as input pull-up
11     pinMode(D6, INPUT_PULLUP);
12     // Set external interrupt from pin D6 that is
13     // connected to button
14     attachInterrupt(digitalPinToInterrupt(D6),
15             isr, CHANGE);
16     // Set PWM range from 0 to 255
17     analogWriteRange(255);
18 }
19
20 void isr()
21 {
22     // *** Read button state ***
23     if (digitalRead(D6) == LOW)
24     {
25         // If button is pressed, then turn on the LED
26         digitalWrite(D3, HIGH);
```

```
    }
    else
    {
        // If button is not pressed, then turn off the LED
        digitalWrite(D3, LOW);
    }
}

void loop()
{
    // *** Blink LED white ***
    rgbLedWrite(255, 255, 255);
    delay(2000);
    rgbLedWrite(0, 0, 0);
    delay(2000);
}

void rgbLedWrite(byte red, byte green, byte blue)
{
    analogWrite(D5, 255-red);
    analogWrite(D7, 255-green);
    analogWrite(D8, 255-blue);
}
```

In the `isr` function, the button state is read by using `digitalRead` function. When the button is pressed, the red LED will be on, otherwise the red LED will be off. In main program, you can blink the RGB LED every 2 seconds. The timing diagram of this program is shown in Figure 7.3. The execution of main program is preempted when the button interrupt occurs, and resumed after the CPU executes button ISR.

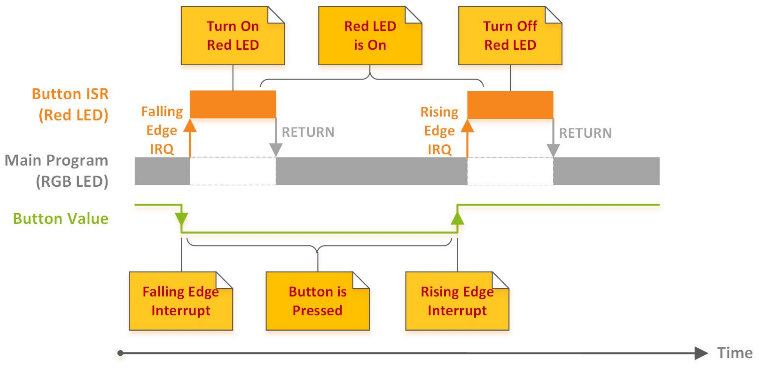


Figure 7.3. Timing diagram of the button interrupt program.

7.3 Summary

By using external interrupt, the button response time is faster than without interrupt. This happens because interrupt can preempt the execution of main program in order to serve the request from button.

Chapter 8

Read a Trimpot using ADC

In chapter 6 and 7, you have learned how to read a digital value from a button. In this chapter, you will learn how to read an analog value from a trimpot using Analog-to-Digital Converter (ADC).

What will you learn in this chapter?

- Read a trimpot using `analogRead` function.
- Display the trimpot value on OLED display.

8.1 ADC

Analog-to-Digital Converter (ADC) is a component that converts an analog voltage to a digital value. In mathematics, it is said that the analog voltage consists of an infinite number of points between point A and point B, for example between 0–5V. In Figure 8.1, you can see that after 2.7V there are 2.71V, 2.718V, 2.7182V, and so on, but a microcontroller can only represent a finite set of numbers (discrete numbers). For example, 8-bit number can only represent numbers

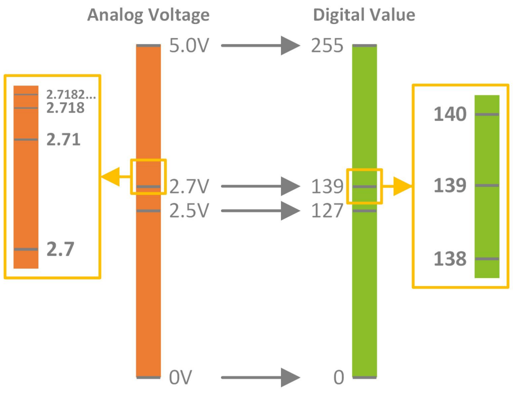


Figure 8.1. Conversion between an analog voltage to a digital value.

from 0–255, so between 139 and 140 there is no other number.

The function of an ADC is to map the infinite number of points (e.g. between 0–5V) to the finite number of points (e.g. between 0–255). This process is called quantization. When the analog voltage is 2.71V then it will be represented as 139. The analog voltage is rounded to the nearest value of the digital number. The ADC converts the analog voltage to the digital value periodically. This process is called sampling. The ADC period or sampling rate determines the bandwidth of the signal that can be captured.

The ESP8266's ADC has 10-bit resulotion, and the input voltage range is between 0–1V. The NodeMCU V3 board has a voltage divider circuit, so the input voltage range on pin A0 is between 0–3.3V. In older version of NodeMCU, the input voltage range on pin A0 may be between 0–1V. If you are not sure, you can check the input voltage range by inputting 0.5V to the pin A0, then read the value using `analogRead` function. If the result is around 512, then the input range is between 0–1V, otherwise the input range is between 0–3.3V.

8.2 Analog Input

When you use the ESP8266 Arduino library, the ADC is already initialized, so you just need to use the `analogRead` function for reading the analog voltage. The `analogRead` function is defined as

```
// pin   : A0
// return: ADC value
int analogRead(pin);
```

The parameter input is the analog pin A0, because the ESP8266 only has one ADC input. The function returns analog value between 0–1023. The `analogRead` function are defined in `core_esp8266_wiring_analog.c`.

8.3 Example Program

In this example program, you will use the ADC to read a trimpot. The code for the program is shown in Listing 8.1. The output of the trimpot is an analog voltage between 0–3.3V. The analog voltage is converted to a digital representation between 0–1023 by using `analogRead` function as shown in line 33, and then the value is stored in variable `trimpotValue`.

The code in line 36–42 is used for displaying the `trimpotValue` on OLED display. In line 44, there is a delay of 1 second, so the `trimpotValue` is updated every 1 second. The `print` method from the OLED library can be used for printing a string or an integer, as shown in line 40 and 41.

Listing 8.1. Read a trimpot using ADC

```
1  // *** File   : /esp8266-arduino/trimpot/trimpot.ino
2  // *** Author : Erwin Ouyang
```

```
// *** Date   : 17 Agt 2018

#include <Wire.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>

// *** Check library setting ***
#if (SSD1306_LCDHEIGHT != 64)    // 128 x 64 pixel display
#error("Height incorrect, please fix Adafruit_SSD1306.h!");
#endif

// OLED I2C address
#define OLED_ADDR 0x3C

// OLED object declaration
Adafruit_SSD1306 oled;

int trimpotValue;

void setup()
{
    // *** Initialize and clear display ***
    oled.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR);
    // Clear OLED buffer
    oled.clearDisplay();
}

void loop()
{
    // Read analog value
    trimpotValue = analogRead(A0);

    // *** Display timpot value ***
    oled.clearDisplay();
    oled.setTextSize(1);
    oled.setTextColor(WHITE);
    oled.setCursor(27,30);
    oled.print("Trimpot: ");
    oled.print(trimpotValue);
    oled.display();

    delay(1000);
}
```

8.4 Summary

The ADC is used for reading an analog voltage. The ADC quantizes and samples the analog voltage to produce the digital value. To use ESP8266's ADC, you can just call the `analogRead` function.

8.5 Coding Challenge

In this coding challenge, you should read a trimpot, and then use the trimpot value as input for cycling an RGB LED through the color spectrum. This is like the coding challenge in chapter 4, but instead of three input (RGB value), you should use one input (trimpot value).This can be done by using HSV to RGB algorithm [7]. Given an HSV color with hue $0° \leq H < 360°$, saturation $0 \leq S \leq 1$, and value $0 \leq V \leq 1$. The R, G, and B values can be calculated by using the following equation:

$$C = V \times S \tag{8.1}$$

$$H' = \frac{H}{60°} \tag{8.2}$$

$$X = C \times (1 - |H' \ mod \ 2 - 1|) \tag{8.3}$$

$$(R', G', B') = \begin{cases} (C, X, 0) & \text{if } 0^\circ \leq H < 60^\circ \\ (X, C, 0) & \text{if } 60^\circ \leq H < 120^\circ \\ (0, C, X) & \text{if } 120^\circ \leq H < 180^\circ \\ (0, X, C) & \text{if } 180^\circ \leq H < 240^\circ \\ (X, 0, C) & \text{if } 240^\circ \leq H < 300^\circ \\ (C, 0, X) & \text{if } 300^\circ \leq H < 360^\circ \end{cases} \tag{8.4}$$

$$m = V - C \tag{8.5}$$

$$(R, G, B) = ((R' + m) \times 255, (G' + m) \times 255, (B' + m) \times 255) \tag{8.6}$$

For this coding challenge, the trimpot value is used as input for H value, while the S and V values are set to 1, so you have only one input which is the trimpot value. By using equation 8.1–8.6, you can calculate the R, G, and B values. Create a function for implemeting this algorithm, and send the RGB value to the RGB LED!

Chapter 9

Serial Communication between ESP8266 and PC

In chapter 6–8, you have learned how to use simple sensors, which are button and trimpot. In this chapter, you will learn how to use serial communication. Serial communication can be used for sending or receiving data to or from PC. For example, you can send the trimpot value to PC instead of displays it on the OLED display.

What will you learn in this chapter?

- Setup the serial communication.
- Send a message to PC using `print` or `println` method.

9.1 Serial Communication

Serial communication is the process of sending data one bit at a time, serially[8]. There are many protocols that send data serially such as SPI, I2C, USB, etc. The term serial in serial communication refers to

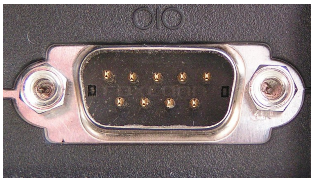


Figure 9.1. DB9 connector for serial port. Retrieved September 26, 2018, from www.wikipedia.com

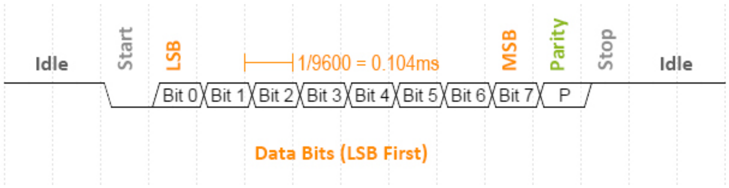


Figure 9.2. UART data frame.

Universal Asynchronous Receiver Transmitter (UART) protocol. This protocol is widely used for communication between device and PC. In PC, the serial communication is called RS232 or serial port, which uses DB9 connector as shown in Figure 9.1. In modern PC, the DB9 connector usually not avaliable, but the serial communication can still be used through the USB port. The software in PC can access the serial port through the COM interface. In Arduino IDE, there is a tool for accessing the serial port, namely Serial Monitor. There is also a tool called Serial Plotter. This tool is useful when the data are numbers, and you want to plot the data in real-time.

The UART protocol uses two wires for sending and receiving data, namely TX and RX. The UART data frame is shown in Figure 9.2. Each of the data frame consists of 5–9 data bits with LSB first configuration. The commonly used data bits is 8 bits or 1 byte. There are start bit and stop bit which are used for indicating where the first and the last data bits. The start bit is 0, and the stop bit is 1. The length of the stop bit can be configured either 1, 1.5, or 2 bits. The parity bit is an optional bit that can be used for detecting error. There are two types of parity bits, namely even and odd parity. There are several standards for baud rate or data speed such as 1200, 2400, 4800, 19200, 38400, 57600, and 115200 bps.

9.2 Serial Print

To use serial communication, you should initilize the transmision speed (baud rate) by using `begin` method. This method is defined in a class named `HardwareSerial`. The `HardwareSerial` class is defined in `HardwareSerial.h`. This class is instantiated as an object called Serial. Hence, you should call `Serial.begin`. The most commonly used `begin` method is defined as

```
// baud: Use one of these rates: 300, 600, 1200,
//       2400, 4800, 9600, 14400, 19200, 28800,
//       38400, 57600, or 115200
void begin(unsigned long baud);
```

After the serial communication is initialized, you can send a message through the serial communication by using `print` or `println` methods. The `println` method adds a carriage return and a newline character (`\r\n`) at the end of the message. The `print` and `println` methods can be used for all data types which are `String`, `char`, `int`, `long`, and `double`.

9.3 Example Program

In this example program, you will use serial communication for sending a message to PC. The code for the program is shown in Listing 9.1. In line 8, the serial communication is initialized with the baud rate of

9600 bps. In line 10, a "Hello, world!" message is sent to the serial communication.

Listing 9.1. Send a message to PC

```
1  // *** File   : /esp8266-arduino/serial/serial.ino
2  // *** Author : Erwin Ouyang
3  // *** Date   : 17 Agt 2018
4  
5  void setup()
6  {
7      // Setup serial communication
8      Serial.begin(9600);
9      // Print text
10     Serial.println("Hello, world!");
11 }
12 
13 void loop()
14 {
15 }
```

To open Serial Monitor, you can go to **Tools** → **Serial Monitor** menu. If you do not see the "Hello, world!" message in the Serial Monitor, then you should open the Serial Monitor first before you upload the program to the ESP8266 or after the program is uploaded, you can press the NodeMCU's reset button. This happens because the "Hello, world!" message is sent only once after you upload the program to the ESP8266, i.e. because the "Hello, world!" message is placed inside the `setup` function. You can place the "Hello, world!" message inside the `loop` function, so the message is sent repeatedly.

What are garbage characters?

When you reset the NodeMCU board, you probably see garbage characters in Serial Monitor. These characters are appeared as garbage because the baud rate between transmitter (ESP8266) and receiver (Serial Monitor) is mismatch. Actually these characters are messages from ESP8266's bootloader. The bootloader uses baud rate of 74880 bps. You can see the bootloader messages by changing the baud rate to 74880.

9.4 Summary

Serial communication is the simplest protocol for communication between ESP8266 and PC. The most commonly used methods in serial communication are `print` and `println`.

Chapter 10

Data Logging to PC using Serial

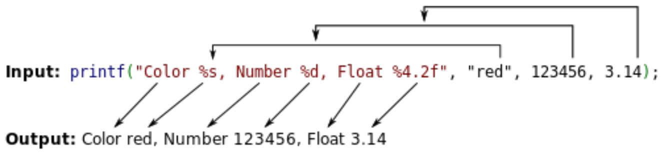
In chapter 9, you have learned how to use serial communication for sending a message to PC. In this chapter, you will learn how to use serial communication for logging ADC data to PC.

What will you learn in this chapter?

- Send the trimpot value to PC using `printf` method.
- Plot the trimpot value using Serial Plotter tool.

10.1 Print Format

In C programming language, there is a function called `printf`, which stands for "print formatted". The `printf` function uses format specifiers to print formatted output to console. Format specifiers start with a % character, and indicate the location to translate a piece of data (char, int, float, etc.) to characters. The example of `printf` func-



```
Input: printf("Color %s, Number %d, Float %4.2f", "red", 123456, 3.14);

Output: Color red, Number 123456, Float 3.14
```

Figure 10.1. The example of `printf` function. Retrieved October 15, 2018, from www.wikipedia.com

Table 10.1. The commonly used format specifiers [9]

Specifier	Output	Example
%d or %i	Signed decimal integer	392
%x	Unsigned hexadecimal integer	7fa
%X	Unsigned hexadecimal integer (uppercase)	7FA
%o	Unsigned octal	610
%f	Decimal floating point	392.65
%c	Character	a
%s	String of characters	sample

Table 10.2. The example of commonly used sub-specifiers [9]

Sub-specifier	Output	Example
%08d	Preceding with zeros	00001977
%#x	Preceding with 0x	0x7fa
%.2f	Number of digit after decimal point	3.14

tion is shown in Figure 10.1. The commonly used format specifiers are shown in Table 10.1. The specifiers can also contain sub-specifiers, which are optional. The example of commonly used sub-specifiers are shown in Table 10.2. In Arduino library, the `printf` function is added to the `Serial` class, so you can use the `printf` method for sending data through serial communication.

How to print a number in binary?

The `printf` method is only able to print a number in base 8 (octal), 10 (decimal), and 16 (hexadecimal), so there is no specifier for binary integer. Alternatively, you can use the `print` or `println` method, which has an optional second parameter specifies the base to use. The optional second parameters are `BIN`, `OCT`, `DEC`, and `HEX`. For example, `Serial.print(78, BIN)` gives "1001110".

10.2 Example Program

In this example program, you will read a trimpot using `analogRead` function, then send the value to the PC using `printf` method. The code for the program is shown in Listing 10.1. In line 17, the trimpot is read, and then in line 19, the value is sent to the PC. The trimpot value is formatted with an escape sequence `\n`, which is a newline character. There are several commonly used escape sequences as shown in Table 10.3.

Listing 10.1. Log the trimpot value to PC

```
1  // *** File   : /esp8266-arduino/serial-data-logger/
2  //              serial-data-logger.ino
3  // *** Author : Erwin Ouyang
4  // *** Date   : 17 Agt 2018
5  
6  int trimpotValue;
7  
8  void setup()
9  {
10     // Setup serial communication
11     Serial.begin(74880);
12 }
13 
14 void loop()
```

Table 10.3. The commonly used escape sequences

Escape Sequence	Character Represented
\b	Backspace
\n	Newline
\r	Carriage return
\t	Horizontal tab
\\	Backslash
\'	Single quotation mark
\"	Double quotation mark

```
15 {
16     // Read analog value
17     trimpotValue = analogRead(A0);
18     // Send trimpot value to serial monitor
19     Serial.printf("%d\n", trimpotValue);
20     delay(1000);
21 }
```

10.3 Serial Plotter

To display the trimpot value, you can use the Serial Monitor, but there is another tool useful for displaying variables, namely Serial Plotter. With Serial Plotter, you can plot the trimpot value as shown in Figure 10.2. You can visualize the trimpot value as a waveform in real-time. To open Serial Plotter, you can go to **Tools** → **Serial Plotter** menu.

To plot one variable in Serial Plotter, you just need to use `printf` method with a newline character at the end of the value as in line 19. You can plot multiple variables in Serial Plotter by separating each of the variable with a space or a comma, and end with newline. The example code how to plot three variables is shown in the following code:

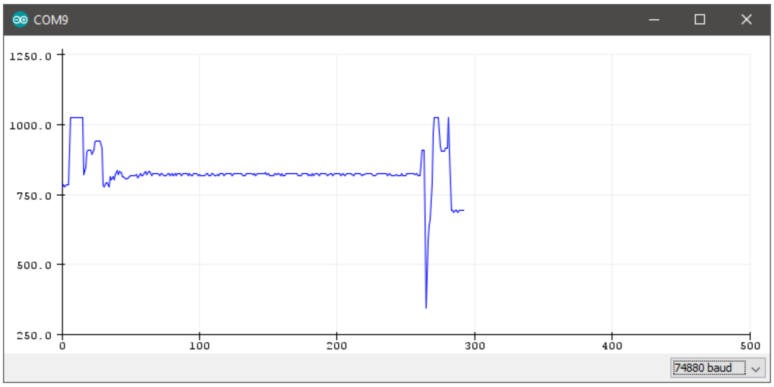


Figure 10.2. Serial Plotter tool.

```
// Using spaces as the separator, or
Serial.printf("%d %d %d\n", var1, var2, var3);
// using commas as the separator
Serial.printf("%d,%d,%d\n", var1, var2, var3);
```

10.4 Summary

Data logging using serial communication is the simplest way to display value from trimpot or other sensors. You can also visualize a variable as a waveform in real-time using Serial Plotter tool.

Chapter 11

Receive Data from PC using Serial

In chapter 9 and 10, you have learned how to send data to PC using serial communication. In this chapter, you will learn how to use serial communication for receiving data from PC.

What will you learn in this chapter?

- Receive a message from PC using `readString` and `readStringUntil` methods.
- Control the LED from PC using Serial Monitor tool.

11.1 Serial Read

There are several methods that will be used for receiving data from PC using serial communication, namely `available`, `readString`, and `readStringUntil`. The `available` method returns the number of data bytes available to read from the receive buffer (which holds 64 bytes). The `available` method is defined in `HardwareSerial.h`. The `available` method is defined as

```
// return: The number of bytes available to read
int available(void);
```

The `readString` method returns a string read from the receive buffer. The `readString` method is defined in `Stream.h`. The `readString` method is defined as

```
// return: A string read from the receive buffer
String readString();
```

The `readStringUntil` method returns the entire string read from the receive buffer, until the terminator character is detected. The `readStringUntil` method is defined in `Stream.h`. The `readStringUntil` method is defined as

```
// terminator: The character to search for (char)
// return    : The entire string read from
//              the receive buffer, until the
//              terminator character is detected
String readStringUntil(char terminator);
```

How to read a number using serial communication?

A number can be received from PC as a string, then it is converted to an `int` or a `float` by using `toInt` or `toFloat` method, repectively.

11.2 Example Program

In the first example program, you will read a message from PC using `readString` method. The code for the program is shown in Listing 11.1. In line 17, the available data bytes in the receive buffer are checked by using `available` method. If the data bytes are available, then the data bytes are read from the receive buffer by using `readString` method. The data bytes are stored in a variable called `serialInput`. The data bytes in the variable `serialInput` are sent back to the PC by using `println` method. To test the program, you can open the Serial Monitor, and then you can send a message to ESP8266. The message will be echoed back to the Serial Monitor.

Listing 11.1. Receive a message from PC

```
// *** File   : /esp8266-arduino/serial-receive-data/
//              serial-receive-data.ino
// *** Author : Erwin Ouyang
// *** Date   : 17 Agt 2018

String serialInput;

void setup()
{
    // Setup serial communication
    Serial.begin(74880);
}

void loop()
{
    // If there is data in receive buffer
    if (Serial.available() > 0)
    {
        // Read string from receive buffer
        serialInput = Serial.readString();
        // Print to serial monitor
        Serial.println(serialInput);
    }
}
```

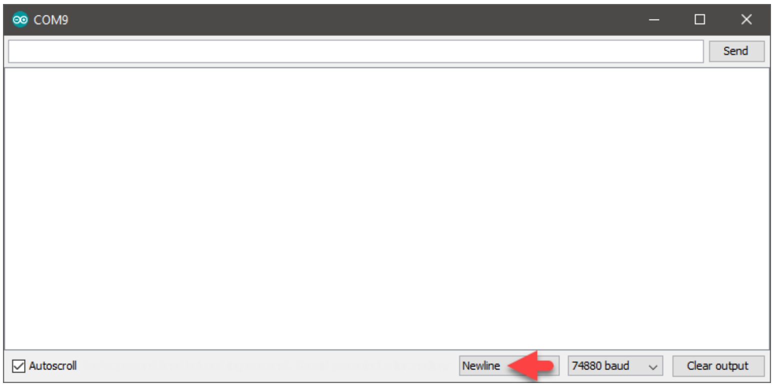
In the second example program, you will create a program for turning on or off the LED from PC using serial communication. The code for the program is shown in Listing 11.2. In main program, the data bytes are read by using `readStringUntil` method as shown in line 21. This method will read a string, which is a command for turning on or off the LED from PC. Every command from Serial Monitor will be sent with a newline ending character.

Listing 11.2. Turn on or off the LED using serial

```
1  // *** File   : /esp8266-arduino/serial-led/
2  //              serial-led.ino
3  // *** Author : Erwin Ouyang
4  // *** Date   : 17 Agt 2018
5
6  String cmd;
7
8  void setup()
9  {
10     // Set pin as output
11     pinMode(D3, OUTPUT);
12     // Setup serial communication
13     Serial.begin(74880);
14 }
15
16 void loop()
17 {
18     if (Serial.available() > 0)
19     {
20         // Read command from receive buffer
21         cmd = Serial.readStringUntil('\n');
22
23         // *** Process the command ***
24         if (cmd == "LED+ON")          // Turn on LED
25         {
26             digitalWrite(D3, HIGH);
27         }
28         else if (cmd == "LED+OFF")   // Turn off LED
29         {
30             digitalWrite(D3, LOW);
31         }
32         else if (cmd == "LED")      // Ask the LED value
33         {
34             if (digitalRead(D3))
35                 Serial.println("LED is on");
36             else
37                 Serial.println("LED is off");
38         }
39         else
40         {
41             Serial.println("Unknown command");
42         }
43     }
44 }
```

There are three commands defined for controlling the LED, namely `LED+ON`, `LED+OFF`, and `LED`. The `LED+ON` command is used for turning on the LED. The `LED+OFF` command is used for turning off the LED. The `LED` command is used for asking the current LED state. The commands are processed by using the `if` statement as shown in line 24–42. To turn on or off the LED, you can use the `digitalWrite` function as shown in line 26 and 30. To read the state of the LED, you can use the `digitalRead` function as shown in line 34. To test the program, you can send the command (`LED+ON`, `LED+OFF`, or `LED`) from the Serial Monitor. The ending character of the Serial Monitor must be set to **Newline** as shown in Figure 11.1.

In the third example program, you will create a program for changing the brightness of the LED from PC using serial communication. The code for the program is shown in Listing 11.3. In main program, the data bytes are read by using `readStringUntil` method. The data bytes are the PWM value, and it is stored in a variable called `pwmValue`. The variable `pwmValue` is a string, so it must be converted to an integer, in order to be used as input for `analogWrite` function. The variable `pwmValue` is converted to an integer by using `toInt`



☑ Autoscroll   Newline   74880 baud

Figure 11.1. Newline ending in Serial Monitor tool.

method as shown in line 21. To test the program, you can send a number between 0–1023 from the Serial Monitor.

Listing 11.3. Change the LED brightness using serial

```
// *** File   : /esp8266-arduino/serial-led-pwm/
//              serial-led-pwm.ino
// *** Author : Erwin Ouyang
// *** Date   : 17 Agt 2018

String pwmValue;

void setup()
{
    // Setup serial communication
    Serial.begin(74880);
}

void loop()
{
    if (Serial.available() > 0)
    {
        // Read PWM value from receive buffer
        pwmValue = Serial.readStringUntil('\n');
        // Process the command
        analogWrite(D3, pwmValue.toInt());
    }
}
```

11.3 Summary

To receive data from PC, you can use the `readString` or `readStringUntil` methods. You can use the serial communication for controlling the LED or other actuators from PC by defining your own commands.

11.4 Coding Challenge

In this coding challenge, you should modify the program in Listing 11.2 by adding two new commands for setting the color of the RGB LED and displaying a message on OLED display. The command for RGB LED should follow this format: `RGB+<red>,<green>,<blue>`. The command for OLED display should follow this format: `OLED+<len>+<message>`, where `len` is the number of characters of the message.

In C library, there is a function called `strtok`. This function is used for splitting a string into tokens. You can use this function for splitting the RGB LED and OLED command. The example how to use this function is shown in the following code:

```
String str_s = "Split+string,into,tokens.";
char str_c[64];
char *chr_p;

// Convert string to array of char
```

```
    str_s.toCharArray(str_c, 64);

    // *** Split string into tokens ***
    chr_p = strtok(str_c, "+,.");
    while (chr_p != NULL)
    {
        Serial.printf("%s\n", chr_p);
        chr_p = strtok(NULL, "+,.");
    }
```

By using `strtok` function, split the RGB LED and OLED display commands, and then procces the commands!

Chapter 12

Measure Time using a Timer

In chapter 9–11, you have learned how to use serial communication for sending and receiving data to and from PC. In this chapter, you will learn how to measure time using a timer.

What will you learn in this chapter?

- Measure time using `millis` or `micros` function.
- Blink an LED without using `delay` function.

12.1 Timer

A timer is one of the basic peripherals that almost all microcontrollers have it. The easiest way how to use the ESP8266's timer is by using `millis` or `micros` function. The `millis` and `micros` functions return the number of milliseconds and microseconds since the program started, respectively. The `millis` function is defined as

```
// return: Number of milliseconds since
//         the program started
```

```
unsigned long millis()
```

The `micros` function is defined as

```
// return: Number of microseconds since
//         the program started
unsigned long micros()
```

For more information about the `millis` and `micros` functions, you can see in core_esp8266_wiring.c.

12.2 Example Program

In the first example program, you will use the `millis` function to read the current milliseconds value. The code for the program is shown in Listing 12.1. In line 16, the current milliseconds value is read, and then it is stored in a variable called `ms`. The milliseconds value is printed to the Serial Monitor by using `println` function every 1 second. The milliseconds value printed on the Serial Monitor is incremented every 1000 because of the 1 second delay.

Listing 12.1. Measure time using `millis` function

```
1  // *** File   : /esp8266-arduino/timer/timer.ino
2  // *** Author : Erwin Ouyang
3  // *** Date   : 17 Agt 2018
4
5  unsigned long ms;
6
7  void setup()
8  {
9      // Setup serial communication
10     Serial.begin(74880);
11 }
12
13 void loop()
```

```
{
    // Get milliseconds value
    ms = millis();
    // Send milliseconds value to serial monitor
    Serial.println(ms);
    delay(1000);
}
```

In the second example program, you will blink an LED without using `delay` fucntion. This method is useful when you need to do more than one thing at once. For example, you need to blink an LED and also read a button. In this case, if you use the `delay` function, then the button response is slow. This happens because the CPU needs to wait until the `delay` is passed. To overcome this problem, you can use the external interrupt as in chapter 7, but in this example, you will use another method, which is the timer millis. The code for the program is shown in Listing 12.2.

In main program, there are codes for blinking the red LED and for reading the button state. For blinking the LED, you need two variables called `now` and `last`. These variables are used for storing the milliseconds values. In line 23, the current milliseconds value is read and stored in variable `now`. On every loop, the variable `now` is checked by subtracting it with the variable `last`, which is start from 0. If the result is larger than 1000, then 1000 milliseconds have passed, so you should toggle the LED. The variable `last` should be updated to be used on the next loop. For the button code, the button state is read, and then turn on the RGB LED if the button is pressed, otherwise turn off the RGB LED.

Listing 12.2. Blink an LED without using `delay` function

```
// *** File   : /esp8266-arduino/led-without-delay/
//              led-without-delay.ino
// *** Author : Erwin Ouyang
// *** Date   : 17 Agt 2018
```

```

unsigned long now = 0;
unsigned long last = 0;

void setup()
{
    // Set pin as output
    pinMode(D3, OUTPUT);
    // Set pin as input pull-up
    pinMode(D6, INPUT_PULLUP);
    // Set PWM range from 0 to 255
    analogWriteRange(255);
}

void loop()
{
    // *** Blink the red LED ***
    // Read current millis value
    now = millis();
    // If 1 second has been elapsed
    if ((now - last) > 1000)
    {
        // Toggle the LED
        digitalWrite(D3, !digitalRead(D3));
        // Save current millis value as a
        // reference for next loop
        last = now;
    }

    // *** Read button state ***
    if (digitalRead(D6) == LOW)
        rgbLedWrite(255, 255, 255);
    else
        rgbLedWrite(0, 0, 0);
}

void rgbLedWrite(byte red, byte green, byte blue)
```

```
{
    analogWrite(D5, 255-red);
    analogWrite(D7, 255-green);
    analogWrite(D8, 255-blue);
}
```

12.3 Ticker

Ticker is a library for calling functions periodically. Ticker works like a timer interrupt occurrs every a certain period. The ticker example is shown in Listing 12.3. To use the ticker, you should include the `Ticker.h` library as shown in line 5. In line 7, the `ticker` object is created. In line 14, a function called `tickerCallback` is attached to the ticker by using `attach` method. The `tickerCallback` function will be executed every 1 second. In the `tickerCallback` function, there is code for blinking the LED.

Listing 12.3. Blink an LED using ticker

```
1  // *** File   : /esp8266-arduino/ticker/ticker.ino
2  // *** Author : Erwin Ouyang
3  // *** Date   : 17 Agt 2018
4  
5  #include <Ticker.h>
6  
7  Ticker ticker;
8  
9  void setup()
10 {
11     // Set pin as output
12     pinMode(D3, OUTPUT);
13     // Set ticker to generate interrupt every 1 second
14     ticker.attach(1, tickerCallback);
15 }
16 
```

```
void tickerCallback()
{
    // Toggle LED
    digitalWrite(D3, !digitalRead(D3));
}

void loop()
{
}
```

12.4 Summary

The `millis` and `micros` functions are used for getting the current milliseconds and microseconds value, respectively since the program started. You can call these functions periodically in order to blink the LED without the `delay` function.

12.5 Coding Challenge

In this coding challenge, you should create a dual function button program, i.e. a program that can detect a short button press and a long button press (press and hold) with only one button. You can use the timer for measuring the button press. Use the external interrupt on the GPIO input pin!

Chapter 13

DS1307 Real-Time Clock

In chapter 12, you have learned how to measure time using timer. In this chapter, you will learn how to use a Real-Time Clock (RTC). You will read the time and date from DS1307 RTC, and send them to PC using serial communication. You will also use the OLED display for displaying the time and date.

What will you learn in this chapter?

- Read the time and date from DS1307 RTC.
- Send the time and date to PC using serial communication.
- Display the time and date on OLED display.

13.1 Real-Time Clock

A Real-Time Clock (RTC) is used for keeping time. The RTC works like a watch, and can run on a battery, so it works even there is a power outage. The DS1307 is a popular and low cost RTC. The DS1307 can be connected to ESP8266 by using I2C bus. To create a program for DS1307, you need an external library. The external library used in this chapter is RTC by Makuna (`https://github.com/Makuna/Rtc`).

13.2 Example Program

In the first example program, you will read the time and date from DS1307, and send them to PC using serial communication. The code for the program is shown in Listing 13.1. In line 5 and 6, you should include the `Wire.h` and `RtcDS1307.h` libraries. In line 9, the `rtc` object is created, and you should initialize it by using `Begin` method as in line 16. In line 20–23, the DS1307's time and date is set to the compiled time of this code, and the DS1307 is enabled by using `SetIsRunning` method.

In main program, the time and date are read by using `GetDateTime` method. The `GetDateTime` method returns the time and date, and then they are stored in a structure called `RtcTimeDate` as shown in line 29. In line 32, you should send the time and date to PC using `printf` method. The time and date are sent to PC every 1 second.

Listing 13.1. Log the DS1307 time and date to PC

```
1  // *** File   : /esp8266-arduino/ds1307/ds1307.ino
2  // *** Author : Erwin Ouyang
3  // *** Date   : 17 Agt 2018
4
5  #include <Wire.h>
6  #include <RtcDS1307.h>
7
8  // RTC object declaration
9  RtcDS1307<TwoWire> rtc(Wire);
10
11 void setup()
12 {
13     // Setup serial communication
```

```
    Serial.begin(74880);
    // Initialize RTC
    rtc.Begin();

    // *** Set RTC date and time to code
    // compiled time ***
    RtcDateTime compiled = RtcDateTime(__DATE__,
            __TIME__);
    rtc.SetDateTime(compiled);
    rtc.SetIsRunning(true);
}

void loop()
{
    // Read RTC date and time
    RtcDateTime now = rtc.GetDateTime();
    // *** Send RTC date and time to serial
    // monitor ***
    Serial.printf("%04d/%02d/%02d %02d:%02d:%02d\n",
            now.Year(), now.Month(), now.Day(),
            now.Hour(), now.Minute(), now.Second());
    delay(1000);
}
```

In the second example, you should modify the code in the previous example. Instead of sending the time and date to PC, you can display them on OLED display. The code for the program is shown in Listing 13.2. The code is similar to the previous example, but you should add the OLED display. In line 48–55, the time and date are displayed on the OLED display.

Listing 13.2. Display the DS1307 time and date on OLED display

```
1 // *** File   : /esp8266-arduino/ds1307-oled-display/
2 //              ds1307-oled-display.ino
3 // *** Author : Erwin Ouyang
4 // *** Date   : 17 Agt 2018
```

```

#include <Wire.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#include <RtcDS1307.h>

// *** Check library setting ***
#if (SSD1306_LCDHEIGHT != 64)     // 128 x 64 pixel display
#error("Height incorrect, please fix Adafruit_SSD1306.h!");
#endif

// OLED I2C address
#define OLED_ADDR 0x3C

// OLED object declaration
Adafruit_SSD1306 oled;
// RTC object declaration
RtcDS1307<TwoWire> rtc(Wire);

void setup()
{
    // *** Initialize and clear display ***
    oled.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR);
    // Clear OLED buffer
    oled.clearDisplay();

    // Initialize RTC
    rtc.Begin();

    // *** Set RTC date and time to code
    // compiled time ***
    RtcDateTime compiled = RtcDateTime(__DATE__,
            __TIME__);
    rtc.SetDateTime(compiled);
    rtc.SetIsRunning(true);
}

```

```
void loop()
{
    // Read RTC date and time
    RtcDateTime now = rtc.GetDateTime();

    // *** Display RTC date and time on OLED ***
    oled.clearDisplay();
    oled.setTextSize(1);
    oled.setTextColor(WHITE);
    oled.setCursor(8, 30);
    oled.printf("%04d/%02d/%02d %02d:%02d:%02d",
            now.Year(), now.Month(), now.Day(),
            now.Hour(), now.Minute(), now.Second());
    oled.display();

    delay(1000);
}
```

13.3 Summary

To set the DS1307's time and date, you can use the `SetDateTime` method. To start the DS1307, you can use the `SetIsRunning` method. To get the current time and date, you can use the `GetDateTime` method.

Chapter 14

DHT11 Temperature and Humidity Sensor

In chapter 13, you have learned how to use the DS1307 RTC. In this chapter, you will learn how to measure temperature and humidity using DHT11 sensor. The temperature and humidity are sent to PC by using serial communication, and also displayed on the OLED display.

What will you learn in this chapter?

- Read the temperature and humidity from DHT11 sensor.
- Send the temperature and humidity to PC using serial communication.
- Display the temperature and humidity on OLED display.

14.1 Temperature and Humidity Sensor

The DHT11 sensor is a popular and low cost temperature and humidity sensor. This sensor uses a digital interface, which is the 1-wire bus, so you do not need an ADC. To create a program for DHT11, you need two external libraries. The first library is Adafruit sensor (`https://github.com/adafruit/Adafruit_Sensor`), and the second library is DHT sensor library (`https://github.com/adafruit/DHT-sensor-library`).

14.2 Example Program

In the first example program, you will read the temperature and humidity from DHT11, and send them to PC using serial communication. The code for the program is shown in Listing 14.1. In line 5, you should include the `DHT.h` library. In line 8, the `dht` object is created. It has two input parameters, which are the DHT GPIO pin and the DHT type, which is DHT11 (because this library can also be used for DHT22 sensor). In line 15, the DHT11 is initialized by using `begin` method.

In main program, you can use the `readTemperature` and `readHumidity` methods for reading the temperature and humidity. In line 22, the value `true` is passed as input parameter for `readTemperature` method. This paratemer is used to return the temperature in Fahrenheit. The temperature and humidity are sent to PC using `printf` function as shown in line 27. You should add 2 seconds delay, because the minimum reading period of the DHT11 is 2 seconds, i.e. the DHT11 is a very slow sensor.

Listing 14.1. Log the DHT11 temperature and humidity to PC

```
1  // *** File   : /esp8266-arduino/dht11/dht11.ino
2  // *** Author : Erwin Ouyang
3  // *** Date   : 17 Agt 2018
4
5  #include <DHT.h>
6
7  // DHT object declaration
8  DHT dht(D4, DHT11);
```

```

void setup()
{
    // Setup serial communication
    Serial.begin(74880);
    // Initialize DHT11
    dht.begin();
}

void loop()
{
    // *** Read temperature and humidity ***
    float celcius = dht.readTemperature();
    float fahrenheit = dht.readTemperature(true);
    float humidity = dht.readHumidity();

    // *** Send temperature and humidity to
    // serial monitor ***
    Serial.printf("Temperature: %.0fC, %.0fF " \
            "Humidity: %.0f%%\n", celcius,
            fahrenheit, humidity);

    delay(2000);
}
```

In the second example program, you should modify the code in the previous example. Instead of sending the temperature and humidity to PC, you can display them on OLED display. The code for the program is shown in Listing 14.2. The code is similar to the previous example, but you should use OLED display instead of serial communication. In main program, the temperature and humidity are read and sent to the OLED display every 3 seconds.

Listing 14.2. Display the DHT11 temperature and humidity on OLED display

```
// *** File   : /esp8266-arduino/dht11-oled-display/
//              dht11-oled-display.ino
// *** Author : Erwin Ouyang
// *** Date   : 17 Agt 2018

#include <Wire.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#include "DHT.h"

// *** Check library setting ***
#if (SSD1306_LCDHEIGHT != 64)    // 128 x 64 pixel display
#error("Height incorrect, please fix Adafruit_SSD1306.h!");
#endif

// OLED I2C address
#define OLED_ADDR 0x3C

// OLED object declaration
Adafruit_SSD1306 oled;
// DHT object declaration
DHT dht(D4, DHT11);

void setup()
{
    // *** Initialize and clear display ***
    oled.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR);
    oled.clearDisplay();

    // Initialize DHT11
    dht.begin();
}

void loop()
{
    // *** Read temperature and humidity ***
    float celcius = dht.readTemperature();
    float humidity = dht.readHumidity();

    // *** Display temperature and humidity on OLED ***
    oled.clearDisplay();
    oled.setTextSize(2);
    oled.setTextColor(WHITE);
    oled.setCursor(45, 12);
    oled.printf("%.0f%cC", celcius, (char)247);
    oled.setCursor(50, 38);
    oled.printf("%.0f%%", humidity);
    oled.display();

    delay(3000);
}
```

14.3 Summary

To get the temperature and humidity values, you can use the `readTemperature` and `readHumidity` methods. The DHT11 is a slow sensor, which the minimum reading period is 2 seconds.

14.4 Coding Challenge

In this coding challenge, you should display the temperature and humidity from the DHT11 along with the time and date from the DS1307 on the OLED display. You should also display the temperature and humidity symbols as shown in Figure 14.1. The DS1307 is read every 1 second, while the DHT11 is read every 3 seconds.

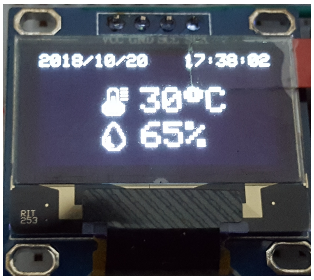


Figure 14.1. Time, date, temperature, and humidity are displayed on OLED display.

Bibliography

[1] Wikipedia, "Esp8266." https://en.wikipedia.org/wiki/ESP8266. Accessed on 2018-08-25.

[2] Arduino, "Pwm." https://www.arduino.cc/en/Tutorial/PWM. Accessed on 2018-09-11.

[3] ESP8266, "Esp8266 arduino core reference." http://esp8266.github.io/Arduino/versions/2.0.0/doc/reference.html. Accessed on 2018-09-11.

[4] Diarmuid, "Pwm exponential led fading on arduino (or other platforms)." https://diarmuid.ie/blog/pwm-exponential-led-fading-on-arduino-or-other-platforms. Accessed on 2018-10-7.

[5] A. Greenste, "Switch debouncing." http://www.labbookpages.co.uk/electronics/debounce.html. Accessed on 2018-10-14.

[6] Wikipedia, "Interrupt." https://en.wikipedia.org/wiki/Interrupt. Accessed on 2018-09-20.

[7] RapidTables, "Hsv to rgb color conversion." https://www.rapidtables.com/convert/color/hsv-to-rgb.html. Accessed on 2018-10-14.

[8] Wikipedia, “Serial communication.” `https://en.wikipedia.org/wiki/Serial_communication`. Accessed on 2018-10-14.

[9] cplusplus, “printf.” `www.cplusplus.com/reference/cstdio/printf/`. Accessed on 2018-10-16.

Solution to Coding Challenge

Chapter 3

Non-linear LED dimming

```
// *** File   : /esp8266-arduino/led-pwm-non-linear/
//              led-pwm-non-linear.ino
// *** Author : Erwin Ouyang
// *** Date   : 17 Agt 2018

void setup()
{
}

void loop()
{
    // *** Increase brightness ***
    for (int i = 0; i <= 1023; i += 20)
    {
        analogWrite(D3, pow(2, (i/102.3))-1);
        delay(25);
    }
    // *** Decrease brightness ***
    for (int i = 1023; i >= 0; i -= 20)
    {
        analogWrite(D3, pow(2, (i/102.3))-1);
        delay(25);
    }
}
```

Chapter 4

RGB LED color spectrum

```
// *** File   : /esp8266-arduino/rgb-led-spectrum/
//              rgb-led-spectrum.ino
// *** Author : Erwin Ouyang
// *** Date   : 17 Agt 2018

// RGB color, starts with red
byte rgb[3] = {255, 0, 0};

void setup()
{
    // Set PWM range from 0 to 255
    analogWriteRange(255);
}

void loop()
{
    for (int i = 0; i <= 2; i++)
    {
        // *** Select which colors to decrement and
        // increment ***
        int dec = i;
        int inc = (dec + 1) % 3;

        // *** Decrement and increment the colors ***
        for (int j = 0; j < 255; j++)
        {
            rgb[dec] -= 1;
            rgb[inc] += 1;
            rgbLedWrite(rgb[0], rgb[1], rgb[2]);
            delay(5);
        }
    }
}

void rgbLedWrite(byte red, byte green, byte blue)
{
    analogWrite(D5, 255-red);
    analogWrite(D7, 255-green);
    analogWrite(D8, 255-blue);
}
```

Chapter 8

RGB LED color spectrum using trimpot

```
// *** File   : /esp8266-arduino/trimpot-rgb-led/
//              rgb-led.ino
// *** Author : Erwin Ouyang
// *** Date   : 17 Agt 2018

#include<math.h>

int trimpotValue;

uint8_t r, g, b;

void setup()
{
    // Set PWM range from 0 to 255
    analogWriteRange(255);
}

void loop()
{
    // Read trimpot value
    trimpotValue = analogRead(A0);
    // Map trimpot value from 0-1023 to 0-359
    trimpotValue = map(trimpotValue, 0, 1023, 0, 359);
    // Convert trimpot value to RGB value
    hsvToRgb(trimpotValue, 1.0, 1.0, &r, &g, &b);
    // Send RGB value to RGB LED
    rgbLedWrite(r, g, b);

    delay(10);
}

void hsvToRgb(uint16_t H, float S, float V,
    uint8_t *R, uint8_t *G, uint8_t *B)
{
    float R_a, G_a, B_a;

    // Calculate C
    float C = V * S;
    // Calculte X
    float X = C * (1.0 - fabs(fmod(H/60.0, 2.0) - 1.0));
    // *** Calculate R', G', and B' ***
    if ((H >= 0) && (H < 60))
    {
        R_a = C; G_a = X; B_a = 0;
    }
    else if ((H >= 60) && (H < 120))
    {
        R_a = X; G_a = C; B_a = 0;
    }
    else if ((H >= 120) && (H < 180))
    {
        R_a = 0; G_a = C; B_a = X;
    }
    else if ((H >= 180) && (H < 240))
    {
        R_a = 0; G_a = X; B_a = C;
    }
    else if ((H >= 240) && (H < 300))
    {
        R_a = X; G_a = 0; B_a = C;
    }
    else if ((H >= 300) && (H < 360))
    {
        R_a = C; G_a = 0; B_a = X;
    }
    // Calculate m
    float m = V - C;
    // *** Calculate R, G, and B ***
    *R = (uint8_t)((R_a + m) * 255);
    *G = (uint8_t)((G_a + m) * 255);
    *B = (uint8_t)((B_a + m) * 255);
}

void rgbLedWrite(byte red, byte green, byte blue)
{
    analogWrite(D5, 255-red);
    analogWrite(D7, 255-green);
    analogWrite(D8, 255-blue);
}
```

Chapter 11

RGB LED and OLED commands

```
// *** File   : /esp8266-arduino/serial-strtok/
//              serial-strtok.ino
// *** Author : Erwin Ouyang
// *** Date   : 17 Agt 2018

#include <Wire.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
```

```

// *** Check library setting ***
#if (SSD1306_LCDHEIGHT != 64)    // 128 x 64 pixel display
#error("Height incorrect, please fix Adafruit_SSD1306.h!");
#endif

// OLED I2C address
#define OLED_ADDR 0x3C

// OLED object declaration
Adafruit_SSD1306 oled;

String cmd_s;
char cmd_c[64];
char *chr_p;

void setup()
{
    // Set pin as output
    pinMode(D3, OUTPUT);
    // Set PWM range from 0 to 255
    analogWriteRange(255);
    // Setup serial communication
    Serial.begin(74880);

    // *** Initialize OLED display ***
    oled.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR);
    oled.clearDisplay();
    oled.setTextSize(1);
    oled.setTextColor(WHITE);
    oled.display();
}

void loop()
{
    if (Serial.available() > 0)
    {
        // Read command from receive buffer
        cmd_s = Serial.readStringUntil('\n');

        // *** Split the command ***
        cmd_s.toCharArray(cmd_c, 64);
        chr_p = strtok(cmd_c, "+,");
        String tokens[4];
        uint8_t idx = 0;
        while (chr_p != NULL)
        {
            tokens[idx++] = chr_p;
            chr_p = strtok(NULL, "+,");
        }

        // *** Process the command ***
        if (tokens[0] == "LED")
        {
            if (tokens[1] == "ON")
            {
                digitalWrite(D3, HIGH);
            }
            else if (tokens[1] == "OFF")
            {
                digitalWrite(D3, LOW);
            }
            else
            {
                if (digitalRead(D3))
                    Serial.println("LED is on");
                else
                    Serial.println("LED is off");
            }
        }
        else if (tokens[0] == "RGB")
        {
            rgbLedWrite(tokens[1].toInt(),
                    tokens[2].toInt(), tokens[3].toInt());
        }
        else if (tokens[0] == "OLED")
        {
            oled.clearDisplay();
            oled.setCursor(0, 0);
            oled.printf("%s", tokens[2].c_str());
            oled.display();
        }
        else
        {
            Serial.println("Unknown command");
        }
    }
}

void rgbLedWrite(byte red, byte green, byte blue)
{
    analogWrite(D5, 255-red);
    analogWrite(D7, 255-green);
    analogWrite(D8, 255-blue);
}
```

Chapter 12

Dual function button

```
// *** File   : /esp8266-arduino/button-dual-function/
//              button-dual-function.ino
// *** Author : Erwin Ouyang
// *** Date   : 17 Agt 2018

unsigned long pressed = 0;
unsigned long released = 0;
uint8_t r = 0, g = 0, b = 0;

void setup()
```

```
{
    // Set pin as output
    pinMode(D3, OUTPUT);
    // Set pin as input pull-up
    pinMode(D6, INPUT_PULLUP);
    // Set external interrupt from pin D6 that is
    // connected to button
    attachInterrupt(digitalPinToInterrupt(D6),
        isr, CHANGE);
    // Set PWM range from 0 to 255
    analogWriteRange(255);
}

void isr()
{
    // *** Read button state ***
    if (digitalRead(D6) == LOW)
    {
        pressed = millis();
    }
    else
    {
        released = millis();
        if ((released - pressed) < 1000)
        {
            // Toggle the LED
            digitalWrite(D3, !digitalRead(D3));
        }
        else
        {
            // *** Toggle the RGB LED ***
            r ^= 0xFF;
            g ^= 0xFF;
            b ^= 0xFF;
            rgbLedWrite(r, g, b);
        }
    }
}

void loop()
{
}

void rgbLedWrite(byte red, byte green, byte blue)
{
    analogWrite(D5, 255-red);
    analogWrite(D7, 255-green);
    analogWrite(D8, 255-blue);
}
```

Chapter 14

DHT11 and DS1307 on OLED display

```
// *** File   : /esp8266-arduino/dht11-ds1307/
//              dht11-ds1307.ino
// *** Author : Erwin Ouyang
// *** Date   : 17 Agt 2018

#include <Wire.h>
#include <Adafruit_SSD1306.h>
#include <Adafruit_GFX.h>
#include <RtcDS1307.h>
#include <DHT.h>

// *** Check library setting ***
#if (SSD1306_LCDHEIGHT != 64)    // 128 x 64 pixel display
#error("Height incorrect, please fix Adafruit_SSD1306.h!");
#endif

// OLED I2C address
#define OLED_ADDR 0x3C

```

```
// Temperature logo
static const unsigned char PROGMEM temperature_bmp[] =
{
    B00000011, B11001111,
    B00000110, B01100000,
    B00000100, B00101111,
    B00000100, B00100000,
    B00000100, B00101111,
    B00000101, B10100000,
    B00000101, B10101111,
    B00000101, B10100000,
    B00001101, B10110000,
    B00011011, B11011000,
    B00110111, B11101100,
    B00101111, B11110100,
    B00101111, B11110100,
    B00110111, B11101100,
    B00011000, B00011000,
    B00001111, B11110000
};
// Humidity logo
static const unsigned char PROGMEM humidity_bmp[] =
{
  B00000001, B10000000,
    B00000011, B11000000,
    B00000110, B01100000,
    B00000100, B00100000,
    B00001100, B00110000,
    B00001000, B00010000,
    B00011000, B00011000,
    B00010000, B00001000,
    B00110100, B00001100,
    B00101100, B00000100,
    B00101100, B00000100,
    B00101100, B00000100,
    B00110110, B00001100,
    B00011011, B00011000,
    B00001100, B00110000,
    B00000111, B11100000
};

// OLED object declaration
Adafruit_SSD1306 oled;
// RTC object declaration
RtcDS1307<TwoWire> rtc(Wire);
// DHT object declaration
DHT dht(D4, DHT11);

RtcDateTime now;
float celcius, humidity;
uint8_t second = 0;

void setup()
{
    // *** Initialize and clear display ***
    oled.begin(SSD1306_SWITCHCAPVCC, OLED_ADDR);
    // Clear OLED buffer
    oled.clearDisplay();

    // Initialize RTC
    rtc.Begin();

    // *** Set RTC date and time to code
    // compiled time ***
    RtcDateTime compiled = RtcDateTime(__DATE__,
            __TIME__);
    rtc.SetDateTime(compiled);
    rtc.SetIsRunning(true);

    // Initialize DHT11
    dht.begin();
}

void loop()
```

```
{
    // Read RTC date and time
    now = rtc.GetDateTime();

    // *** Read temperature and humidity every 3
    // seconds ***
    if (second == 3)
    {
        celcius = dht.readTemperature();
        humidity = dht.readHumidity();
        second = 0;
    }

    // *** Display date, time, temperature,
    // and humidity on OLED ***
    oled.clearDisplay();
    oled.setTextSize(1);
    oled.setTextColor(WHITE);
    oled.setCursor(0, 0);
    oled.printf("%04d/%02d/%02d",
            now.Year(), now.Month(), now.Day());
    oled.setCursor(80, 0);
    oled.printf("%02d:%02d:%02d",
            now.Hour(), now.Minute(), now.Second());
    oled.drawBitmap(32, 20, temperature_bmp, 16, 16, 1);
    oled.drawBitmap(32, 40, humidity_bmp, 16, 16, 1);
    oled.setTextSize(2);
    oled.setCursor(55, 20);
    oled.printf("%.0f%cC", celcius, (char)247);
    oled.setCursor(55, 40);
    oled.printf("%.0f%%", humidity);
    oled.display();

    delay(1000);
    second++;
}
```