ARTICLE

Project X Session 1 - Introduction to Arduino IDE and ESP8266

Yuxuan Han, Junzhe Chen, and Oli Sharratt

Introduction

In the first Project X session, you will recall basic electronic theorem, and learn how to use *Arduino IDE* to program the *ESP8266* microcontroller board. Try to follow the instructions in this document and first your first two simple programs: *Blink* and *Hello World*.

1. Configure Arduino IDE for ESP8266

1.1 Library, board and port

To make sure the Arduino IDE can program the ESP8266 without any problem, the following procedures should be done:

- Download the ESP8266 library
- Select the correct ESP8266 board to program
- Select the correct port to communicate between the ESP and the Arduino IDE

Follow this link to learn how to configure the Arduino IDE for the ESP8266: https://randomnerdtutorials.com/how-to-install-esp8266-board-arduino-ide/

1.2 Pin-out of the ESP8266

Figure 1 below is the *pin-out* of the ESP8266 board. It defines the functions of all the pins on the board. It is not that important for today's session, but you may need this pinout diagram in the future, for example, finding the I2C pins.

2. The *Blinky* Program

For the blink program, you will program your ESP8266 board to blink an external LED.

2.1 Build the External Circuit

Please build the circuit for the *Blinky* program according to Figure 2. Pay attention to the legs of the LED. The longer leg is the anode (+), where the current flows in, and the shorter leg is the cathode (-), where the current flows out.

To connect the LED to the microcontroller shown in Figure 2, we need to use a breadboard. This is an external board where you can connect through-hole components (those components with pins) together quickly without needing to solder. It is very often used when prototyping such as making a *blinky* program and testing a sensor.

According to the circuit connection in Figure 3. If we want to illuminate the LED, the pin *D5* should be turned to logic high (i.e. supplying 3.3 V). Similarly, if we want to turn off the LED, the pin *D5* should be turned to logic low (i.e. supplying 0 V). Thus, if we alter the D5 pin output between logic low and logic high, the LED will start to blink.

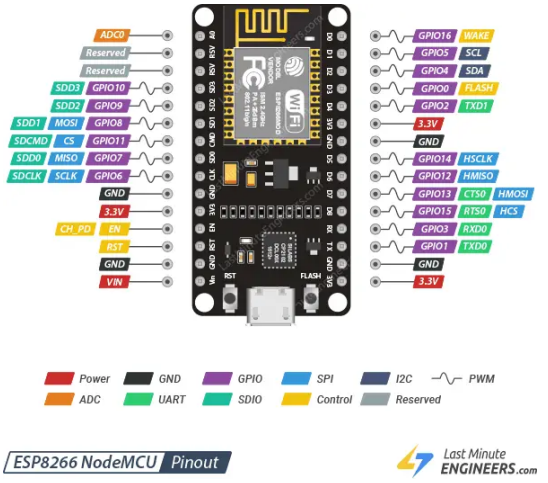

Figure 1. Pin-out of the ESP8266

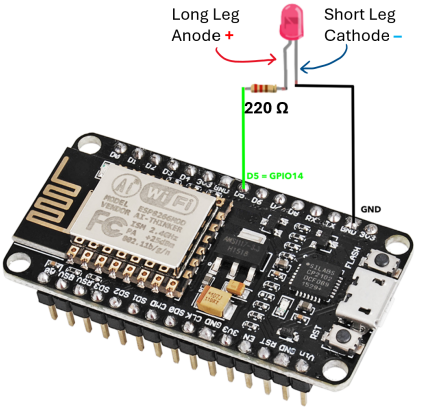

Figure 2. Circuit connection for *Blinky*

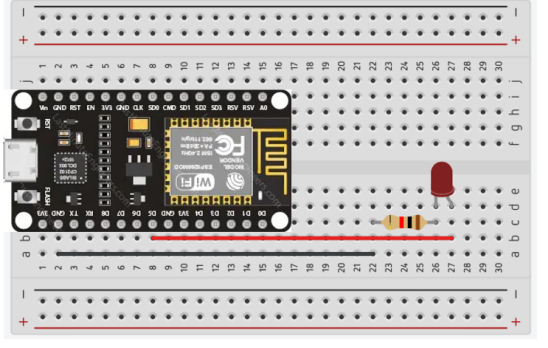

Figure 3. LED connection for blink program

Table 1. Pins used for *Blinky*

Pin Used	Connected to	Purpose
D5	LED Anode	Supply power to and control the LED
GND	LED Cathode	Ground

2.2 Program the C Code

The C code of the *Blinky* program has been provided in Figure 4. You can try to create a new file in Arduino IDE, and type it in. Let's go through all the lines of this *blinky* program together and try to understand their meaning.

```
1   #define LED D5
2
3   void setup()
4   {
5     pinMode(LED, OUTPUT);
6   }
7
8   void loop()
9   {
10    digitalWrite(LED, HIGH);
11    delay(1000);
12    digitalWrite(LED, LOW);
13    delay(1000);
14  }
```

Figure 4. C code for *blinky*

When you created a new file in Arduino IDE, you may already noticed that there are already some pieces of code written inside: `void setup()` and `void loop()`. They are the most important two *functions* in Arduino and each of them follows a pair of curly braces '`{ }`'. Each pair of curly braces represents a section of code, which is the content of the function. For the *setup* function, the code inside will only be executed once when the microcontroller powers up or after being reset manually. For the *loop* function, the code inside will be executed in a cyclic behaviour, line by line, after the *setup* function finished - that's also why it is called *loop*.

Line 1 of the code is a *macro* definition. The effect of this line of code is to tell the computer, if you see the word "LED" again below this line, replace them with *D5*, which is the pin number that we connect the LED to. Using macro definition is a good practise of coding. For example, if you want to change the LED pin number at a later point, you only need to edit the macro definition line once and no need to search and change all of the pin numbers inside a large amount of code.

Line 3 – 6 is the setup function. One important thing we need to do is to configure the D5 pin as an output device. This is done in line 5.

Line 10 – 14 is the loop function. In the loop, we want the D5 pin to continuously toggle high and low, to blink our LED on and off. In line 10, we configure the LED *D5* pin to logic high (3.3 V), and in line 11, we let the microcontroller to wait for 1000 milliseconds (1 second) by using `delay(1000)`. Then, in line 12 we configure the LED pin to logic low (0 V), and in line 13 we wait for another 1000 ms. The loop function then comes to the end. However, since it is a "loop", the microcontroller will jump back to line 10 and re-execute the code in this section.

2.3 Compile and Flash the Code

After you finished typing in the code in Figure 4 and correctly configured the Arduino IDE by following the instructions in Section 1.1, you can upload your code to the ESP8266. To compile and flash the code onto the microcontroller and see the code running, you can press the *upload* icon on the top-left corner of the Arduino IDE, or simply use `<Ctrl+U>`. The code will automatically be compiled and uploaded onto your board.

If you are unable to upload the code, or see any error message, or your LED light does not blink, please call the volunteer near you and we can offer you some help.

2.4 Serial Communication

The microcontroller and the computer can communication with each other. One of the popular communication protocols between the microcontroller and the Arduino IDE is `Serial` communication.

The first step is to set up the microcontroller to begin the serial communication. The command `Serial.begin(baud_rate)` in the `setup` loop represents the configuration of the serial communication. The baud rate indicates how fast the microcontroller will make communication with the computer per second, usually `115200` is a common baud rate to use, as seen in line 3 of the example code.

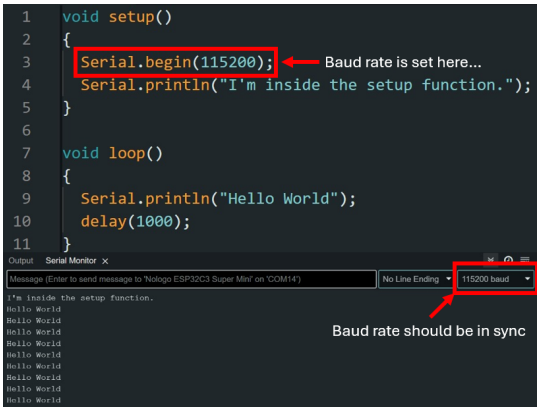

Figure 5. Codes and serial outputs of example serial communication

When the serial communication is enabled, we can use the `Serial.print` or `Serial.println` to print out text or variable values through serial and view them in the serial monitor. To open serial monitor, go to the top bar and click `Tools/Open Serial Monitor`. Alternatively, click the magnifying glass icon on the right hand side of the Arduino IDE or press `<Ctrl+Shift+M>`.

It is important to set the baud rate of the serial monitor to be the same as the initiated baud rate. Sometimes, it is a good practice to perform a sanity check in the `setup` function. Line 4 will be printed out when the serial connection initiated. If the baud rate is set to be faulty, the information will not be corrected displayed, which can be a hint during the debugging process.