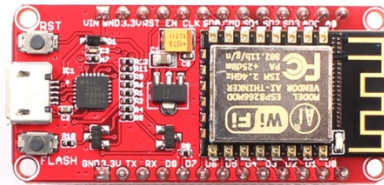


2.0 ASSEMBLY

2.1 ESP8266 MODULE

There are many different modules available based on ESP8266s, they differ in a number of aspects such as the quantity of available GPIO pins or if they can be programmed easily without need of an additional Serial-to-USB converter. If you are a beginner I suggest you use a developer-friendly module like the NodeMCU V1.0 or the Wemos D1 mini. They come with a USB connector and have the maximum number of available pins ready for your usage. The absolute minimal requirement is that your ESP8266 module has at least two free GPIO pins to connect it to the OLED display.



2.1.1 HELLO WORLD: THE SERIAL CONSOLE

Every self-respecting programming tutorial starts with a “Hello World” program. And I don’t want to break with this tradition here. A Hello-World program usually does no more than printing these two words somewhere on the screen. But we are programming a microcontroller which does not have a screen yet. So where can we display the text? We will use the Serial object to do that. While you are developing a program on the ESP8266, the microcontroller is connected to the computer that the Arduino IDE is running on. We use this connection to write a new binary onto the flash memory of the ESP8266. And while our program is running we can also use it to write messages from the ESP8266 back to our computer.

Using the Serial object is fairly easy. You have to initialize it first:

```
Serial.begin(115200);
```

This tells the Serial object that you want to communicate with a baud rate of 115200. Remember to set the same transfer rate later in the serial console on your computer. Both partners in the communication need to have the same speed settings or you will just see garbage. If you want to send a message from your program to your computer, you just do this:

```
Serial.print("Hello ");
```

```
Serial.println("World");
```

Please have a look at the little difference between the first and the second line. The first uses a

method called `print()` and the second `println()`. The only difference is that the latter adds a line break to the output.

You will need to use another important built-in function:

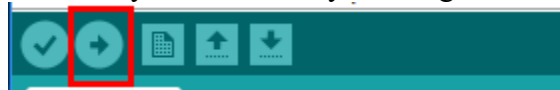
```
delay(1000);
```

This instructs the processor to wait 1000 milliseconds (1 second) before continuing with the execution. This command it also gives the processor time to handle other tasks, such as receiving or sending network packets over WiFi.

Now it is time to write our first program.

Download from Moodle and open the project **Exercise_2_1_E** in your Arduino IDE and fill in the required code to print “Hello World”.

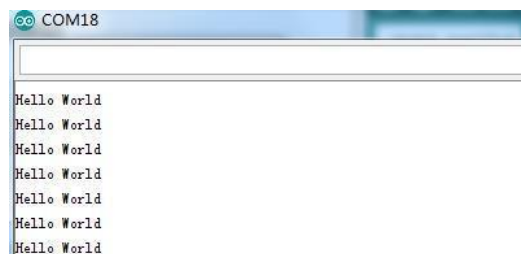
Remember that you only need to initialize the Serial object once. When you are happy with your solution upload the sketch to your Arduino by clicking



If that was successful open the serial console by clicking on the magnifying glass:



Now your output should look something like this:



CHALLENGE:

Make your code print the following pr “<number>. Hello world” as long as the code runs for example: “1. Hello World”, “2. Hello World”

2.1.2 ADDINONAL

If you want to learn more about the Serial object you can find more in the following link:

<http://esp8266.github.io/Arduino/versions/2.3.0/doc/reference.html#serial>

2.2: INPUT/OUTPUT: GPIO PINS

Now that we can talk to our microcontroller over the serial line it is time to interact with the real world. Our ESP8266 is equipped with several so-called “General Purpose Input Output” or in short “GPIO “pins. They can be used for many different applications such as sensing and generating digital signals of the 3.3 Volt range.

Using a GPIO pin is quite easy: first you tell the microcontroller if you want to read or write from the pin. Then you do it.

Here is the code for reading: `pinMode(PIN, INPUT);`
`int state = digitalRead(PIN)`

Unless you want to change the mode of a pin you only need to call `pinMode()` once. Please note that depending on the pin you can also use `INPUT_PULLUP` or `INPUT_PULLDOWN`. Writing to a pin is not much different:

```
pinMode(PIN, OUTPUT);
digitalWrite(PIN, HIGH); // or
digitalWrite(PIN, LOW);
```

The second statement will show a HIGH level on PIN which will be 3.3V. The third statement will set the pin to LOW which is 0V.

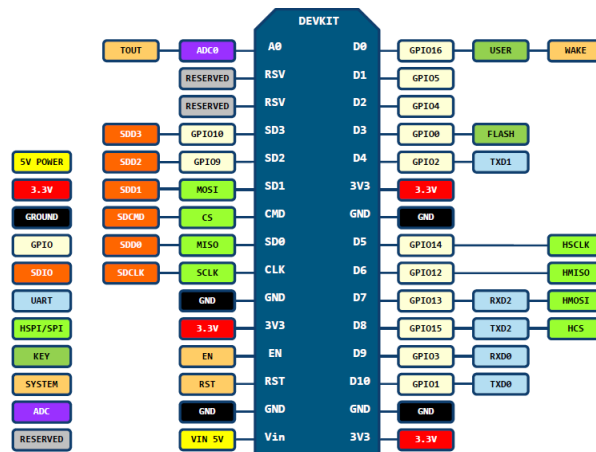
What values for PIN can you use? If you are using a generic ESP8266 module (not a NodeMCU) your pins will be labeled GPIO0, GPIO1, etc.

To use pin GPIO0 you would write `digitalWrite(0, HIGH);` If you are using a NodeMCU things get a little bit more complicated.

The original creators of the NodeMCU LUA firmware and the development module of the same name had the idea to give the pins different names. They are called D0, D1, etc. That by itself would not be confusing yet but they are not using the same digits, e.g. GPIO1 is not equal to D1.

Here is a table to map the pins:

Raw Module Name	NodeMCU	Raw Module Name	NodeMCU
GPIO0	D3	GPIO9	D11
GPIO1	D10	GPIO10	D12
GPIO2	D4	GPIO11	N/A
GPIO3	D9	GPIO12	D6
GPIO4	D2	GPIO13	D7
GPIO5	D1	GPIO14	D5
GPIO6	N/A	GPIO15	D8
GPIO7	N/A	GPIO16	D0
GPIO8	N/A		



NodeMCU DevKit Pin Map <https://github.com/nodemcu/nodemcu-devkit-v1.0>

Note: D0(GPIO16) can only be used as GPIO read/write, no interrupt supported, no pwm/i2c/ow supported.

2.2.1 LED IT SHINE!

From Moodle down load the file **Exercise 2 3 E**

Let's control an LED! Your NodeMCU has one built in and it is connected to D0. Like in the previous example set the pin to the correct mode and then toggle it once per second.

Please Note that D0 is the pin number noted by all the official documentations, but when tested it turns out to be |D2 or just 2.

CHALLENGE:

Pure blinking is boring. We want our little ESP8266 to live! Control the intensity of the LED and make it look as if the ESP8266 was breathing.

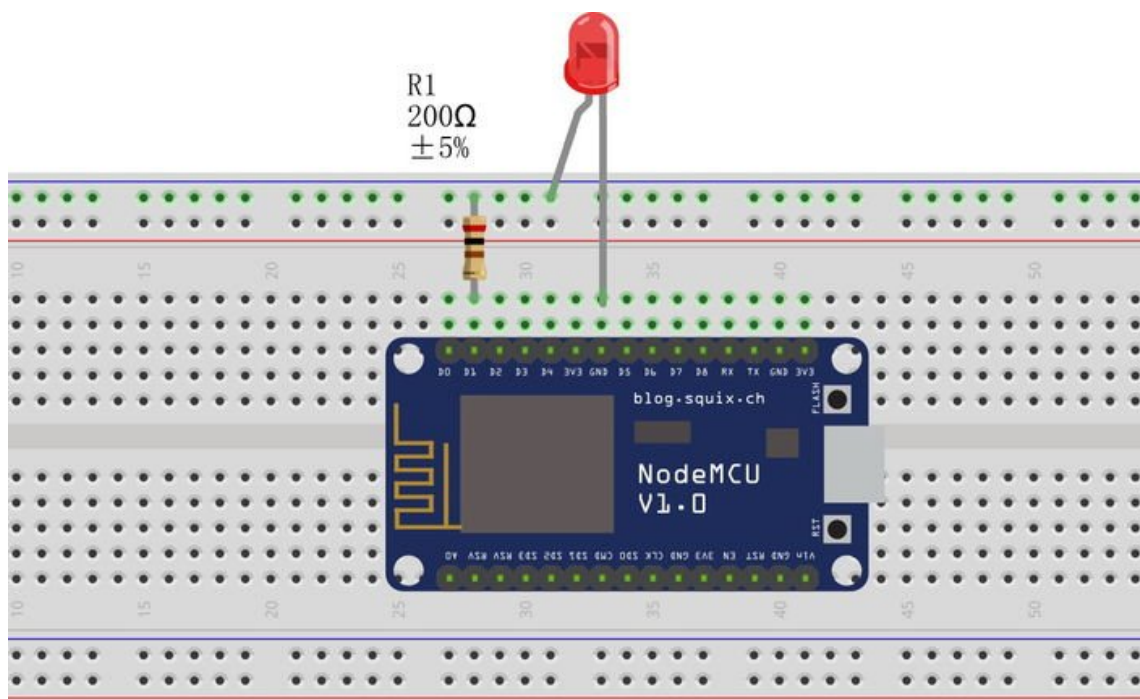
Note: if the LED was on another pin than (D0) we would use pulse width modulation (PWM) and the `writeAnalog(PIN, INTENSITY)` method. But this feature is not available on (D0) so you will have to do this in your code.

2.2.2 CONNECTING LED

The "Hello World" of any new HW project without any doubt, is a LED blinking. To connect a LED to your ESP-12E, you can use any of its digital GPIO. The above pin diagram shows the Pin layout of 2nd generation ESP8266 NodeMCU development board.

In our case, we will use pin D7 or its equivalent. You can test the code using "D7", or "13". both ways will work. The D7 Pin does not need an external resistor to power the LED (it has one internally). To test the LED use your code from last week, if you don't have it then write one quickly ☺.

The connection is as follows:

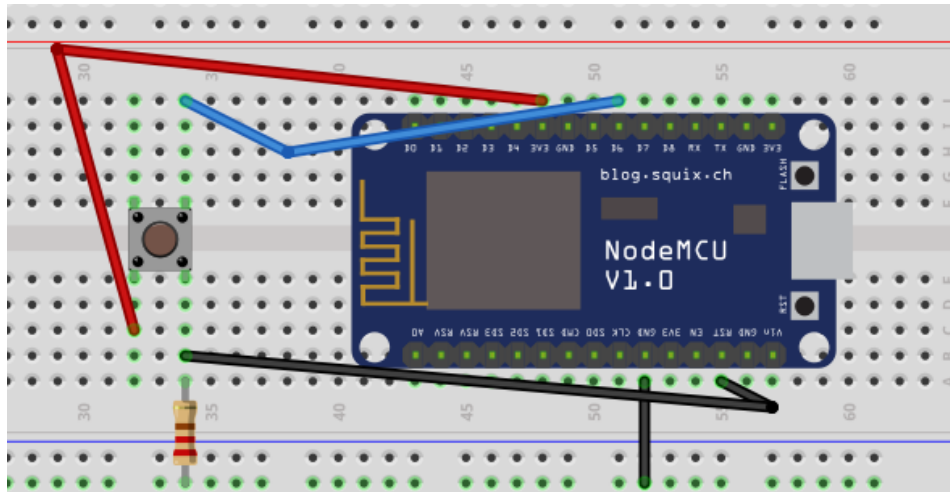


2.2.3 DON'T MISS A THING

Depending on your age you might remember interrupts from your PC. They were always important to get your sound card to play beautiful music.

The ESP8266 can also be controlled by interrupts. In the previous exercises we were checking regularly for the state of a GPIO pin. This is fine if you are not doing anything else in the main loop.

The connection is as follows :



You might miss a change in a state if it is very short, and that is where the interrupts can help. With an interrupt handler you can tell the processor that you are interested in a specific type of change and a given pin. This is how it works:

```
void buttonPressed() { ...
}
```

```
void setup()
{ pinMode(PIN, INPUT);
  attachInterrupt(digitalPinToInterrupt(PIN), buttonPressed, CHANGE); }
```

- `buttonPressed()` is a method without parameters that gets called when there is a change on PIN. Instead of `CHANGE` you can also use `RISING` which triggers the call-back when the pin changes from LOW to HIGH, or `FALLING` for a change in the opposite direction.

Please do not execute long tasks in the call-back method. The ESP's watchdog will reset the processor if calling the interrupt takes too much time. You should not do much more than changing a flag.

In this exercise we will setup an interrupt and turn an LED on and off with every press to the button. See what happens when you use different interrupt modes like `RISING`, `FALLING` and `CHANGE`.

CHALLENGE:

Connect the bottom with the LED and interrupt the status of the LED, build the hardware and write the code.