# Boards

## Adafruit HUZZAH ESP8266 (ESP-12)

Розглянемо лише плату NodeMCU, з якою ми будемо працювати, інші конфігурації ви зможете знайти за [посиланням](http://arduino-esp8266.readthedocs.io/en/2.4.0/boards.html).

## NodeMCU 0.9

### Pin mapping

Pin numbers written on the board itself do not correspond to ESP8266 GPIO pin numbers. Constants are defined to make using this board easier:

static const uint8\_t SDA = 4;

static const uint8\_t SCL = 5;

static const uint8\_t LED\_BUILTIN = 16;

static const uint8\_t BUILTIN\_LED = 16;

static const uint8\_t D0 = 16;

static const uint8\_t D1 = 5;

static const uint8\_t D2 = 4;

static const uint8\_t D3 = 0;

static const uint8\_t D4 = 2;

static const uint8\_t D5 = 14;

static const uint8\_t D6 = 12;

static const uint8\_t D7 = 13;

static const uint8\_t D8 = 15;

static const uint8\_t D9 = 3;

static const uint8\_t D10 = 1;

If you want to use NodeMCU pin 5, use D5 for pin number, and it will be translated to ‘real’ GPIO pin 14.

## NodeMCU 1.0

This module is sold under many names for around $6.50 on AliExpress and it’s one of the cheapest, fully integrated ESP8266 solutions.

It’s an open hardware design with an ESP-12E core and 4 MB of SPI flash.

According to the manufacturer, “with a micro USB cable, you can connect NodeMCU devkit to your laptop and flash it without any trouble”. This is more or less true: the board comes with a CP2102 onboard USB to serial adapter which just works, well, the majority of the time. Sometimes flashing fails and you have to reset the board by holding down FLASH + RST, then releasing FLASH, then releasing RST. This forces the CP2102 device to power cycle and to be re-numbered by Linux.

The board also features a NCP1117 voltage regulator, a blue LED on GPIO16 and a 220k/100k Ohm voltage divider on the ADC input pin.

Full pinout and PDF schematics can be found [here](https://github.com/nodemcu/nodemcu-devkit-v1.0)

## Generic ESP8266 modules

These modules come in different form factors and pinouts. See the page at ESP8266 community wiki for more info: [ESP8266 Module Family](http://www.esp8266.com/wiki/doku.php?id=esp8266-module-family).

Usually these modules have no bootstapping resistors on board, insufficient decoupling capacitors, no voltage regulator, no reset circuit, and no USB-serial adapter. This makes using them somewhat tricky, compared to development boards which add these features.

In order to use these modules, make sure to observe the following:

* **Provide sufficient power to the module.** For stable use of the ESP8266 a power supply with 3.3V and >= 250mA is required. Using the power available from USB to Serial adapter is not recommended, these adapters typically do not supply enough current to run ESP8266 reliably in every situation. An external supply or regulator alongwith filtering capacitors is preferred.
* **Connect bootstapping resistors** to GPIO0, GPIO2, GPIO15 according to the schematics below.
* **Put ESP8266 into bootloader mode** before uploading code.

## Serial Adapter

There are many different USB to Serial adapters / boards. To be able to put ESP8266 into bootloader mode using serial handshaking lines, you need the adapter which breaks out RTS and DTR outputs. CTS and DSR are not useful for upload (they are inputs). Make sure the adapter can work with 3.3V IO voltage: it should have a jumper or a switch to select between 5V and 3.3V, or be marked as 3.3V only.

Adapters based around the following ICs should work:

* FT232RL
* CP2102
* CH340G

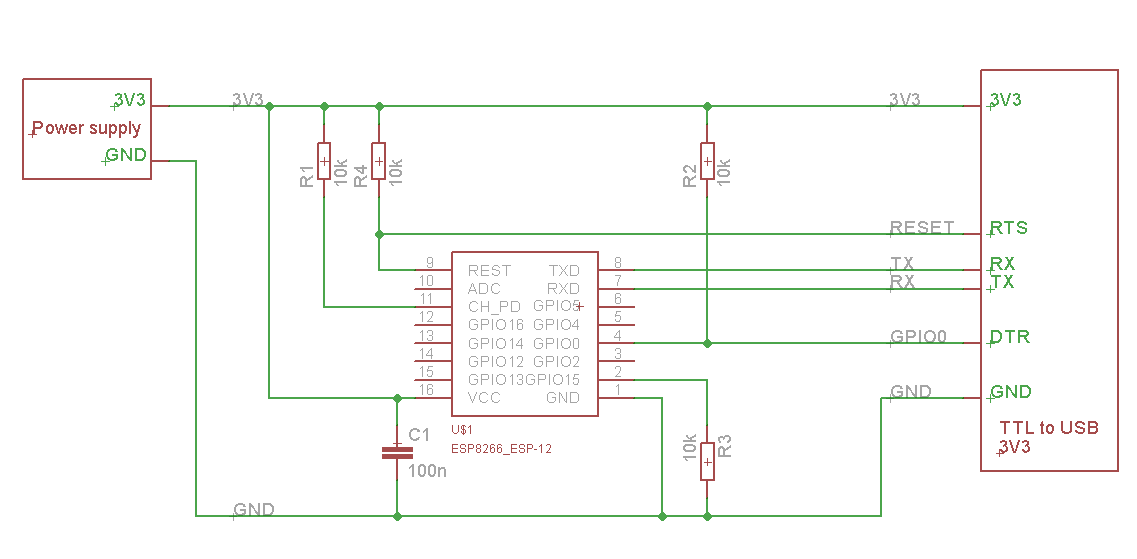
PL2303-based adapters are known not to work on Mac OS X. See <https://github.com/igrr/esptool-ck/issues/9> for more info.

## Minimal Hardware Setup for Bootloading and Usage

| **PIN** | **Resistor** | **Serial Adapter** |
| --- | --- | --- |
| VCC |  | VCC (3.3V) |
| GND |  | GND |
| TX or GPIO2\* |  | RX |
| RX |  | TX |
| GPIO0 | PullUp | DTR |
| Reset\* | PullUp | RTS |
| GPIO15\* | PullDown |  |
| CH\_PD | PullUp |  |

* Note
* GPIO15 is also named MTDO
* Reset is also named RSBT or REST (adding PullUp improves the stability of the module)
* GPIO2 is alternative TX for the boot loader mode
* **Directly connecting a pin to VCC or GND is not a substitute for a PullUp or PullDown resistor, doing this can break upload management and the serial console, instability has also been noted in some cases.**

## ESP to Serial



*ESP to Serial*

### Minimal Hardware Setup for Bootloading only

ESPxx Hardware

| **PIN** | **Resistor** | **Serial Adapter** |
| --- | --- | --- |
| VCC |  | VCC (3.3V) |
| GND |  | GND |
| TX or GPIO2 |  | RX |
| RX |  | TX |
| GPIO0 |  | GND |
| Reset |  | RTS\* |
| GPIO15 | PullDown |  |
| CH\_PD | PullUp |  |

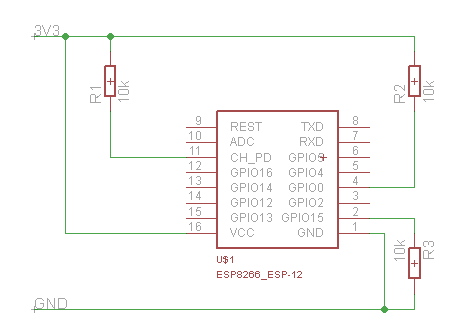
* Note
* if no RTS is used a manual power toggle is needed

### Minimal Hardware Setup for Running only

ESPxx Hardware

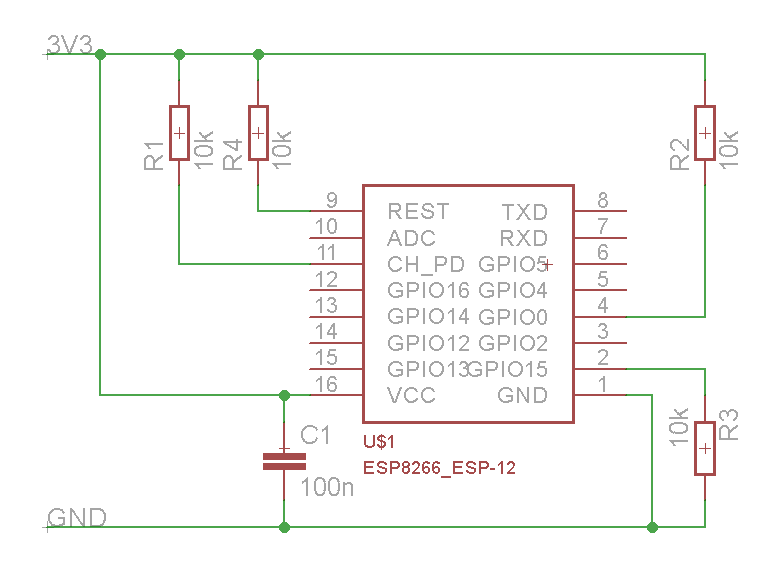
| **PIN** | **Resistor** | **Power supply** |
| --- | --- | --- |
| VCC |  | VCC (3.3V) |
| GND |  | GND |
| GPIO0 | PullUp |  |
| GPIO15 | PullDown |  |
| CH\_PD | PullUp |  |

## Minimal



*ESP min*

## Improved Stability



*ESP improved stability*

## Boot Messages and Modes

The ESP module checks at every boot the Pins 0, 2 and 15. based on them its boots in different modes:

| **GPIO15** | **GPIO0** | **GPIO2** | **Mode** |
| --- | --- | --- | --- |
| 0V | 0V | 3.3V | Uart Bootloader |
| 0V | 3.3V | 3.3V | Boot sketch (SPI flash) |
| 3.3V | x | x | SDIO mode (not used for Arduino) |

at startup the ESP prints out the current boot mode example:

rst cause:2, boot mode:(3,6)

note: - GPIO2 is used as TX output and the internal Pullup is enabled on boot.

### rst cause

| **Number** | **Description** |
| --- | --- |
| 0 | unknown |
| 1 | normal boot |
| 2 | reset pin |
| 3 | software reset |
| 4 | watchdog reset |

### boot mode

the first value respects the pin setup of the Pins 0, 2 and 15.

| **Number** | **GPIO15** | **GPIO0** | **GPIO2** | **Mode** |
| --- | --- | --- | --- | --- |
| 0 | 0V | 0V | 0V | Not valid |
| 1 | 0V | 0V | 3.3V | Uart |
| 2 | 0V | 3.3V | 0V | Not valid |
| 3 | 0V | 3.3V | 3.3V | Flash |
| 4 | 3.3V | 0V | 0V | SDIO |
| 5 | 3.3V | 0V | 3.3V | SDIO |
| 6 | 3.3V | 3.3V | 0V | SDIO |
| 7 | 3.3V | 3.3V | 3.3V | SDIO |

note: - number = ((GPIO15 << 2) | (GPIO0 << 1) | GPIO2);

## Generic ESP8285 modules

ESP8285 ([datasheet](http://www.espressif.com/sites/default/files/0a-esp8285_datasheet_en_v1.0_20160422.pdf)) is a multi-chip package which contains ESP8266 and 1MB flash. All points related to bootstrapping resistors and recommended circuits listed above apply to ESP8285 as well.

Note that since ESP8285 has SPI flash memory internally connected in DOUT mode, pins 9 and 10 may be used as GPIO / I2C / PWM pins.

# Debugging

## Introduction

Since 2.1.0-rc1 the core includes a Debugging feature that is controllable over the IDE menu.

The new menu points manage the real-time Debug messages.

### Requirements

For usage of the debugging a Serial connection is required (Serial or Serial1).

The Serial Interface need to be initialized in the setup().

Set the Serial baud rate as high as possible for your Hardware setup.

Minimum sketch to use debugging:

void setup() {

Serial**.**begin(115200);

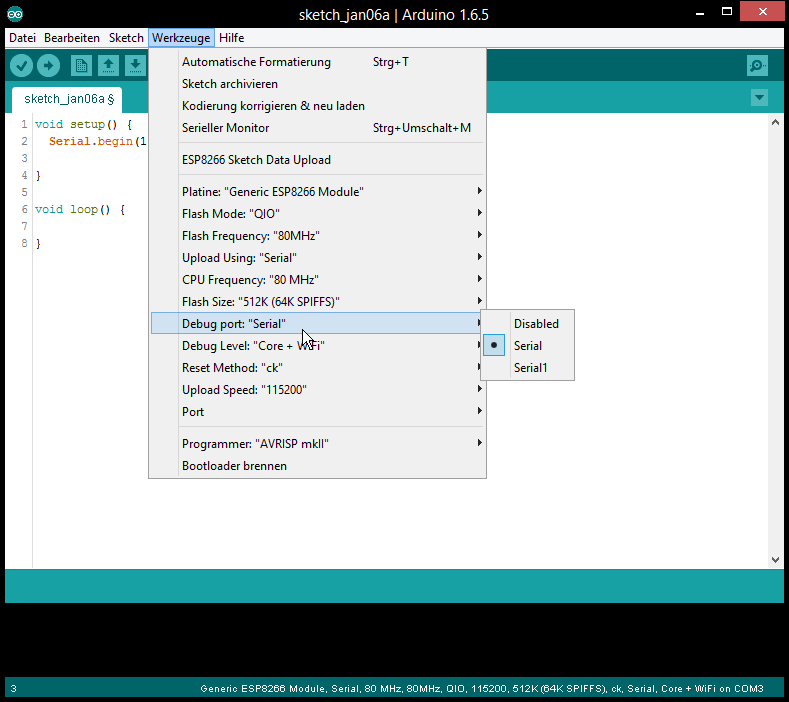
}

void loop() {

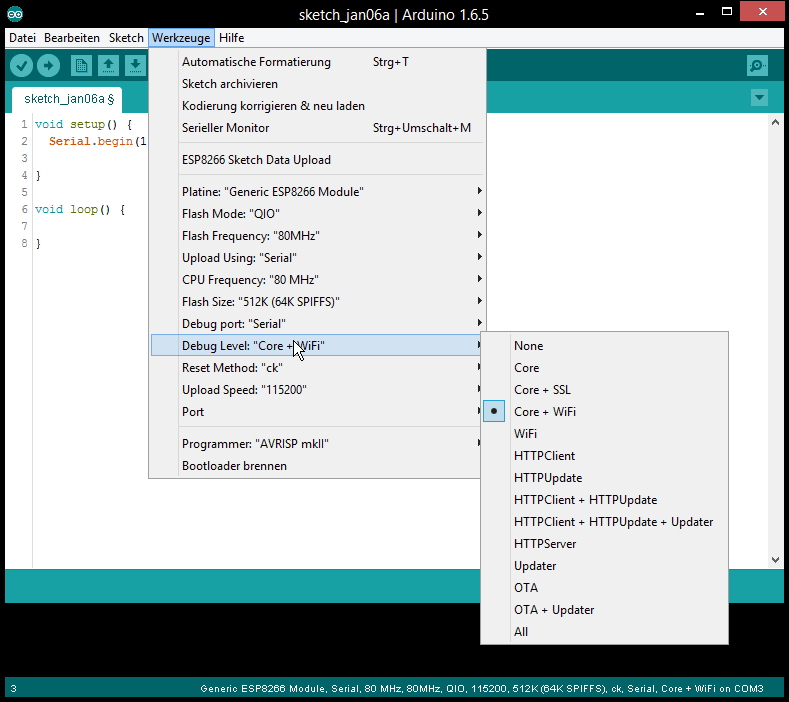
}

### Usage

1. Select the Serial interface for the Debugging messages:



1. Select which type / level you want debug messages for:



1. Check if the Serial interface is initialized in setup() (see[Requirements](http://arduino-esp8266.readthedocs.io/en/2.4.0/Troubleshooting/debugging.html#requirements))
2. Flash sketch
3. Check the Serial Output

## Informations

It work with every sketch that enables the Serial interface that is selected as debug port.

The Serial interface can still be used normal in the Sketch.

The debug output is additional and will not disable any interface from usage in the sketch.

### For Developers

For the debug handling uses defines.

The defined are set by command line.

#### Debug Port

The port has the define DEBUG\_ESP\_PORT possible value: - Disabled: define not existing - Serial: Serial - Serial1: Serial1

#### Debug Level

All defines for the different levels starts with DEBUG\_ESP\_

a full list can be found here in the [boards.txt](https://github.com/esp8266/Arduino/blob/master/boards.txt#L180)

#### Example for own debug messages

The debug messages will be only shown when the Debug Port in the IDE menu is set.

*#ifdef DEBUG\_ESP\_PORT*

*#define DEBUG\_MSG(...) DEBUG\_ESP\_PORT.printf( \_\_VA\_ARGS\_\_ )*

*#else*

*#define DEBUG\_MSG(...)*

*#endif*

void setup() {

Serial**.**begin(115200);

delay(3000);

DEBUG\_MSG("bootup...\n");

}

void loop() {

DEBUG\_MSG("loop %d\n", millis());

delay(1000);

}

# Stack Dumps

## Introduction

If the ESP crash the Exception Cause will be shown and the current stack will be dumped.

Example:

**Exception** (0): epc1**=**0x402103f4 epc2**=**0x00000000 epc3**=**0x00000000 excvaddr**=**0x00000000 depc**=**0x00000000

ctx: sys

sp: 3ffffc10 end: 3fffffb0 offset: 01a0

**>>>**stack**>>>**

3ffffdb0: 40223e00 3fff6f50 00000010 60000600

3ffffdc0: 00000001 4021f774 3fffc250 4000050c

3ffffdd0: 400043d5 00000030 00000016 ffffffff

3ffffde0: 400044ab 3fffc718 3ffffed0 08000000

3ffffdf0: 60000200 08000000 00000003 00000000

3ffffe00: 0000ffff 00000001 04000002 003fd000

3ffffe10: 3fff7188 000003fd 3fff2564 00000030

3ffffe20: 40101709 00000008 00000008 00000020

3ffffe30: c1948db3 394c5e70 7f2060f2 c6ba0c87

3ffffe40: 3fff7058 00000001 40238d41 3fff6ff0

3ffffe50: 3fff6f50 00000010 60000600 00000020

3ffffe60: 402301a8 3fff7098 3fff7014 40238c77

3ffffe70: 4022fb6c 40230ebe 3fff1a5b 3fff6f00

3ffffe80: 3ffffec8 00000010 40231061 3fff0f90

3ffffe90: 3fff6848 3ffed0c0 60000600 3fff6ae0

3ffffea0: 3fff0f90 3fff0f90 3fff6848 3fff6d40

3ffffeb0: 3fff28e8 40101233 d634fe1a fffeffff

3ffffec0: 00000001 00000000 4022d5d6 3fff6848

3ffffed0: 00000002 4000410f 3fff2394 3fff6848

3ffffee0: 3fffc718 40004a3c 000003fd 3fff7188

3ffffef0: 3fffc718 40101510 00000378 3fff1a5b

3fffff00: 000003fd 4021d2e7 00000378 000003ff

3fffff10: 00001000 4021d37d 3fff2564 000003ff

3fffff20: 000003fd 60000600 003fd000 3fff2564

3fffff30: ffffff00 55aa55aa 00000312 0000001c

3fffff40: 0000001c 0000008a 0000006d 000003ff

3fffff50: 4021d224 3ffecf90 00000000 3ffed0c0

3fffff60: 00000001 4021c2e9 00000003 3fff1238

3fffff70: 4021c071 3ffecf84 3ffecf30 0026a2b0

3fffff80: 4021c0b6 3fffdab0 00000000 3fffdcb0

3fffff90: 3ffecf40 3fffdab0 00000000 3fffdcc0

3fffffa0: 40000f49 40000f49 3fffdab0 40000f49

**<<<**stack**<<<**

The first number after Exception gives the cause of the reset. a full ist of all causes can be found [here](http://arduino-esp8266.readthedocs.io/en/2.4.0/exception_causes.html) the hex after are the stack dump.

### Decode

It’s possible to decode the Stack to readable information. For more info see the [Esp Exception Decoder](https://github.com/me-no-dev/EspExceptionDecoder) tool.

