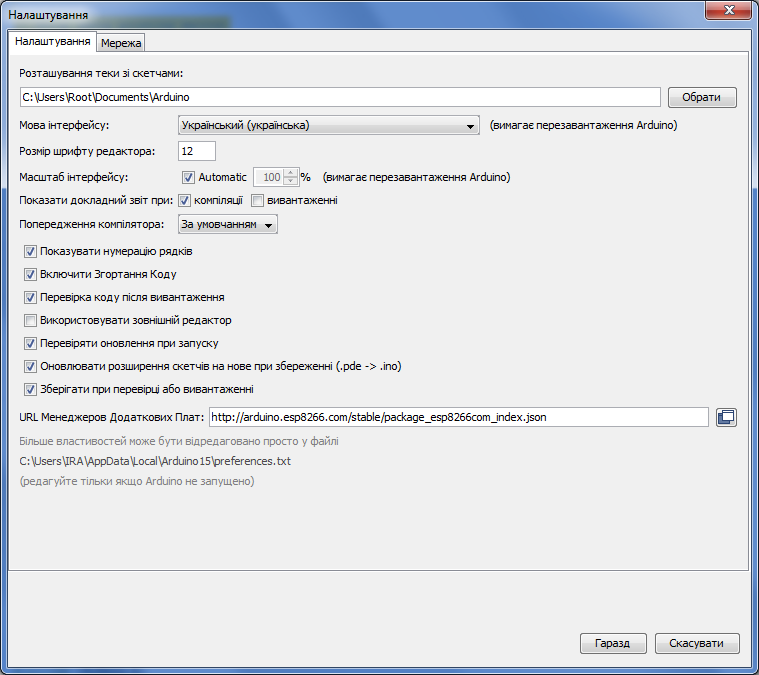
**Встановлення програмного забезпечення для роботи з контролером**

1. Завантажте та встановіть Arduino IDE з [офіційного веб-сайту](https://www.arduino.cc/en/Main/Software).
2. Запустіть Arduino IDE та відкрийте вікно налаштувань.
3. Додайте наступну URL стрічку в поле «Менеджер додаткових плат». <http://arduino.esp8266.com/stable/package_esp8266com_index>



1. Відкрийте **Менеджер плат** з меню **Інструменти> Плата:** та знайдіть платформу esp8266.
2. Виберіть найактуальнішу версію у спадному списку (на момент написання цієї документації це є версія 2.3.0).
3. Натисніть кнопку "Встановити".
4. Не забудьте обрати вашу конфігурацію плати ESP8266, за допомогою меню **Інструменти> Плата:** , після встановлення.

P.S. Також, за бажанням, ви можете використати тестовий пакунок менеджера плат: http://arduino.esp8266.com/staging/package\_esp8266com\_index.json.  Він може містити деякі нові функції, але в той же час деякі речі можуть неправильно працювати.

Список посилань: http://esp8266.github.io/Arduino/versions/2.3.0/

**Синтаксис**

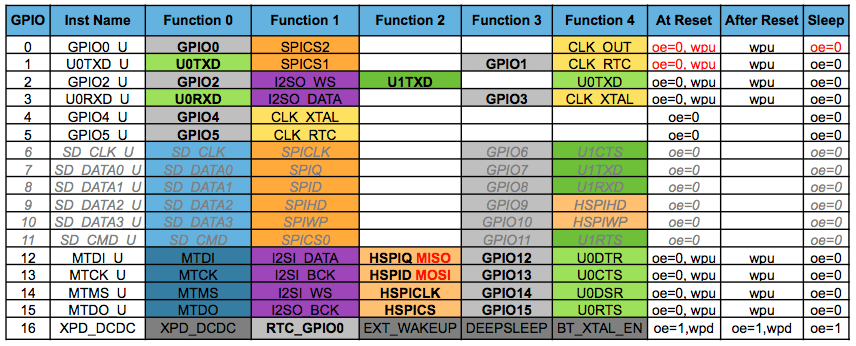
**Digital IO**

Існують спрощені функції налаштування – pinMode, зчитування стану – digitalRead та запису – digitalWrite. Номери цифрових входів/виходів в Arduino безпосередньо відповідають номерам GPIO ESP8266, так що щоб зчитати стан GPIO2 треба викликати функцію – digitalRead(2).

Цифрові контакти 0-15 може бути налаштовані на приймання – INPUT, виведення – OUTPUT або прийом з підтяжкою до логічної одиниці – INPUT\_PULLUP . GPIO16 не має налаштування – INPUT\_PULLUP , але натомість він має прийом з підтяжкою до логічної нуля – INPUT\_PULLDOWN\_16. По замовчанню всі GPIO налаштовані як – INPUT, але щоб розширити flash пам'ять більшість модулів ESP містить додаткову мікросхему пам’яті яка підключається до GPIO 6,7,8,9,10,11 і відповідно їх не варто використовувати, щоб мати змогу користуватися додатковою пам’яттю, також це може призвести до повного краху програми адже ця пам'ять використовується для запам’ятовування самої прошивки (контакти 9,10 можуть бути задіяні в якості IO, якщо флеш-чіп працює в режимі DIO (на відміну від QIO, який за замовчуванням один).

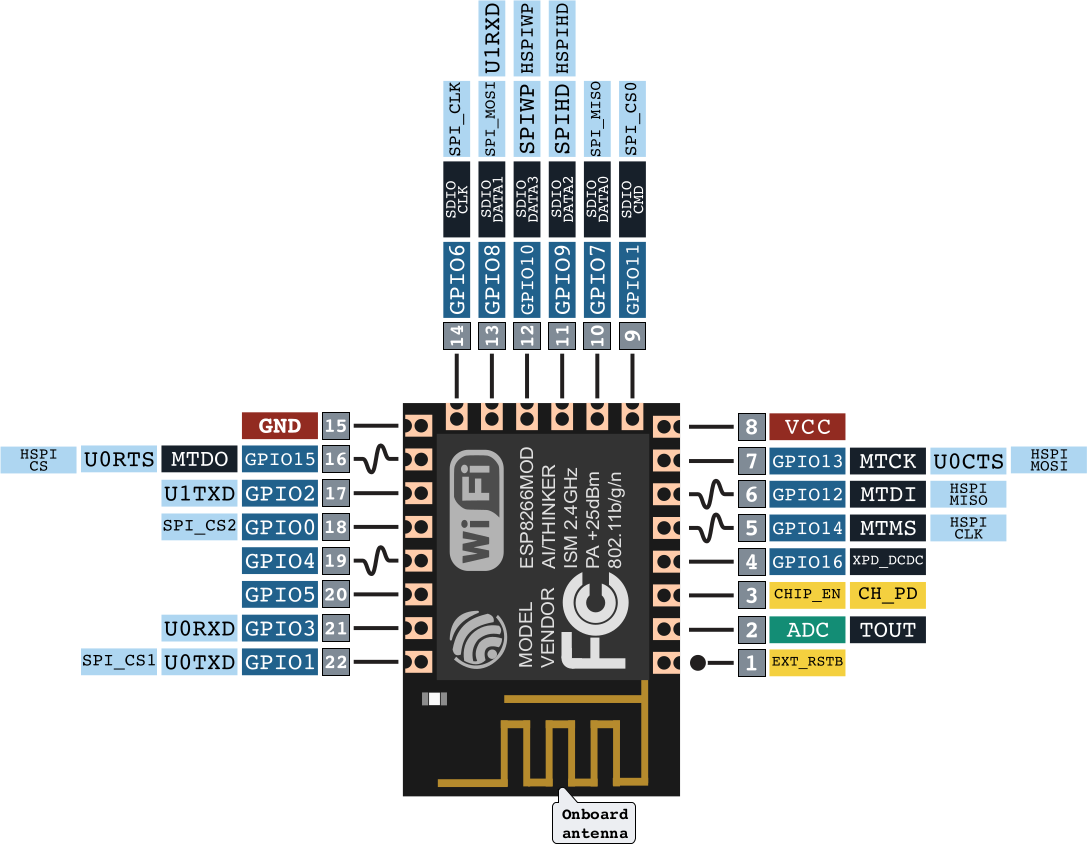
Насправді це далеко не всі налаштування GPIO, так наприклад існують спеціальні налаштування для конкретних виводів, щоб налаштувати апаратні інтерфейси обміну даних таких як SPI, I2C абощо. Ці функції, як правило, активується відповідною бібліотекою.

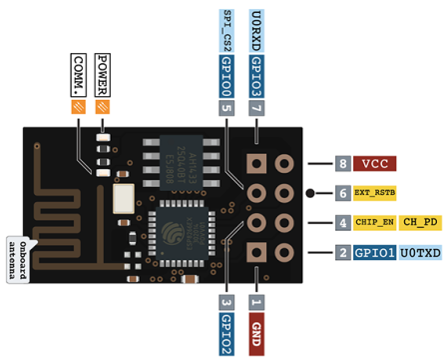
Функції контактів:



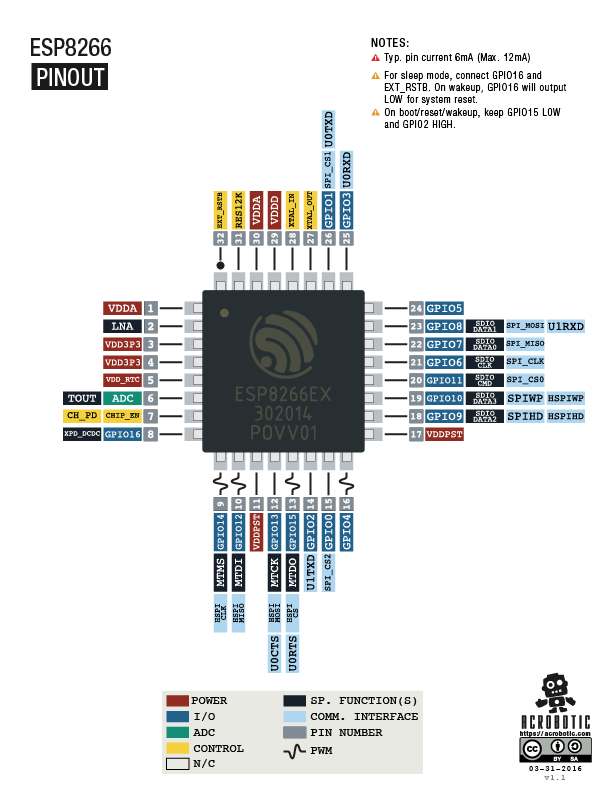
Переривання контактів задається за допомогою функції – attachInterrupt , а відключається за допомогою – detachInterrupt і можуть бути налаштовані на всіх GPIO окрім GPIO16. Підтримуються такі типи переривання: CHANGE,  RISING,  FALLING, так щоб налаштувати переривання по зміні рівня з 1 в 0 для GPIO2 треба викликати функцію – attachInterrupt(GPIO2,callback, FALLING); , де callback – це функція що буде викликана в результаті переривання.

На зображеннях нижче показана назви та призначення всіх контактів популярних модулів ESP-12 та ESP-1 відповідно.





Також ви можете роздивитися назви та призначення всіх контактів на самому чіпі.



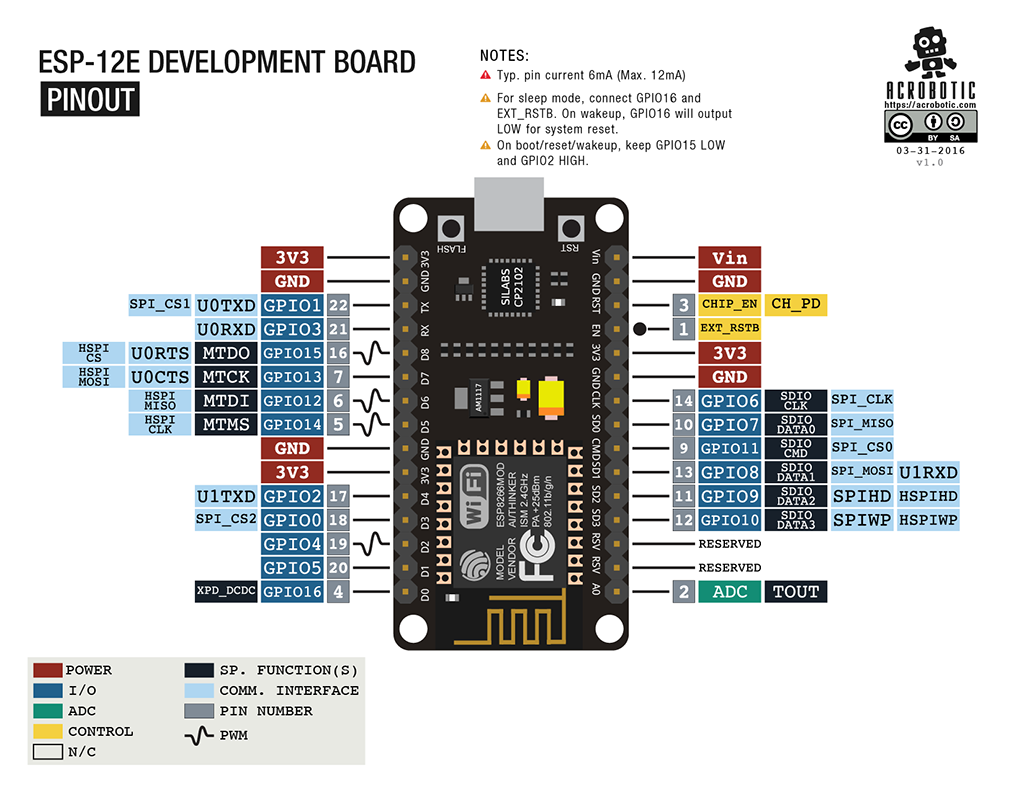
POWER – живлення I/O – контакти входу/виходу інформації

ADC – АЦП CONTROL – контакти для регулювання режимів

SP.FUN – спец функції COMM. INT – функції різних інтерфейсів

PWM – ШИМ PIN NUM – номер ніжки

І звичайно зображення плати NodeMCU



Як бачимо на картинці є застереження:

* Типове значення струму від GPIO 6mA (Max. 12mA)
* Для режиму сну з’єднайте GPIO16 та EXT\_RSTB. Щоб модуль прокинувся треба притягнути GPIO16 до лог. нуля, або перезавантажити модуль
* Щоб плата прокинулася, перезавантажилася тримайте значення GPIO15 лог. нуль та притягніть GPIO2 до лог. одиниці. (інколи так може вийти випадково тому слідкуйте за з’єднаннями)

**Analog input**

ESP8266 має лише один канал 10-розрядного АЦП. Він може бути використаний або для зчитування напруги з контакту АЦП, або для визначення напруги живлення модуля (VCC), що інколи може бути корисним.

Для зчитування зовнішньої напруги , що подається на АЦП, використовують функцію –analogRead(A0), що повертає значення 0-1023. Діапазон вхідної напруги 0 - 1.0V, але можна подавати на вхід напругу вище, рекомендовано до напруги живлення (VCC – 3.3V).

Для читання VCC напруги, використовується функція – ESP.getVcc() , при цьому ADC контакт повинен бути не підключеним! Крім того, обов’язково повинен бути додана команда налаштування ADC – ADC\_MODE ( ADC\_VCC ); що повинна бути додана до програми на початок програми, перед всіми функціями, наприклад, відразу після підключення всіх бібліотек (#include).

## Timing and delays

## Функції millis()і micros()повертають кількість мілісекунд і мікросекунд відповідно, що минули після скидання.

Функція delay(ms)призупиняє роботу основної програми на заданий час в мілісекундах і резервує цей час для обробки завдань WiFi та TCP/IP. Функція – delayMicroseconds(us) призначена для паузи на задане число мікросекунд

Пам'ятайте , що є багато задач, що повинні забезпечувати стабільну роботу Wi-Fi, і для них потрібно періодично виділяти час роботи. WiFi і TCP/IP бібліотека отримують можливість обробляти будь-які очікувані події кожен раз , коли функція loop()завершується, або коли викликається функція delay. Якщо у вас в програмі є будь яке оброблення, що займає багато часу (> 50 мс) без виклику delay, ви можете розглянути варіант спеціально додати виклик функції delay для підтримки нормального виконання стеку Wi-Fi.

Існує також функція yield(), яка еквівалентна delay(0). Функція  delayMicroseconds не поступається часом виконання іншим завданням, тому використання її для затримки більш ніж на 20 мілісекунд , не рекомендується.

**Serial**

Об'єкт Serial працює так само, як і в Arduino. Крім апаратного FIFO (128 байт для TX і RX) Serial має додаткові 256 байт для RX TX буферів. Як передавати і приймати це переривання. Запис і зчитування функції тільки блокувати ескіз виконання , коли відповідний FIFO / буфера повні / порожні. Зверніть увагу , що довжина додаткового 256-бітний буфер може бути змінена.

Serialвикористовує УАПП0, який відображається на штирі GPIO1 (TX) і GPIO3 (RX). Послідовний може бути перепризначений до GPIO15 (TX) і GPIO13 (RX) з допомогою виклику Serial.swap()після того, як Serial.begin. Виклик swapзнову карти UART0 назад GPIO1 і GPIO3.

Serial1використовує УАПП1, TX штифт GPIO2. UART1 не може бути використана для отримання даних , так як зазвичай це RX контактний зайнятий для підключення флеш - чіпа. Для використання Serial1, виклику Serial1.begin(baudrate).

Якщо Serial1не використовується , а Serialне місце - TX для UART0 може бути відображений в GPIO2 замість шляхом виклику Serial.set\_tx(2)після Serial.beginабо безпосередньо.Serial.begin(baud, config, mode, 2)

За замовчуванням діагностичний вихід з бібліотек WiFi відключається , коли ви телефонуєте Serial.begin. Щоб включити зневаджувальної знову, викличтеSerial.setDebugOutput(true). Для того, щоб перенаправити висновок налагодження Serial1 замість, викличте Serial1.setDebugOutput(true).

Крім того, необхідно використовуватиSerial.setDebugOutput(true)для того, щоб вихід з printf()функції.

Метод дозволяє визначити глибину приймального буфера. Значення за замовчуванням дорівнює 256.Serial.setRxBufferSize(size\_t size)

Як Serialі Serial1об'єкти підтримують 5, 6, 7, 8 біт даних, непарна (O), навіть (Е), і немає (N) , парності і 1 або 2 степових бита. Щоб встановити потрібний режим, виклик , і т.д.Serial.begin(baudrate, SERIAL\_8N1)Serial.begin(baudrate, SERIAL\_6E2)

Новий метод був реалізований на обох Serialі Serial1отримати поточне значення швидкості передачі даних. Щоб отримати поточну швидкість передачі даних, викликSerial.baudRate(), Serial1.baudRate(). Повертати а intпоточна швидкість.

Наприклад

**//** Set Baud rate to 57600

**//** Установити бод швидкість 57600

Serial**.**begin(57600);

**//** Get current baud rate

**//** Отримати поточну бод швидкість

int br **=** Serial**.**baudRate();

**//** Will print "Serial is 57600 bps"

**//** Надрукує "Serial is 57600 bps"

Serial**.**printf("Serial is %d bps", br);

Serialі Serial1об'єкти є екземплярамиHardwareSerialкласу.

Я зробив це і для офіційного ESP8266[послідовного програмного забезпечення](https://github.com/esp8266/Arduino/blob/master/doc/libraries.md#softwareserial)бібліотеки, побачити цей [запит тягнути](https://github.com/plerup/espsoftwareserial/pull/22) .

Зверніть увагу , що ця реалізація **тільки для плат на базі ESP8266** , і не буде працює з іншими платами Arduino.

## Progmem

Функції пам'яті програми працюють так само, як на регулярній Arduino; розміщуючи дані тільки для читання і рядків в пам'яті тільки для читання і звільняючи купу для вашого застосування. Важливою відмінністю є те , що на ESP8266 буквальні рядки не об'єднані. Це означає , що той же символьний рядок , певні всередині F("")і / або PSTR("")будуть займати місце для кожного екземпляра в коді. Так що вам потрібно буде управляти Дублікат струни самостійно.

Існує один додатковий допоміжний макрос , щоб зробити його простіше передати рядки до методів , які приймають називаються . Використання це допоможе зробити його простіше об'єднувати рядки. Чи не об'єднання рядків ...const PROGMEM\_\_FlashStringHelperFPSTR()

String response1;

response1 **+=** F("http:");

**...**

String response2;

response2 **+=** F("http:");

використовуючи FPSTR став би ...

const char HTTP[] PROGMEM **=** "http:";

**...**

{

String response1;

response1 **+=** FPSTR(HTTP);

**...**

String response2;

response2 **+=** FPSTR(HTTP);

}