**ESP8266 GPIO Pin Allocations**

# Basic Wiring functions

pinMode(), digitalRead(), digitalWrite(), analogWrite() work as usual.

Pin numbers correspond directly to the esp8266 GPIO pin numbers. To read GPIO2, call digitalRead(2);

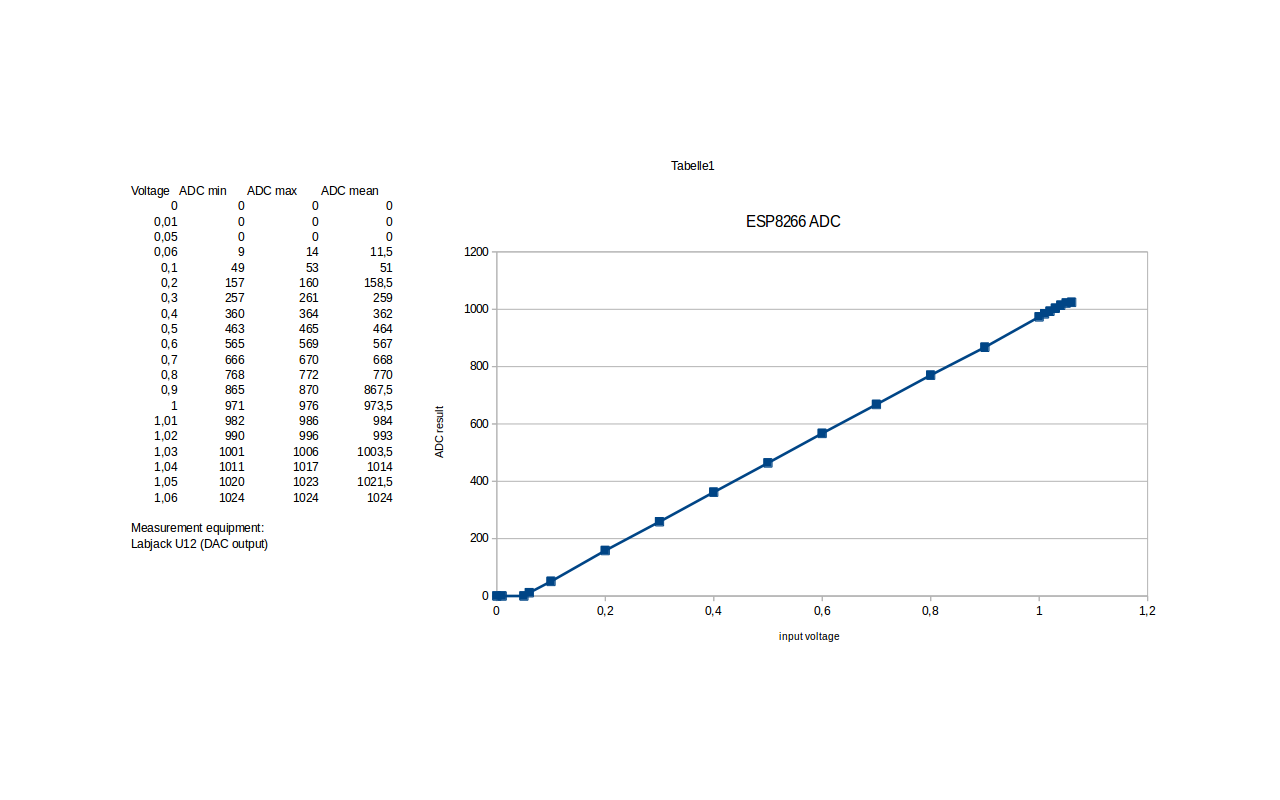
All digital IO pins are protected from over-voltage with a snap-back circuit connected between the pad and ground. The snap back voltage is typically about 6V, and the holding voltage is 5.8V. This provides protection from over-voltages and ESD. The output devices are also protected from reversed voltages with diodes.

At startup, pins are configured as INPUT.  
GPIO0-GPIO15 can be INPUT, OUTPUT, or INPUT\_PULLUP.  
GPIO16 can be INPUT, OUTPUT, or INPUT\_PULLDOWN\_16. It is also XPD for deepSleep() (perhaps via a small capacitor.)  
Note that GPIO6-GPIO11 are typically used to interface with the flash memory ICs on most esp8266 modules, so these pins should not generally be used.

analogRead(A0) reads the value of the ADC channel connected to the TOUT pin.

## Analog ADC

ESP8266EX also integrates a generic purpose 10-bit analog ADC. The ADC range is from 0V to 1.0V. It is typically used to measure the voltages from the sensor or battery status. The ADC cannot be used when the chip is transmitting. Otherwise the voltage may be inaccurate. (From Expressif datasheet CH 8.5)



### Interrupts

Pin interrupts are supported through attachInterrupt(), detachInterrupt() functions. Interrupts may be attached to any GPIO pin except GPIO16, but since GPIO6-GPIO11 are typically used to interface with the flash memory ICs on most esp8266 modules, applying interrupts to these pins are likely to cause problems. Standard Arduino interrupt types are supported: CHANGE, RISING, FALLING.

### PWM

analogWrite(pin, value) enables software PWM on the given pin. PWM may be used on pins 0 to 15. Call analogWrite(pin, 0) to disable PWM on the pin. Value may be in range from 0 to 1023. 0 to 255 is normal on an Arduino board, as it's an 8bit ADC, but ESP8266 is 10 bit so 1023 is full duty cycle.

## Pin Functions

The most usable pin functions are mapped to the macro SPECIAL, so calling pinMode(pin, SPECIAL) will switch that pin to UART RX/TX on pins 1 - 3, HSPI for pins 12-15 and CLK functions for pins 0, 4 and 5.

SPECIAL maps to:

* 0. CLK\_OUT
* 1. TX0
* 2. TX1
* 3. RX0
* 4. CLK\_XTAL
* 5. CLK\_RTC
* 12. SPI\_MISO
* 13. SPI\_MOSI
* 14. SPI\_CLK
* 15. SPI\_SS

You can activate any “FUNCTION” with pinMode(pin, FUNCTION1) for example.

Note: Func number 1-5 in Expressif table corresponds to FUNCTION 0-4 in SDK  
<http://bbs.espressif.com/download/file.php?id=442>

ESP8266 has three modes of operation: SDIO mode, UART mode and FLASH mode. These are obtained by pulling three pins either high or low at bootup: GPIO15, GPIO0, GPIO2 according to this table:

0 Low  
1 High  
x floating

| MODE | GPIO15 | GPIO0 | GPIO2 |
| --- | --- | --- | --- |
| SDIO (BootSDCard) | 1 | x | x |
| UART (UploadCode) | 0 | 0 | x or 1 |
| FLASH (NormalRunning) | 0 | 1 | x or 1 |

Note :- GPIO2 It is considered safer to pull it high with a resistor on boot rather than leave it floating to avoid possible chip damage if pin is pulled high by your program but also see [http://www.esp8266.com/viewtopic.php?f=6&t=3862&p=22524#p22524](http://www.esp8266.com/viewtopic.php?f=6&t=3862&p=22524" \l "p22524).

FLASH mode is when running the program. Take GPIO0 high or float or it will stall on first reset, intentional or not.

UART mode is how the code is uploaded to the chip and GPIO0 and GPIO15 must be low on boot to enter this mode.

SDIO mode is where the chip boots from an SD card. I don't think this is available to us yet.

## UART pins

Note that there's a special switch to swap UART0 RX/TX with UART0 CTS/RTS which is what systemuartswap does and is explained in [http://bbs.espressif.com/viewtopic.php?f=7&t=22&p=54&hilit=system\_uart\_swap#p54](http://bbs.espressif.com/viewtopic.php?f=7&t=22&p=54&hilit=system_uart_swap" \l "p54):

#define FUNC\_U0CTS 4

#define FUNC\_U0RTS 4

void user\_init(void) {

PIN\_FUNC\_SELECT(PERIPHS\_IO\_MUX\_MTCK\_U, FUNC\_U0CTS);//CONFIG MTCK PIN FUNC TO U0CTS

PIN\_FUNC\_SELECT(PERIPHS\_IO\_MUX\_MTDO\_U, FUNC\_U0RTS);//CONFIG MTDO PIN FUNC TO U0RTS

SET\_PERI\_REG\_MASK(0x3ff00028 , BIT2);//SWAP PIN : U0TXD<==>U0RTS(MTDO) , U0RXD<==>U0CTS(MTCK)

......

}

Original func:  
U0TXD⇒pin:U0TXD  
U0RXD⇒pin:U0RXD  
U0CTS⇒pin:MTCK  
U0RTS⇒pin:MTDO

After pin swap:  
U0TXD⇒pin:MTDO  
U0RXD⇒pin:MTCK  
U0CTS⇒pin:U0RXD  
U0RTS⇒pin:U0TXD

This allows the MTDO and MTCK to be connected to a microcontroller as UART tx/rx. The bootup info will be output via U0TXD pin, but after start-up, the UART0 will output data via MTDO and receive via MTCK.

kolban asserts: “The device writes to UART1 by itself in certain circumstances. First, when you flash the device, the data received is apparently also written to the TX pin of UART1 (GPIO2). Second, the 'debug' code in the device writes its output to UART1.”

If anyone has a reference for this please post it on the thread [http://www.esp8266.com/viewtopic.php?f=6&t=3862&p=22524#p22524](http://www.esp8266.com/viewtopic.php?f=6&t=3862&p=22524" \l "p22524)

## LED Pin

GPIO1, which is also TX, is wired to the blue LED on many devices. Note that the LED is active low (connected to Vcc and sinks through the chip to ground) so setting a logical value of 0 will light it up. Since GPIO1 is also the TX pin, you won't be able to blink the LED and perform Serial communications at thew same time unless you switch TX/RX pins.

## Maximum Current

When using a GPIO as output (i.e. to drive something such as an LED) it is important to note that the maximum output current is 12mA. (IMAX=12mA per Espressif datasheet) If you try and output more current than that, you run the risk of damaging the device. Since many LEDs are able to draw 20mA you should adjust your current limiting resistor to be 12mA or less. Using <https://en.wikipedia.org/wiki/LED_circuit> - R=(Vout-Vled)/I so (3.3V-1.8Vred) at 12mA = 125Ω min for a red LED.