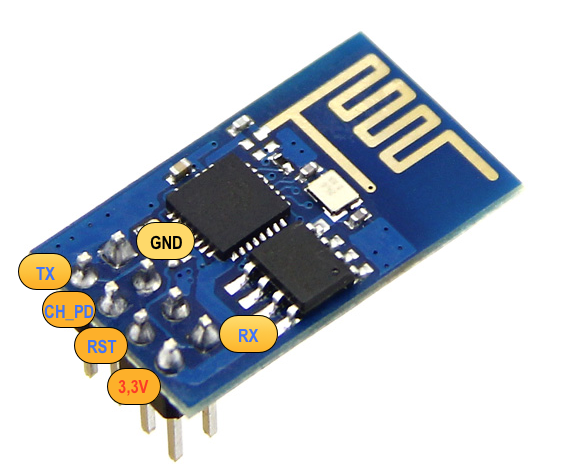
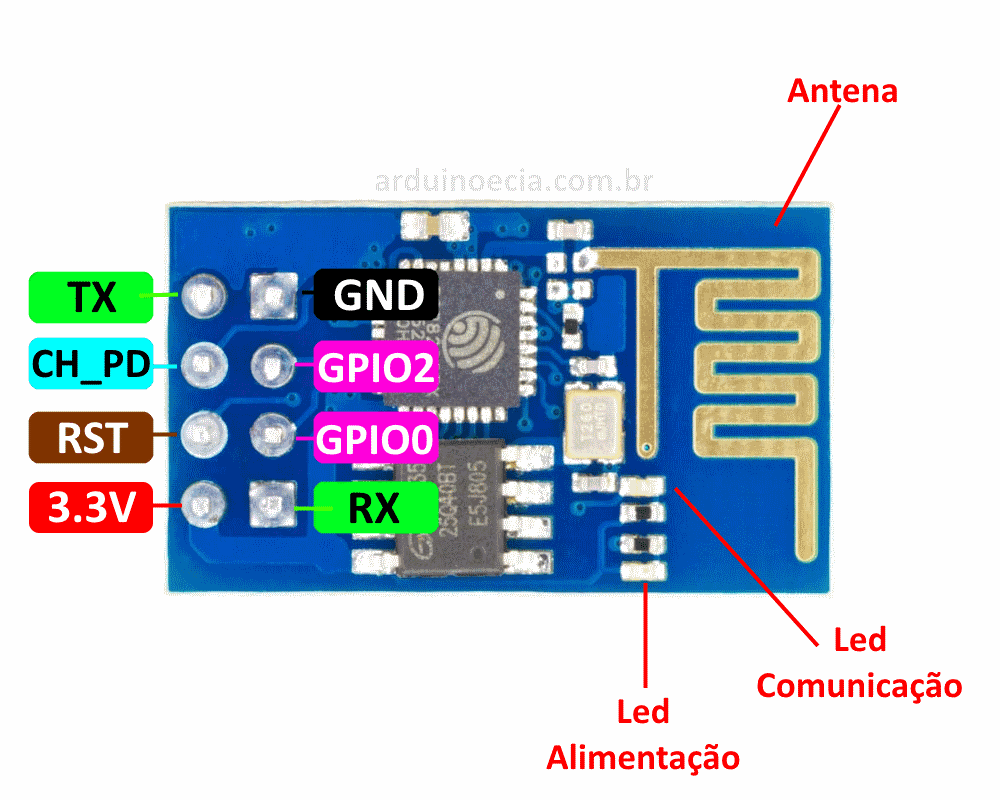
**Módulo Wireless ESP8266**

[ESP8266](https://www.filipeflop.com/blog/category/esp8266/), [Wireless](https://www.filipeflop.com/blog/category/wireless/) [23 de junho de 2015](https://www.filipeflop.com/blog/esp8266-arduino-tutorial/) [Adilson Thomsen](https://www.filipeflop.com/blog/author/adilsonth/)

Com o [Módulo Wireless ESP8266](https://www.filipeflop.com/produto/modulo-wifi-esp8266-esp-01/" \t "_blank) você pode conectar o seu Arduino nas redes wireless 802.11 b/g/n, enviando e recebendo dados nos modos AP (Access Point/Ponto de acesso) e STA (Station), e neste tutorial vamos mostrar como configurar esse módulo como web server, enviando dados para um browser.



Esse módulo se comunica com o microcontrolador utilizando interface serial e seu firmware pode ser atualizado, se necessário. Possui ainda 2 pinos **GPIO** (**General Purpose Input Output**, ou **Entrada e Saída de uso geral**), permitindo que o módulo seja programado diretamente e a GPIO acionada sem a necessidade de uso de um microcontrolador.



Outras características do **Módulo Wireless ESP8266**:

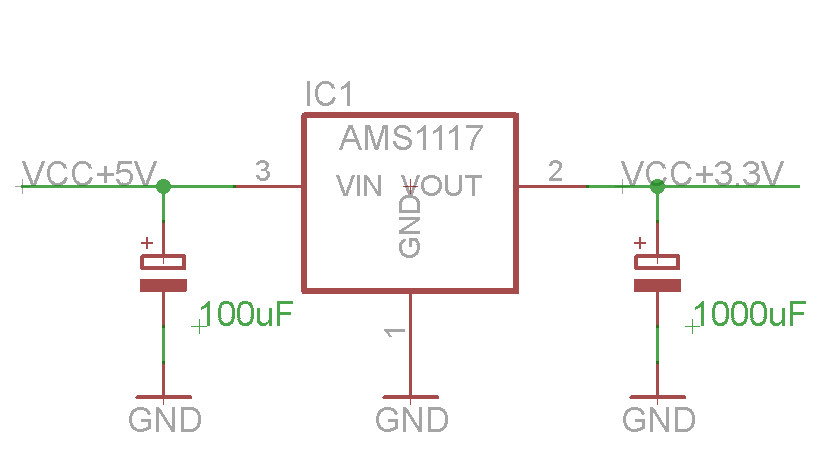
* Conexão à redes padrão 802.11 B/G/N
* Alcance aproximado: 91 metros
* Tensão de operação : 3.3 VDC
* Comunicação serial: pinos TX e RX
* Modos de operação : Cliente, Access Point, Cliente+Access Point
* Modos de segurança wireless : OPEN/WEP/WPA\_PSK/WPA2\_PSK/WPA\_WPA2\_PSK.
* Suporta comunicação TCP e UDP, com até 5 conexões simultâneas

#### Conectando o ESP8266 ao Arduino

O meu ESP-01 tem 8 pinos físicos, dos quais 6 foram usados neste experimento: GND, 3.3V, TX, RX, CH\_PD e RST. Os 2 primeiros da lista acima (GND e 3.3V) são os de alimentação elétrica, e foram conectados ao GND e 3V3 do Arduino, respectivamente. Os 2 últimos (CH\_PD e RST) foram conectados a pinos digitais, sendo que o primeiro é permanentemente mantido em HIGH, e o segundo fica LOW brevemente na inicialização do meu programa (causando um reset do ESP8266) e a partir daí fica permanentemente em HIGH.

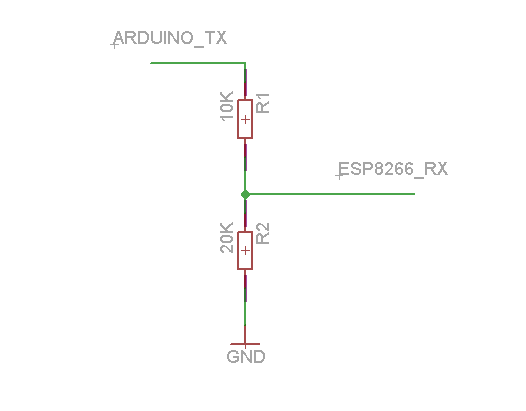
**Em conexões serias “clássicas”, o TX de um vai no RX do outro, e vice-versa.**

Já os pinos RX e TX são os responsáveis pela comunicação de dados entre Arduino e ESP8266. O RX recebe bits (e deve ser conectado ao TX do Arduino – o pino 1, no caso do Uno), e o TX envia bits (e deve ser conectado ao RX do Arduino - o pino 0, no caso do Uno).

Todas as conexões de alimentação, sinal e dados mencionadas precisam considerar a tensão do ESP8266, que é de 3,3V, bem abaixo dos 5V típicos dos pinos de dados do Arduino. Eu optei por usar um Arduino cuja tensão dos pinos de dados é configurável e tem opção de 3,3V, mas teria outras alternativas que talvez explore em artigos posteriores em Arduinos de 5V, como [divisores de tensão construídos com resistores](http://iot-playground.com/2-uncategorised/17-esp8266-wifi-module-and-5v-arduino-connection), ou mesmo CIs conversores de nível lógico. 

Atenção para o nível de sinal utilizado pelo módulo, que é de 3.3V, assim o pino RX (Recepção serial) não pode ser ligado diretamente ao Arduino. Você pode montar um divisor de tensão com dois resistores, utilizando o calculador [deste link](https://www.arduinoecia.com.br/p/calculador-divisor-de-tensao-function.html" \t "_blank).

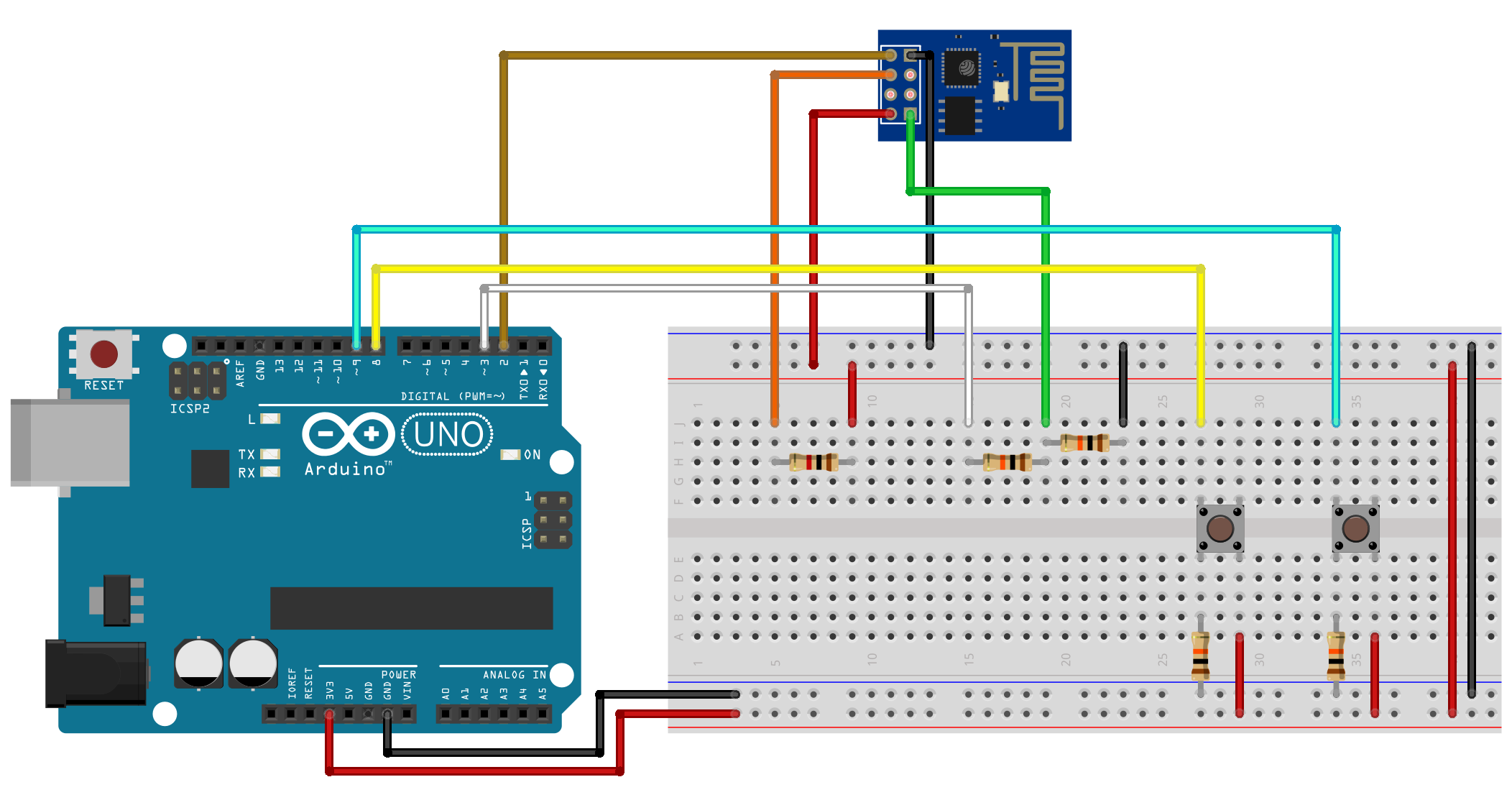
Arduino TX -> ESP8266 RX



Arduino DO pin -> ESP8266 reset

**Ligação do módulo ESP8266 ao Arduino**

Na ligação do ESP8266 com o Arduino usamos um resistor de 1K entre o Vcc (3.3V) e o pino **CH\_PD** (Chip Enable). Para o divisor de tensão, utilizamos 2 resistores de 10K, o que diminuiu a tensão do nível de sinal para um valor suficiente para os testes. Os dois push-buttons do circuito serão utilizados para enviar informações à uma página web, utilizando um web server. Os resistores utilizados nos botões também são de 10K.



Recomendamos a utilização de uma fonte externa para alimentação do módulo, pois dependendo da situação ele pode exigir até 300mA de corrente, e o limite do Arduino é de 50mA.

* **Programa web server ESP8266**

Antes de carregarmos o programa do web server, vamos alterar a velocidade de comunicação (baud rate) do módulo, que por padrão está setada em 115200 (firmware versão 0.9.5). Nessa velocidade, a biblioteca Software Serial não foi capaz de realizar a comunicação adequadamente, por isso alteramos a velocidade para **19200**, utilizando o programa abaixo.

Na linha 17 colocamos a velocidade padrão (115200), e na linha 25 setamos a nova velocidade (19200):

//           mudanca de baud rate

// Autor : FILIPEFLOP

#include <SoftwareSerial.h>

//RX pino 2, TX pino 3

SoftwareSerial esp8266(2, 3);

#define DEBUG true

void setup()

{

  Serial.begin(9600);

  // Configure na linha abaixo a velocidade inicial do

  // modulo ESP8266

  esp8266.begin(115200);

  sendData("AT+RSTrn", 2000, DEBUG);

  delay(1000);

  Serial.println("Versao de firmware");

  delay(3000);

  sendData("AT+GMRrn", 2000, DEBUG); // rst

  // Configure na linha abaixo a velocidade desejada para a

  // comunicacao do modulo ESP8266 (9600, 19200, 38400, etc)

  sendData("AT+CIOBAUD=19200rn", 2000, DEBUG);

  Serial.println("\*\* Final \*\*");

}

void loop() {}

String sendData(String command, const int timeout, boolean debug)

{

  // Envio dos comandos AT para o modulo

  String response = "";

  esp8266.print(command);

  long int time = millis();

  while ( (time + timeout) > millis())

  {

    while (esp8266.available())

    {

      // The esp has data so display its output to the serial window

      char c = esp8266.read(); // read the next character.

      response += c;

    }

  }

  if (debug)

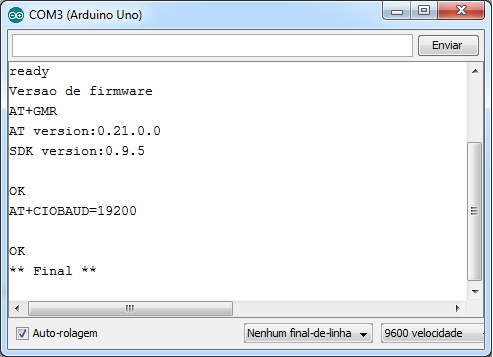
  {

    Serial.print(response);

  }

  return response;

}



No programa vamos utilizar a biblioteca **SoftwareSerial** para efetuar a comunicação com o módulo usando os pinos 2 (RX) e 3 (TX). Assim, podemos utilizar o serial monitor para acompanhar o envio dos comandos ao módulo. Na linha 18 do programa, substitua as informações de **SSID** (nome da rede wireless) e **SENHA** pelas informações da rede à qual o módulo irá se conectar.

// Alteracoes e adaptacoes: FILIPEFLOP

#include <SoftwareSerial.h>

//RX pino 2, TX pino 3

SoftwareSerial esp8266(2, 3);

#define DEBUG true

void setup()

{

  Serial.begin(9600);

  esp8266.begin(19200);

  sendData("AT+RSTrn", 2000, DEBUG); // rst

  // Conecta a rede wireless

  sendData("AT+CWJAP="SSID","SENHA"rn", 2000, DEBUG);

  delay(3000);

  sendData("AT+CWMODE=1rn", 1000, DEBUG);

  // Mostra o endereco IP

  sendData("AT+CIFSRrn", 1000, DEBUG);

  // Configura para multiplas conexoes

  sendData("AT+CIPMUX=1rn", 1000, DEBUG);

  // Inicia o web server na porta 80

  sendData("AT+CIPSERVER=1,80rn", 1000, DEBUG);

}

void loop()

{

  // Verifica se o ESP8266 esta enviando dados

  if (esp8266.available())

  {

    if (esp8266.find("+IPD,"))

    {

      delay(300);

      int connectionId = esp8266.read() - 48;

      String webpage = "<head><meta http-equiv=""refresh"" content=""3"">";

      webpage += "</head><h1><u>ESP8266 - Web Server</u></h1><h2>Porta";

      webpage += "Digital 8: ";

      int a = digitalRead(8);

      webpage += a;

      webpage += "<h2>Porta Digital 9: ";

      int b = digitalRead(9);

      webpage += b;

      webpage += "</h2>";

      String cipSend = "AT+CIPSEND=";

      cipSend += connectionId;

      cipSend += ",";

      cipSend += webpage.length();

      cipSend += "rn";

      sendData(cipSend, 1000, DEBUG);

      sendData(webpage, 1000, DEBUG);

      String closeCommand = "AT+CIPCLOSE=";

      closeCommand += connectionId; // append connection id

      closeCommand += "rn";

      sendData(closeCommand, 3000, DEBUG);

    }

  }

}

String sendData(String command, const int timeout, boolean debug)

{

  // Envio dos comandos AT para o modulo

  String response = "";

  esp8266.print(command);

  long int time = millis();

  while ( (time + timeout) > millis())

  {

    while (esp8266.available())

    {

      // The esp has data so display its output to the serial window

      char c = esp8266.read(); // read the next character.

      response += c;

    }

  }

  if (debug)

  {

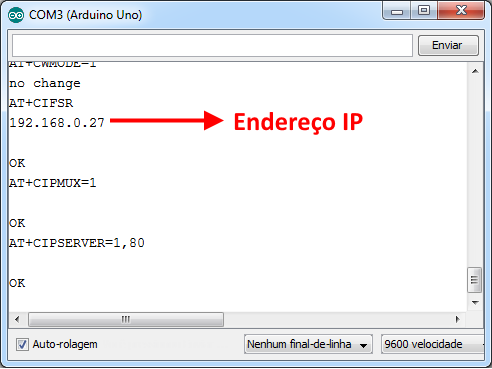
    Serial.print(response);

  }

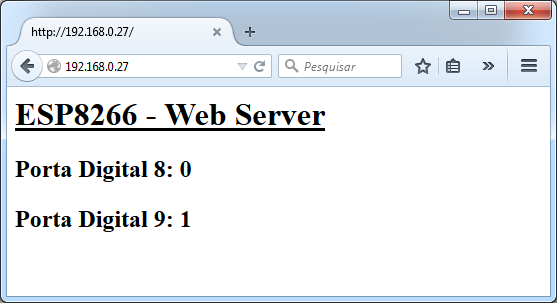
  return response;

}

Acompanhe no serial monitor as informações de conexão ao access point (AP) e também o endereço IP obtido pela placa.



Para testar o funcionamento do circuito, abra um browser (Chrome, Firexox, etc) e digite na barra de endereços o endereço IP que foi mostrado no passo anterior, no nosso caso, 192.168.0.27:



Acione os botões do circuito para que as informações correspondentes sejam enviadas ao browser. Devido à taxa de atualização, as informações podem demorar alguns segundos para aparecer na tela. Abaixo, temos uma imagem do circuito que utilizamos para montar este post:

