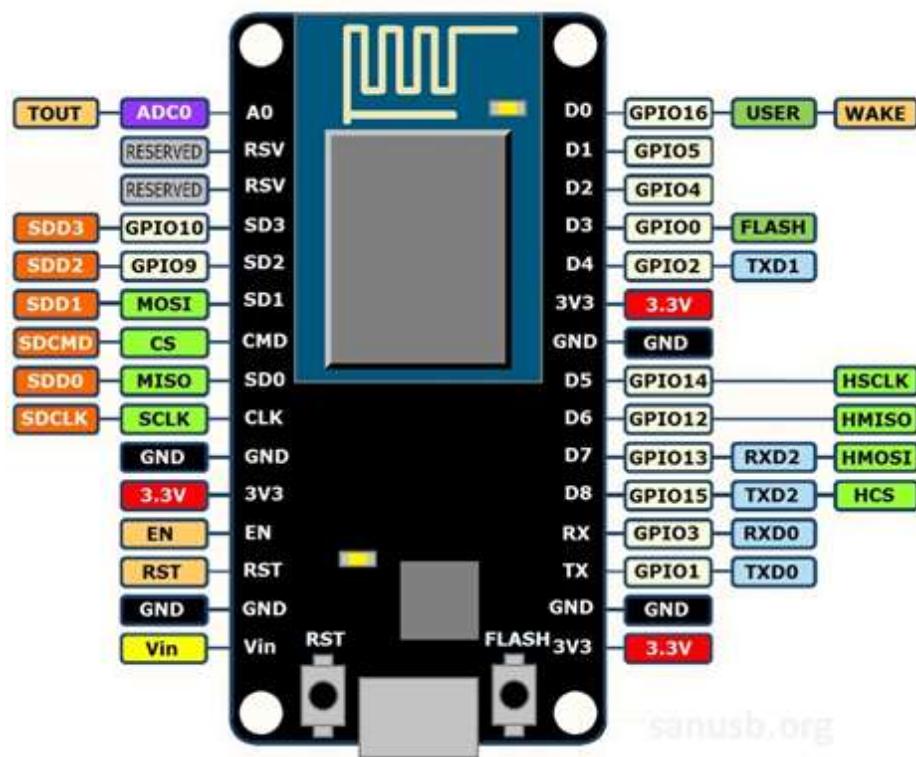


Aplicações Práticas de ESP8266em IoT



sanusb.org



Grupo SanUSB

*Dedicamos este trabalho
a Deus, às nossas famílias, a todos
os IFCEanos, amigos, alunos e integrantes
do grupo SanUSB.*

Sumário

INTRODUÇÃO	4
1.1 Hardware do módulo ESP8266.....	6
<i>Integrando o ESP8266 à IDE Arduino.....</i>	<i>7</i>
<i>Piscando o LED.....</i>	<i>11</i>
ATUALIZAÇÃO WIFI PELO BROWSER FUNCIONAL.....	16
<i>Instalando a IDE platformIO</i>	<i>36</i>
<i>Como utilizar.....</i>	<i>49</i>
INTRODUÇÃO A IoT COM MQTT	54
Caso prático de uso:	57
COMO RESTAURAR O FIRMWARE ORIGINAL COM COMANDOS AT	78
SERVIDOR IOT PUSH	83
Conexão via Putty.....	83

INTRODUÇÃO

O NodeMCU ESP-12E é uma versão integrada do popular ESP8266, um *Serial to Wi-Fi System On a Chip (SoC)*, que apareceu pela primeira vez em 2013 e lançado no mercado já no ano seguinte. O ESP8266 foi desenvolvido pela empresa chinesa com sede em Shanghai, *Espressif Systems*, uma fabricante de circuitos integrados focada no desenvolvimento de chips de RF, particularmente WiFi.

Os dois módulos mais conhecidos dessa família atualmente são a ESP-01 e o ESP-12E. O ESP-01 tem sido amplamente utilizado em projetos de IoT devido ao tamanho e custo, porém possui apenas 2 pinos digitais disponíveis. Por outro lado, o NodeMCU ESP-12E surge com mais pinos e periféricos. Algumas características do NodeMCU são descritas abaixo:

- Suporte a clientes TCP em múltiplos canais (máx. 5);
- Pinos GPIO (D0 a D8 e SD1 a SD3), PWM (D1 a D8), I2C, SPI e sensor interno de temperatura;
- Um conversor AD de 10 bits em AD0;
- Tensão de entrada Vin: 4.5V a 9V, e alimentação por USB;
- Corrente de trabalho: \approx 70mA(máx.200mA), standby menor que 200uA;
- Taxa de transmissão de dados: 110 a 460800bps;
- Suporte a atualização remota de firmware (OTA);
- Temperatura de trabalho:-40°C a +125°C ;
- Drive com duplaponte H;
- System-On-Chip com Wi-Fi embutido;
- CPU que opera em 80MHz, com possibilidade de operar em 160MHz;
- Arquitetura RISC de 32 bits;
- 32KBytes de RAM para instruções;
- 96KBytes de RAM para dados;
- 64KBytes de ROM para boot;
- Possui uma memória Flash SPI externa de 4Mbytes;

Existem módulos de diferentes tamanhos e fabricantes. Até o presente momento, existem módulos numerados de ESP-01 até ESP-12. O objetivo dos modelos ESP-01 e ESP-10 era servir como gateway Serial-WiFi, ou seja, de um lado eles recebem comandos AT via Serial (UART) e interagem com a rede WiFi por meio de conexões TCP/UDP. Os demais também podem operar nesse modo, embora sejam capazes de desempenhar mais funcionalidades, ou seja, como microcontroladores com Wi-Fi.

O módulo WiFi ESP8266 NodeMCU ESP-12E é uma das placas mais interessantes da família ESP8266, já que pode ser facilmente ligada à um computador e programada utilizando a IDE do Arduino, pela IDE platformIO, entre outras formas como por comandos AT ou com a linguagem Lua descritos no final do trabalho.

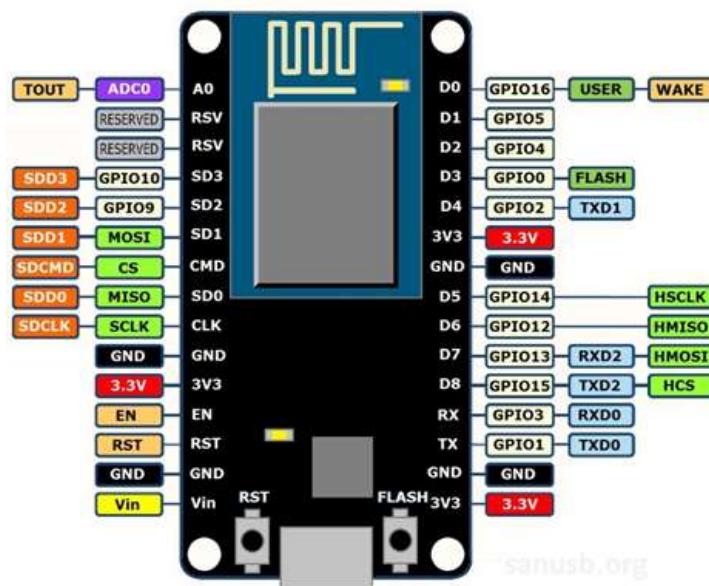


Figura 1: Placa ESP8266 NodeMCU

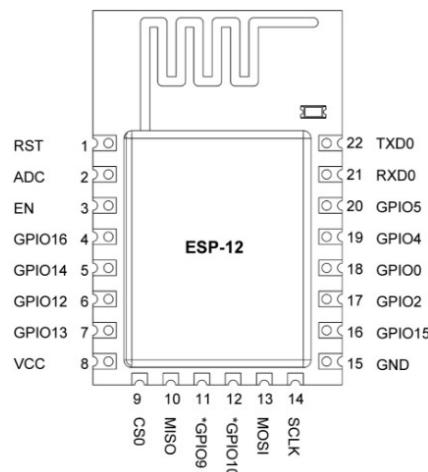


Figura 2: Chip ESP8266

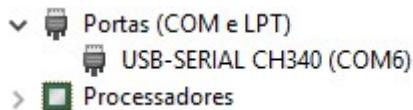
Essa placa possui suporta funções como PWM, I2C e 1-wire. Tem antena embutida, conversor USB-TLL integrado e o seu formato é ideal para ambientes de prototipação, encaixando facilmente em um *protoboard*.

1.1 Hardware do módulo ESP8266

O módulo Wifi ESP8266 NodeMCU tem dois botões, conforme mostrado na imagem abaixo: **Flash** (utilizado na gravação do firmware) e **RST** (Reset). No mesmo lado tem-se o conector USB para alimentação e conexão com o computador. O Led azul é ligado ao pino Tx que também é pino de GPIO 1.

No lado oposto, temos o ESP-12E e sua antena embutida, já soldado na placa. Nas laterais temos os pinos de GPIO, alimentação externa, comunicação, etc.

No Windows, a instalação e download de drivers é normalmente realizada de forma automática. Para verificar a instalação do adaptador USB-serial, acesse o Gerenciador de dispositivos clicando **Win+x**. Caso não esteja instalado, baixe e instale os drivers USB-serial em <http://sanusb.org/esp/driver>. Após a instalação, um dispositivo USB-TTL CH340 (NodeMCU 0.9) ou USB-TTL CP210 (NodeMCU 1.0) será adicionado ao gerenciador de dispositivos. Nesse exemplo, foi atribuída a porta COM6:



Vamos programar o módulo utilizando o Lua, e isso pode ser feito, à princípio, de duas maneiras.

A plataforma NodeMCU foi iniciada em 2014, menos de um ano após o começo das vendas do ESP8266, e contendo uma série de recursos adicionais para uso embarcado, e uma placa de desenvolvimento (de uso opcional) baseada no popular módulo ESP-12.

A imagem acima mostra um dos modelos do módulo de desenvolvimento do NodeMCU – note que na sua parte superior está presente um módulo ESP-12 completo. Além de apresentar os pinos do ESP-12 em uma configuração fácil de plugar em protoboards ou outros recursos de prototipação, o módulo de desenvolvimento também

oferece, entre outros confortos, a regulação de tensão, uma porta USB para a comunicação USB-serial, um botão de reset e um botão 'flash', cujo pressionamento coloca o ESP8266 no modo de *upload* de firmware.

Os módulos de desenvolvimento NodeMCU são didáticos, mas é possível utilizar também um ESP8266 12E comum.

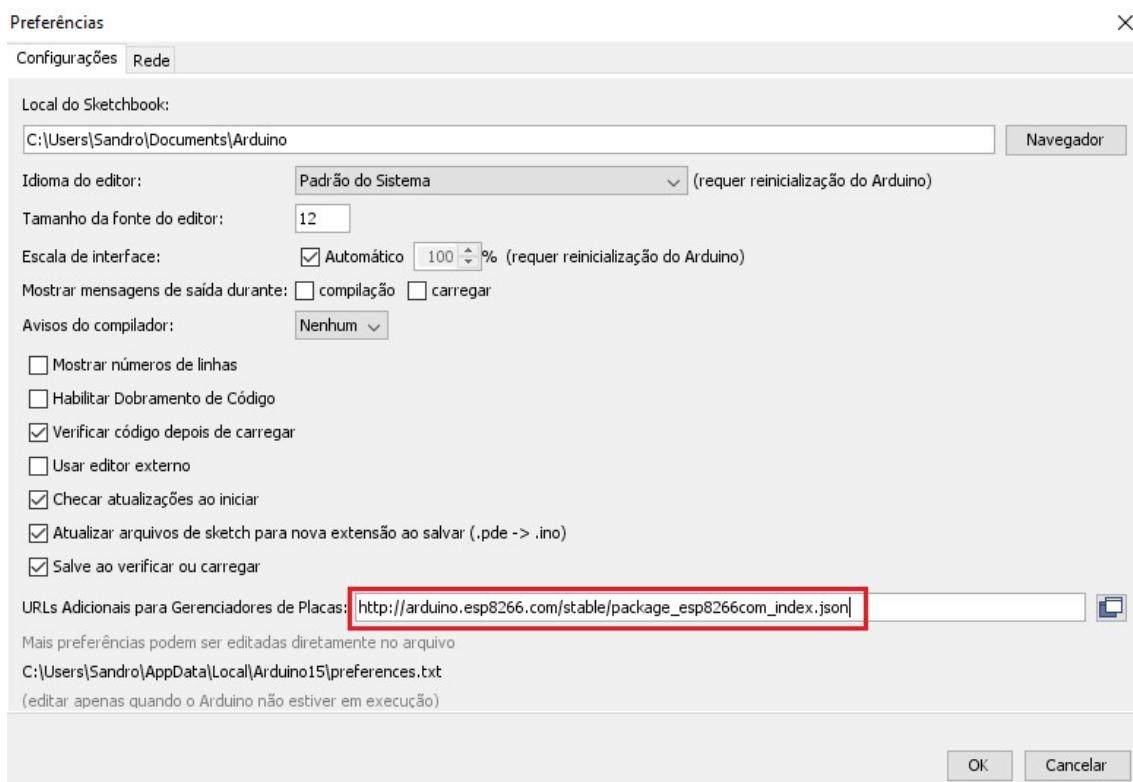
Dessa forma, o uso desse hardware é opcional. Tendo alguns jumpers e um adaptador USB-serial, é possível instalar o firmware NodeMCU na maioria dos módulos ESP8266, desde os modestos ESP01 até os avançados como o ESP12 e a cada vez maior gama de derivados e placas de desenvolvimento.

Integrando o ESP8266 à IDE Arduino

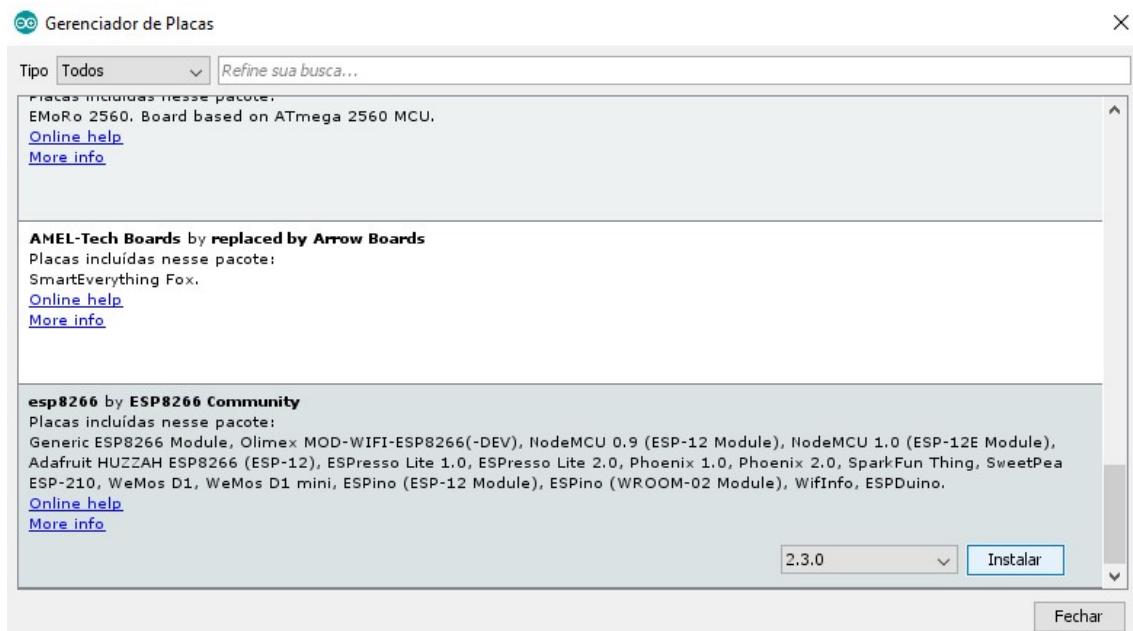
Para programar e usar o NodeMCU como se fosse um Arduino, a boa notícia é que é possível escrever-se firmwares personalizados e carregá-los no chip ("flash-it"). É importante lembrar que qualquer novo "firmware personalizado" irá substituir qualquer coisa previamente armazenada na memória flash do chip, incluindo o firmware original carregado em fábrica (aquele que aceita os comandos AT). Embora possamos usar o SDK do fabricante para o desenvolvimento de firmwares personalizados, é mais simples utilizar nesse caso a IDE Arduino. A IDE do Arduino torna muito mais fácil a programação do ESP8266, mantendo os comandos compatíveis com Arduino, basta fazer um #include das bibliotecas, sem se preocupar com detalhes do SDK da ESPRESSIF. Bibliotecas em <https://github.com/esp8266/Arduino/tree/master/libraries>.

Na IDE Arduino, abra a janela de preferências em Arquivo e digite a URL (marcado em vermelho na foto abaixo) no campo URLs adicionais para Gerenciadores de Placas e selecione OK.

http://arduino.esp8266.com/stable/package_esp8266com_index.json



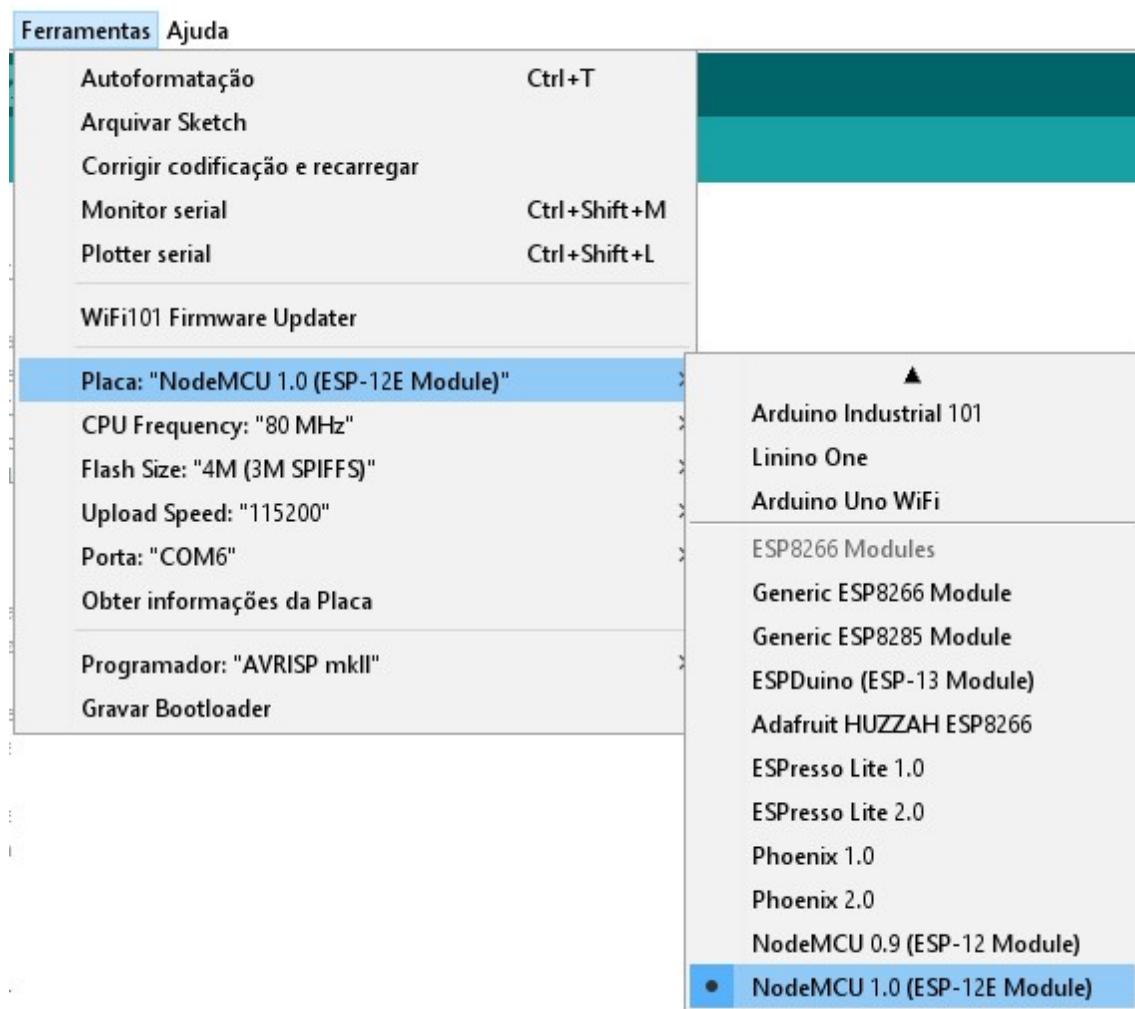
Selecione MENU → Ferramentas → Placas → Gerenciador de placas... e vá rolando até encontrar a opção: *esp8266 by ESP8266 Community*, a qual deverá ser o último item da lista e clique INSTALAR. É importante salientar que todas as bibliotecas instaladas ficam disponíveis em um diretório convencional como em ~\Documents\Arduino\libraries.



A principal diferença entre o NodeMCU 0.9 e NodeMCU 1.0, é que a versão do conversor USB-Serial do NodeMCU 0.9 é o chip CH340 e do NodeMCU 0.9 é o chip CP2102, para o qual deveremos instalar o driver Virtual COM Port (VCP). É possível

encontrar o driver apropriado CP2102 para o computador no seguinte link: sanusb.org/esp/driver.

Depois de restartar a IDE Arduino, é possível selecionar a placa no menu: **Tools → Ferramentas → Placa → NodeMCU 1.0 (ESP-12E Module)**. Em seguida, especificar a correta frequência de operação da CPU: (**Ferramentas → CPU Frequency: “” → 80MHz**) e velocidade de comunicação (**Ferramentas → Upload Speed: “” → 115,200**). Finalmente, selecionar o porto apropriado ao seu computador: (**Ferramentas → Porta → /COM6**).



Neste ponto estamos prontos para escrever nosso próprio firmware e enviá-lo ao dispositivo, mas vamos primeiramente tentar um dos exemplos incluídos com a biblioteca: **Arquivo → Exemplos → ESP8266WiFi → Blink**. Após o *upload*, podemos abrir a janela do *Serial Monitor* e observar os resultados. Verifique que 115.200 baud é a velocidade selecionada no menu do canto inferior direito do *Serial Monitor*.

FUNÇÕES BÁSICAS DOS PINOS

Funções como PinMode(), digitalWrite(), digitalRead(), analogWrite() operam normalmente. Os números dos pinos correspondem diretamente aos números de pinos do GPIO esp8266. Para ler GPIO2, chame digitalRead(2);

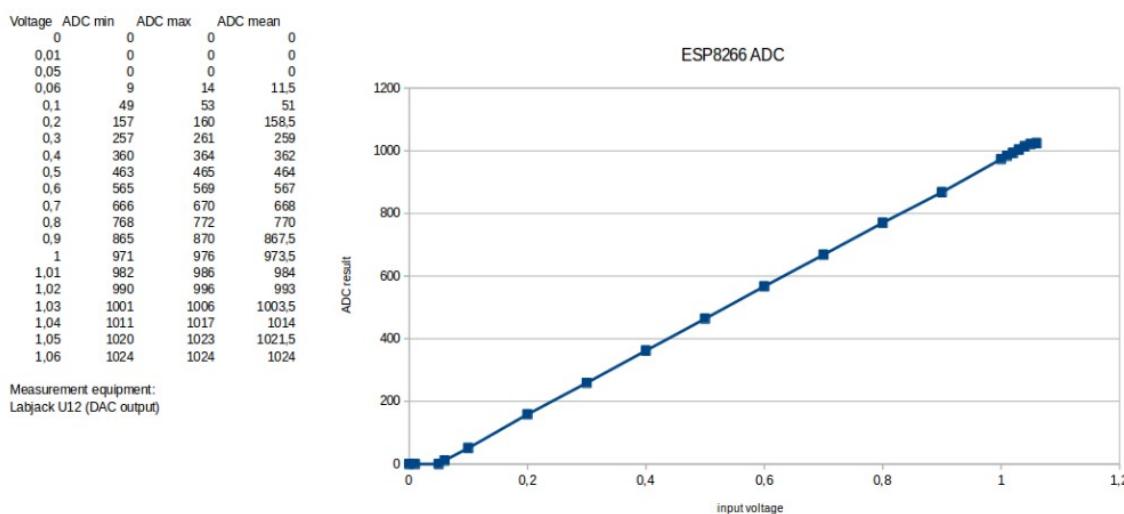
Todos os pinos digitais de IO são protegidos contra sobretensão com um circuito grampeador conectado entre o Vcc e o Gnd. Os dispositivos de saída também são protegidos contra tensões reversas com diodos.

Na inicialização, os pinos são configurados como INPUT. Os pinos GPIO0 a GPIO15 pode ser INPUT, OUTPUT ou INPUT_PULLUP.

O GPIO16 pode ser INPUT, OUTPUT ou INPUT_PULLDOWN_16. Observe que os pinos GPIO6 a GPIO11 são normalmente usados para interagir com os CIs de memória flash na maioria dos módulos esp8266. Dessa forma, esses pinos não devem ser usados e não estão disponíveis no módulo NodeMCU. É importante salientar que ao usar um GPIO como saída ligado a uma carga como um LED a corrente máxima de saída é 12mA.

Conversor analógico-digital

O ESP8266EX também integra um conversor AD (ADC) analógico de 10 bits de propósito genérico. A faixa de ADC é de 0V a 1.0V. É normalmente usado para medir as tensões a partir do estado do sensor ou da bateria. O ADC não pode ser usado quando o chip está transmitindo. Caso contrário, a tensão pode ser imprecisa. (Datasheet Expressif CH 8.5)



Interrupções

As interrupções de pin são suportadas através das funções `attachInterrupt()`, `detachInterrupt()`. As interrupções podem ser anexadas a qualquer pino GPIO, exceto o GPIO16. Os pinos GPIO6 a GPIO11 são normalmente utilizados para interface com a memória flash ICs na maioria dos módulos esp8266, a aplicação de interrupções para estes pinos são susceptíveis de causar problemas, por isso não são utilizados. Os tipos de interrupção padrão do Arduino são suportados: CHANGE, RISING, FALLING.

PWM

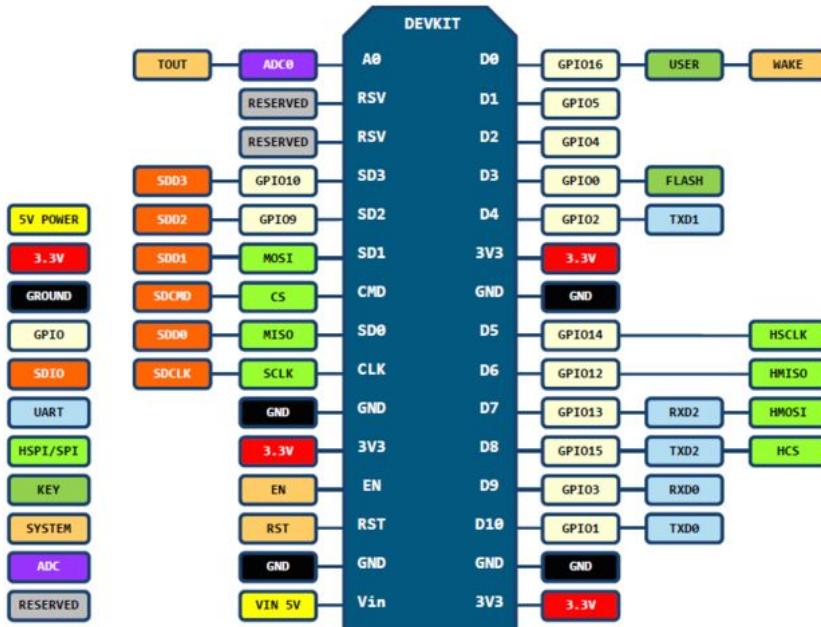
O comando `AnalogWrite(pino, valor)` permite implementar o software PWM no pino especificado. O PWM pode ser usado nos pinos 0 a 15. Chame `analogWrite(pino, 0)` para desabilitar o PWM no pino. O valor pode estar no intervalo de 0 a 1023.

LED

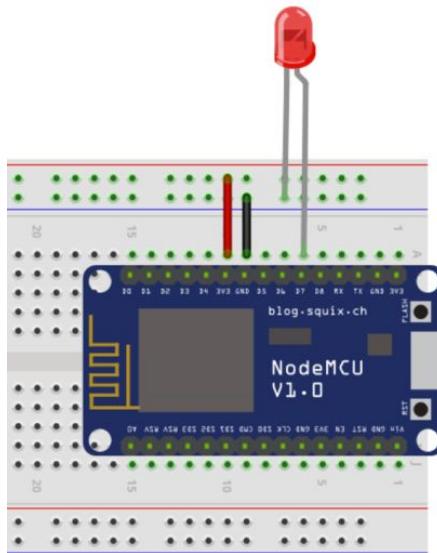
O GPIO1, que também é TX, é ligado ao LED azul em muitos dispositivos. Observe que o LED está conectado ao Vcc, ou seja, um valor lógico 0 irá acendê-lo. Como o GPIO1 é também o pino TX, não é possível piscar o LED e executar comunicações serial ao mesmo tempo, a menos que se alterne pinos TX / RX.

Piscando o LED

O “Olá Mundo” de qualquer novo projeto de sistema embarcado é sem dúvida alguma, um LED piscando. Para conectar-se um LED em seu ESP-12E, você poderá usar qualquer um dos seus GPIO digitais.



O diagrama de pinos acima mostra o layout da 2ª geração do NodeMCU ESP8266 . Em nosso caso, usaremos o pino D7 ou seu equivalente para Arduino: GPIO13.



Você poderá testar o código usando tanto “D7” quanto “13”, ambas formas funcionarão. O pino D7 não precisa de um resistor externo para alimentar o LED, pois possui um internamente. Abaixo um código simples para piscar o LED:

```
*****
Blink
Connected to pin D7 (GPIO13) ESP8266 NODEMCU
```

```
*****  

#define ledPin 13 // #define ledPin D7  

void setup()  

{  

    pinMode(ledPin, OUTPUT);  

}  

void loop()  

{  

    digitalWrite(ledPin, HIGH);  

    delay(1000);  

    digitalWrite(ledPin, LOW);  

    delay(1000);  

}
```

Há uma relação entre alguns dos pinos do NodeMCU e do Arduino, 13lente13camente identificados pelo IDE, tal como descrito abaixo:

Placa ESP → IDE Arduino (gpio)

- D0 → 16
- D1 → 5
- D2 → 4
- D3 → 0
- D4 → 2
- D5 → 14
- D6 → 12
- **D7** → **13**
- D8 → 15
- D9 → 3
- D10 → 1

Abaixo o código pronto para ser executado no IDE do Arduino como servidor de página HTML com acionamento dos pinos GPIO 12 e 13:

```
#include <ESP8266WiFi.h>

const char* ssid = "SETA-AF82";
const char* password = "14502384";

WiFiServer server(80);

void setup() {
    Serial.begin(115200);
    delay(10);

    // prepare GPIO 12 => D6 e GPIO 13 => D7
```

```
pinMode(12, OUTPUT);
digitalWrite(12, 0);

pinMode(13, OUTPUT);
digitalWrite(13, 0);

// Connect to WiFi network
Serial.println();
Serial.println();
Serial.print("Connecting to ");
Serial.println(ssid);

WiFi.begin(ssid, password);

while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}

Serial.println("");
Serial.println("WiFi connected");

// Start the server
server.begin();
Serial.println("Server started");
Serial.println(WiFi.localIP());
}

void loop() {
    WiFiClient 14liente = server.available();
    if (!client) {
        return;
    }

    Serial.println("new 14liente");
    while(!client.available()){
        delay(1);
    }

    String req = 14liente.readStringUntil('\r');
    Serial.println(req);
    14liente.flush();

    String buf = "";
    buf += "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE HTML>\r\n<html>\r\n";
}
```

```

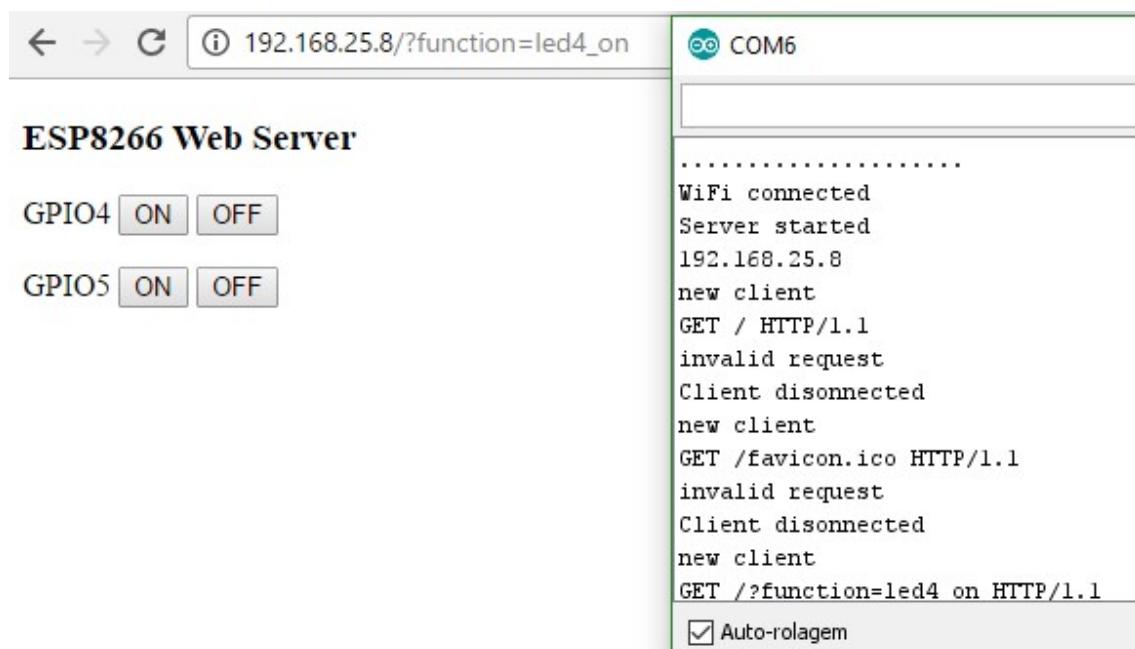
buf += "<h3> ESP8266 Web Server</h3>";
buf += "<p>GPIO12 (D6) <a href=\"?function=led6_on\"><button>ON</button></a>&nbsp;<a href=\"?function=led6_off\"><button>OFF</button></a></p>";
buf += "<p>GPIO13 (D7) <a href=\"?function=led7_on\"><button>ON</button></a>&nbsp;<a href=\"?function=led7_off\"><button>OFF</button></a></p>";
buf += "<h4>Teste On Off</h4>";
buf += "</html>\n";

15liente.print(buf);
15liente.flush();

if (req.indexOf("led7_on") != -1)
    digitalWrite(13, 1);
else if (req.indexOf("led7_off") != -1)
    digitalWrite(13, 0);
else if (req.indexOf("led6_on") != -1)
    digitalWrite(12, 1);
else if (req.indexOf("led6_off") != -1)
    digitalWrite(12, 0);
else {
    Serial.println("invalid request");
15liente.stop();
}
Serial.println("Client disconnected");
}

```

O *printscreen* abaixo ilustra a operação do código do ESP operando como servidor.
Mais detalhes desse exemplo no vídeo em: https://youtu.be/jt_elBF6YE



Para construir uma página web, embarcar em um servidor e acessá-la a partir de qualquer computador, é possível utilizar a ferramenta online RIB (<https://01.org/rib/online>), que é uma GUI de código aberto para construir páginas web gráficas. Para se tornar funcional, é necessário também adicionar funções JavaScript que abrem o websocket e enviam os dados.

MDNS

MDNS ou Multicast DNS, é uma maneira fácil de se comunicar com o ESP8266, usando um nome de host constante em vez de um endereço IP (que não é constante a menos que se use IP estático). A última versão do arduino IDE suporta mdns.

ATUALIZAÇÃO WIFI PELO BROWSER FUNCIONAL

No capítulo anterior foi apresentada a introdução à atualização via OTA, com o firmware transferido pela IDE do Arduino para o ESP8266. Nesse capítulo é mostrado como fazer a atualização via browser. Para essa implementação serão utilizadas duas classes principais, sendo a `ESP8266mDNS` e a `ESP8266WebServer`. Por pouco, os includes não ocupam mais espaço que o código necessário para habilitar o servidor no ESP.

As atualizações descritas neste capítulo são feitas com um navegador da Web que pode ser útil no caso de carregar o firmware binário diretamente de uma página Web. A biblioteca para este fim é a `ESP8266HTTPUpdateServer` que cria um servidor HTTP no ESP8266, onde você pode carregar o firmware binário via WiFi. Isso basicamente significa que você pode atualizar o código “Sobre o ar”. Abaixo um exemplo básico de código para gravação OTA pelo browser. As funções em negrito referem-se a parte do código para atualização do firmware via Browser.

```
#include <ESP8266WiFi.h> //Video: https://www.youtube.com/watch?v=wG1I7JeEXE8
#include <WiFiClient.h>
#include <ESP8266WebServer.h>
#include <ESP8266mDNS.h>
#include <ESP8266HTTPUpdateServer.h>

const char* host = "esp8266-sanusb";
const char* update_path = "/update";
const char* update_username = "sanusb";
```

```
const char* update_password = "laese";
const char* ssid = "RFID";
const char* password = "l43s3rf1d";

ESP8266WebServer httpServer(80);
ESP8266HTTPUpdateServer httpUpdater;
void setup(void);
void loop(void);
void setup(void){

    Serial.begin(115200);
    Serial.println();
    Serial.println("Booting Sketch...");
WiFi.mode(WIFI_AP_STA);//Permite gravar o OTA automaticamente
    WiFi.begin(ssid, password);

    while(WiFi.waitForConnectResult() != WL_CONNECTED){
        delay(500);
        Serial.print(".");
    }

MDNS.begin(host);

httpUpdater.setup(&httpServer, update_path, update_username, update_password);
httpServer.begin();

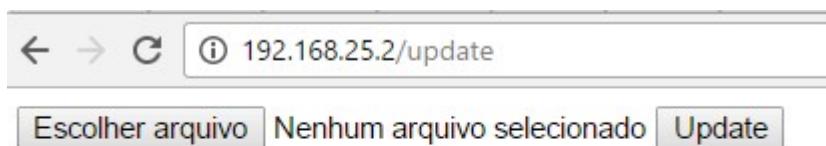
MDNS.addService("http", "tcp", 80);
    Serial.print("Atualizacao remota (OTA) iniciada com sucesso! Abra http://");
    Serial.print(WiFi.localIP());
    Serial.println("/update no browser");
    //Serial.printf("HTTPUpdateServer ready! Open http://%s.local%s in your browser and login with username '%s'
    //and password '%s\n", host, update_path, update_username, update_password);
}

void loop(void){
    httpServer.handleClient();

    digitalWrite(13, 0); digitalWrite(12, 1);// turn the LED on
    delay(1500); // wait
    digitalWrite(13, 1); digitalWrite(12, 0);// turn the LED off
    delay(1500);
    Serial.println("SanUSB-esp");
}
```

O código contém uma proteção contra gravação que necessita de senha. Foi utilizado nesse exemplo a compilador PlatformIO, como pode ser visto no vídeo <https://www.youtube.com/watch?v=wG1I7JeEXE8>. O PlatformIO gera automaticamente o binário em **.pioenvs\esp12e\firmware.bin** que pode ser utilizado para gravação *Over the Air* através da biblioteca WebUpdater. Após gravar do primeiro firmware de atualização um upload utilizando a COM convencional de gravação e clicar na serial para depurar a resposta aparecerá a seguinte informação:

Depois de gerar o arquivo.bin, em **sketch-> exportar binário compilado**, digite no seu navegador o nome ou IP do host, adicione /update e aparecerá esta página:



Basta selecionar o arquivo binário e clicar em update e o arquivo será transferido. Vale salientar que, depois de gerar o arquivo.bin, esse modo de atualização/gravação do firmware tende a ser mais rápido do que o upload convencional em modo serial. Mais detalhes no vídeo: https://www.youtube.com/watch?v=hqIJf_xMa0E.

Nota: Como foi visto, se o endereço DNS "<http://esp8266-sanusb.local/update>" não funcionar, tente substituir o DNS pelo endereço IP do módulo como na figura acima. Abaixo é mostrado o código do exemplo do vídeo que implementa a gravação/atualização do firmware via WiFi e realiza o controle também de um LED RGB simples sem resistor através de uma página hospedada no esp8266. Mais detalhes em: <https://www.youtube.com/watch?v=RzqR5tQoEVM>.

```
#define DEBUGGING(...) Serial.println( __VA_ARGS__ )
#define DEBUGGING_L(...) Serial.print( __VA_ARGS__ )

#include <ESP8266WiFi.h> //https://www.youtube.com/watch?v=RzqR5tQoEVM
#include <ESP8266WebServer.h>
```

```

#include <ESP8266mDNS.h> //Biblioteca que permite chamar o seu modulo ESP8266 na sua rede pelo nome ao inves do IP.
#include <ESP8266HTTPUpdateServer.h> //Biblioteca que cria o servico de atualizacão via wifi (ou Over The Air - OTA)

//Habilitando a saída serial com as mensagens de debugging
#ifndef DEBUGGING
#define DEBUGGING(...)
#endif

#ifndef DEBUGGING_L
#define DEBUGGING_L(...)
#endif

#define BLUEPIN 12 //Pino D5 do NodeMCU
#define GREENPIN 13 //Pino D6 do NodeMCU
#define REDPIN 14 //Pino D7 do NodeMCU

#define INICAR_CORES_ALEATORIAS 15000 //Tempo ocioso antes de começar a trocar as cores automaticamente

unsigned long ultimoAcessoHost = 0;
unsigned long ultimaTrocaCor = 0;
unsigned long ultimaTrocaCorAutomatica = 0;
boolean trocaAutomatica = 1; //Se voce mudar para 0 (zero), ira desligar a troca automatica de cores.
const char* host = "sanusb-esp"; //Nome que seu ESP8266 (ou NodeMCU) tera na rede
const char* ssid = "SETA-AF82"; //Nome da rede wifi da sua casa
const char* password = "14502384"; //Senha da rede wifi da sua casa
int RGB[3];
int cnt = 0;
int tempoTrocaCor = 50; //Velicidade que as cores trocam automaticamente
ESP8266HTTPUpdateServer atualizadorOTA; //Este e o objeto que permite atualizacao do programa via wifi (OTA)
ESP8266WebServer servidorWeb(80); //Servidor Web na porta 80

//Esta e a pagina enviada para o navegador de internet
String paginaWeb = ""
"<!DOCTYPE html><html><head><title>Controle de Fita de LED RGB</title>"
"<meta name='mobile-web-app-capable' content='yes' />"
"<meta name='viewport' content='width=device-width' />"
"</head><body style='margin: 0px; padding: 0px;'>"
"<canvas id='colorspace'></canvas>"
"</body>"
"<script type='text/javascript'>"
"(function () {"
"    var canvas = document.getElementById('colorspace');"
"    var ctx = canvas.getContext('2d');"
"    function drawCanvas() {"
```

```
" var colours = ctx.createLinearGradient(0, 0, window.innerWidth, 0);"
" for(var i=0; i <= 360; i+=10) {
"   colours.addColorStop(i/360, 'hsl(' + i + ', 100%, 50%)');
" }
" ctx.fillStyle = colours;
" ctx.fillRect(0, 0, window.innerWidth, window.innerHeight);
" var luminance = ctx.createLinearGradient(0, 0, 0, ctx.canvas.height);
" luminance.addColorStop(0, '#ffffff');
" luminance.addColorStop(0.05, '#ffffff');
" luminance.addColorStop(0.5, 'rgba(0,0,0,0)');
" luminance.addColorStop(0.95, '#000000');
" luminance.addColorStop(1, '#000000');
" ctx.fillStyle = luminance;
" ctx.fillRect(0, 0, ctx.canvas.width, ctx.canvas.height);
"
" var eventLocked = false;
" function handleEvent(clientX, clientY) {
"   if(eventLocked) {
"     return;
"   }
"   function colourCorrect(v) {
"     return Math.round(1023-(v*v)/64);
"   }
"   var data = ctx.getImageData(clientX, clientY, 1, 1).data;
"   var params = [
"     'r=' + colourCorrect(data[0]),
"     'g=' + colourCorrect(data[1]),
"     'b=' + colourCorrect(data[2])
"   ].join('&');
"   var req = new XMLHttpRequest();
"   req.open('POST', '?' + params, true);
"   req.send();
"   eventLocked = true;
"   req.onreadystatechange = function() {
"     if(req.readyState == 4) {
"       eventLocked = false;
"     }
"   }
" }
" }
" }

" canvas.addEventListener('click', function(event) {
"   handleEvent(event.clientX, event.clientY, true);
" }, false);
" canvas.addEventListener('touchmove', function(event){
"   handleEvent(event.touches[0].clientX, event.touches[0].clientY);
" }, false);
" function resizeCanvas() {
```

```

" canvas.width = window.innerWidth;"  

" canvas.height = window.innerHeight;"  

" drawCanvas();"  

" }"  

" window.addEventListener('resize', resizeCanvas, false);"  

" resizeCanvas();"  

" drawCanvas();"  

" document.ontouchmove = function(e) {e.preventDefault();}"  

" })();"  

"</script></html>";  
  

//////////////////////////////////////////////////////////////////////////  
  

void setup() {  

    //Se vc ativou o debugging, devera descomentar esta linha abaixo tambem.  

    Serial.begin(115200);  

    InicializaPinos();  

    InicializaWifi();  

    InicializaMDNS();  

    InicializaServicoAtualizacao();  

}  
  

//////////////////////////////////////////////////////////////////////////  
  

void loop() {  

    if (WiFi.status() != WL_CONNECTED) {  

        InicializaWifi();  

        InicializaMDNS();  

    }  

    else {  

        if (millis() - ultimoAcessoHost > 10) {  

            servidorWeb.handleClient();  

            ultimoAcessoHost = millis();  

        }  

        /*  

        if (trocaAutomatica && (millis() - ultimaTrocaCor > INICAR_CORES_ALEATORIAS) && (millis() -  

ultimaTrocaCorAutomatica > tempoTrocaCor))  

{  

    ultimaTrocaCorAutomatica = millis();  

    CoresAleatorias(cnt++, RGB);  

}  

*/
    }  

}

```

```

void RecepcaoClienteWeb() {
    String red = servidorWeb.arg(0);
    String green = servidorWeb.arg(1);
    String blue = servidorWeb.arg(2);

    /*
    analogWrite(REDPIN, 1023 - red.toInt()); //anodo comum com 1
    analogWrite(GREENPIN, 1023 - green.toInt());
    analogWrite(BLUEPIN, 1023 - blue.toInt());
    */

    analogWrite(REDPIN, red.toInt()); //catodo comum - acende com 0
    analogWrite(GREENPIN, green.toInt());
    analogWrite(BLUEPIN, blue.toInt());

    ultimaTrocaCor = millis();

    servidorWeb.send(200, "text/html", paginaWeb);
}

///////////////////////////////



void InicializaServicoAtualizacao() {
    atualizadorOTA.setup(&servidorWeb);
    servidorWeb.begin();
    DEBUGGING_L("O servico de atualizacao remota (OTA) Foi iniciado com sucesso! Abra http://");
    DEBUGGING_L(host);
    DEBUGGING(".local/update no seu browser para iniciar a atualizacao\n");
}

///////////////////////////////



void InicializaWifi() {
    //WiFi.mode(WIFI_AP_STA);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED){
        delay(500);
        Serial.print(".");
    }
    servidorWeb.on("/", RecepcaoClienteWeb);

    DEBUGGING("Conectado!");
    DEBUGGING(WiFi.localIP());
}

```

```

void InicializaMDNS() {
    if (!MDNS.begin(host)) {
        DEBUGGING("Erro ao iniciar o servico mDNS!");
        while (1) {
            delay(1000);
        }
    }
    DEBUGGING("O servico mDNS foi iniciado com sucesso!");
    MDNS.addService("http", "tcp", 80);
}

```

||||||||||||||||||||||||||||||||||||||||||||||||

```

void InicializaPinos(){
    //Iniciando os pinos como saida e com o valor alto (ligado)
    pinMode(REDPIN, OUTPUT);
    pinMode(GREENPIN, OUTPUT);
    pinMode(BLUEPIN, OUTPUT);

    analogWrite(REDPIN, HIGH);
    analogWrite(GREENPIN, HIGH);
    analogWrite(BLUEPIN, HIGH);

    delay(1000);
}

```

||||||||||||||||||||||||||||||||||||||||||||

```

void CoresAleatorias(int PosicaoNaRoda, int* RGB) {
    RodaDeCores(PosicaoNaRoda, RGB);
    analogWrite(REDPIN, map(RGB[0], 0, 255, 0, 1023));
    analogWrite(GREENPIN, map(RGB[1], 0, 255, 0, 1023));
    analogWrite(BLUEPIN, map(RGB[2], 0, 255, 0, 1023));
}

```

||||||||||||||||||||||||||||||||||||||||||||

```

void RodaDeCores(int PosicaoNaRoda, int* RGB) {
    PosicaoNaRoda = PosicaoNaRoda % 256;

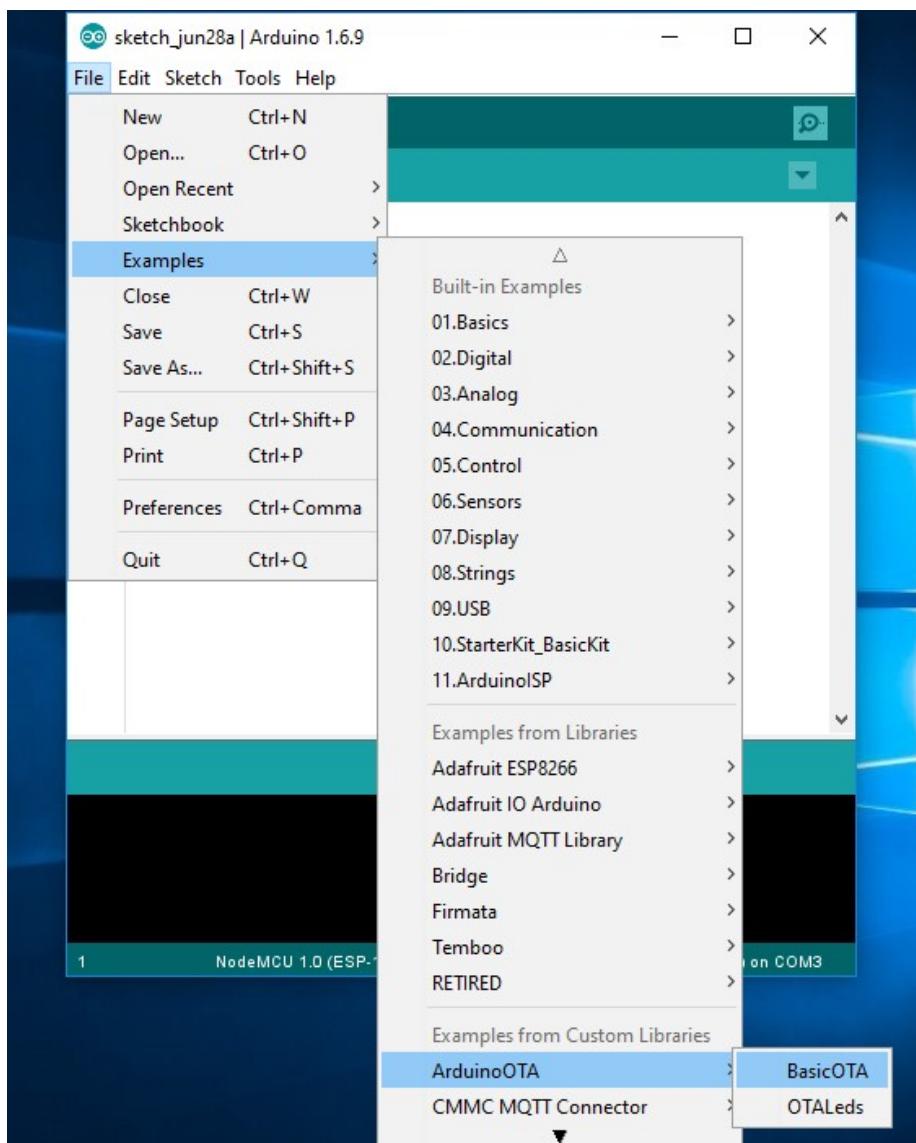
    if (PosicaoNaRoda < 85) {
        RGB[0] = PosicaoNaRoda * 3;
        RGB[1] = 255 - PosicaoNaRoda * 3;
        RGB[2] = 0;
    }
}

```

```
else if (PosicaoNaRoda < 170) {  
    PosicaoNaRoda -= 85;  
    RGB[2] = PosicaoNaRoda * 3;  
    RGB[0] = 255 - PosicaoNaRoda * 3;  
    RGB[1] = 0;  
}  
else if (PosicaoNaRoda < 255) {  
    PosicaoNaRoda -= 170;  
    RGB[1] = PosicaoNaRoda * 3;  
    RGB[2] = 255 - PosicaoNaRoda * 3;  
    RGB[0] = 0;  
}  
else  
{  
    PosicaoNaRoda -= 255;  
    RGB[0] = PosicaoNaRoda * 3;  
    RGB[1] = 255 - PosicaoNaRoda * 3;  
    RGB[2] = 0;  
}
```

ATUALIZAÇÕES DE CÓDIGO OTA (OVER-THE-AIR) NA IDE ARDUINO

1. Abra o Arduino IDE e selecione BasicOTA sketch em Arquivo> Exemplos> ArduinoOTA> BasicOTA



```
#include <ESP8266WiFi.h>

#include <ESP8266mDNS.h>

#include <WiFiUdp.h>

#include <ArduinoOTA.h>

const char* ssid = ".....";

const char* password = ".....";

void setup() {

  Serial.begin(115200);
```

```

Serial.println("Booting");
WiFi.mode(WIFI_STA);
WiFi.begin(ssid, password);
while (WiFi.waitForConnectResult() != WL_CONNECTED) {
    Serial.println("Connection Failed! Rebooting...");
    delay(5000);
    ESP.restart();
}

// Port defaults to 8266
// ArduinoOTA.setPort(8266);

// Hostname defaults to esp8266-[ChipID]
// ArduinoOTA.setHostname("myesp8266");

// No authentication by default
// ArduinoOTA.setPassword((const char *)"123");

ArduinoOTA.onStart([]) {
    Serial.println("Start");
};

ArduinoOTA.onEnd([]) {
    Serial.println("\nEnd");
};

ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
});

ArduinoOTA.onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
    else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
    else if (error == OTA_END_ERROR) Serial.println("End Failed");
});

ArduinoOTA.begin();
Serial.println("Ready");
Serial.print("IPess: ");
Serial.println(WiFi.localIP());
}

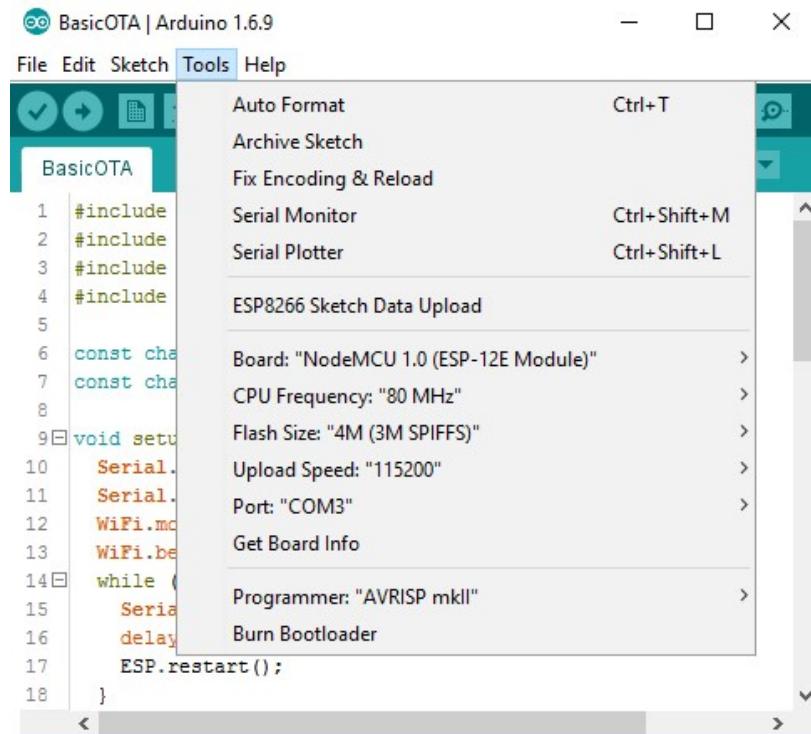
void loop() {
    ArduinoOTA.handle();
}

```

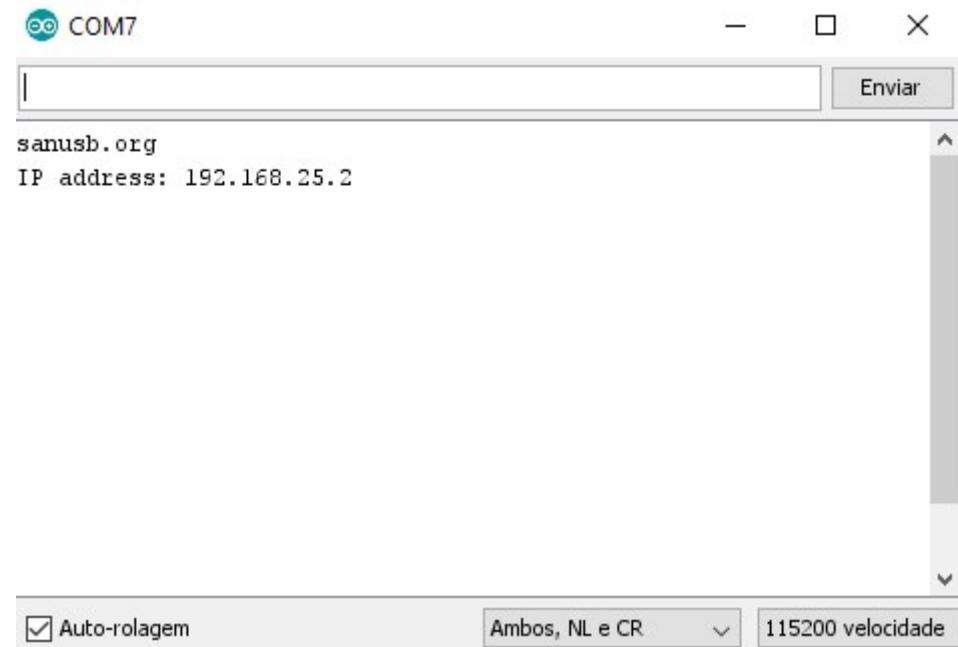
Edite o código e adicione suas credenciais WiFi.

```
Const char * ssid = "Your_WiFi_SSID";
Const char * password = "Your_WiFi_Password"
```

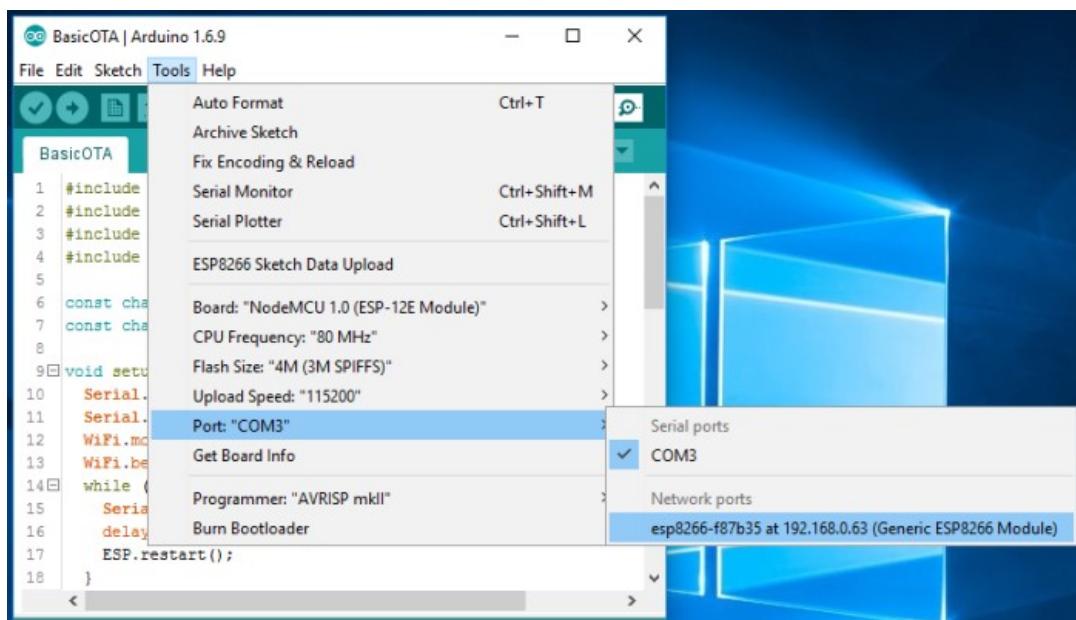
Na IDE do Arduino, vá para as ferramentas e configure os parâmetros de upload serial do seu módulo e faça o upload do código.



Em seguida, abra o monitor serial. Se o módulo estiver conectado à rede, você verá algo como isto.



Depois de algum tempo em Ferramentas> Ports> Portas de rede uma nova porta OTA aparecerá.



Nota: Se a porta OTA não aparecer, reinicie o IDE do Arduino, se isso não ajudar a verificar as configurações do firewall.

Agora, seu dispositivo está pronto para atualização OTA. Agora você pode enviar qualquer código para o seu dispositivo remotamente. Abra a IDE Arduino e selecione a porta OTA e desfrute da atualização OTA. Nota: Para fazer upload de seu esboço repetidamente usando OTA, é necessário incorporar rotinas OTA (em negrito) dentro do código. Use o BasicOTA.ino abaixo como exemplo com LED intermitente e rotinas OTA

```
#include <ESP8266WiFi.h>
#include <ESP8266mDNS.h>
#include <WiFiUdp.h>
#include <ArduinoOTA.h>

const char* ssid = "SETA-AF82";
const char* password = "14502384";

void setup() {
    Serial.begin(115200);
    pinMode(13, OUTPUT);
    Serial.println("Booting");
    WiFi.mode(WIFI_STA);
    WiFi.begin(ssid, password);
    while (WiFi.waitForConnectResult() != WL_CONNECTED) {
        Serial.println("Connection Failed! Rebooting...");
        delay(5000);
        ESP.restart();
    }
}

// Port defaults to 8266
// ArduinoOTA.setPort(8266);
```

```
// Hostname defaults to esp8266-[ChipID]
ArduinoOTA.setHostname("esp-sanusb");

// No authentication by default
ArduinoOTA.setPassword((const char *)"123"); //senha para gravação

ArduinoOTA.onStart([]) {
    Serial.println("Start");
};

ArduinoOTA.onEnd([]) {
    Serial.println("\nEnd");
};

ArduinoOTA.onProgress([](unsigned int progress, unsigned int total) {
    Serial.printf("Progress: %u%%\r", (progress / (total / 100)));
});

ArduinoOTA.onError([](ota_error_t error) {
    Serial.printf("Error[%u]: ", error);
    if (error == OTA_AUTH_ERROR) Serial.println("Auth Failed");
    else if (error == OTA_BEGIN_ERROR) Serial.println("Begin Failed");
    else if (error == OTA_CONNECT_ERROR) Serial.println("Connect Failed");
    else if (error == OTA_RECEIVE_ERROR) Serial.println("Receive Failed");
    else if (error == OTA_END_ERROR) Serial.println("End Failed");
});

ArduinoOTA.begin();
Serial.println("sanusb.org");
Serial.print("IP address: ");
Serial.println(WiFi.localIP());
}

void loop() {
    ArduinoOTA.handle();
    digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
    delay(100); // wait for 2 second
    digitalWrite(13, LOW); // turn the LED off by making the voltage LOW
    delay(100);
}
```

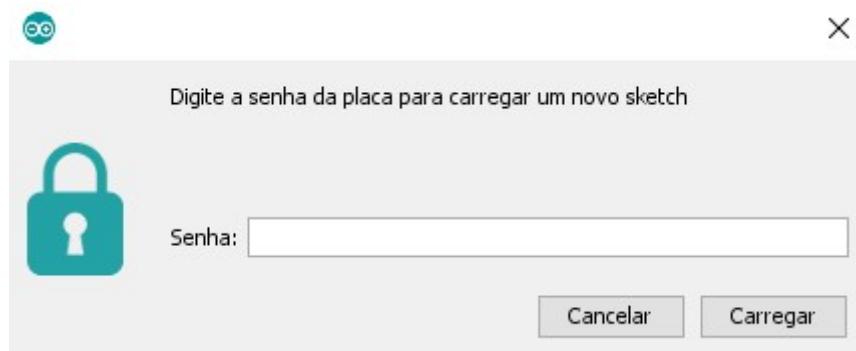
Se aparecer o erro durante a gravação *Cannot run program "python.exe"*, baixe em <https://www.python.org/downloads/> , instale o python selecionando a opção *Add python.exe.to Path* e reinicie.

- o Python 2.7 (do not install Python 3.5 that is not supported) - <https://www.python.org/>

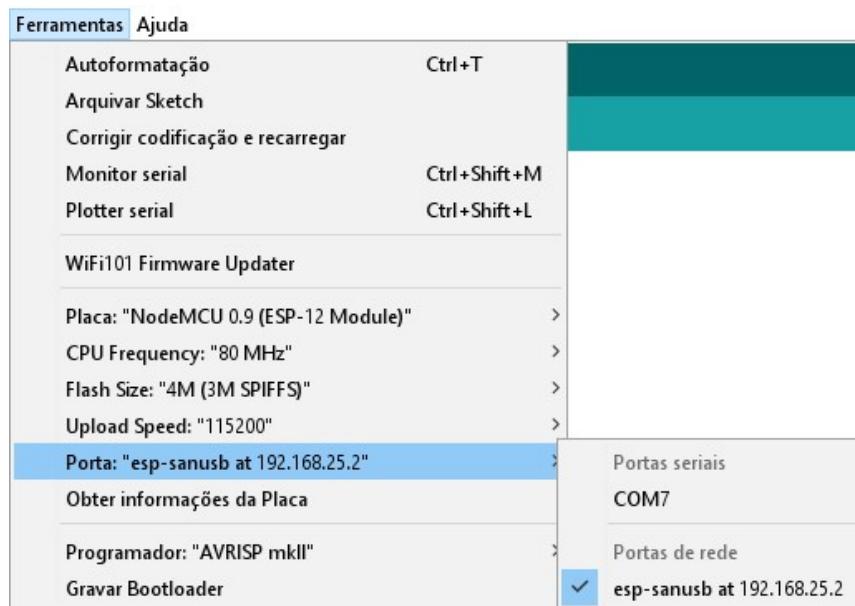
Note: Windows users should select "Add python.exe to Path" (see below – this option is not selected by default).



Ao selecionar a porta para gravação esp-sanusb em ferramentas, e mandar gravar, caso esteja descrito no código a função `ArduinoOTA.setPassword((const char *)"123");` aparecerá, antes da gravação final, uma tela como baixo de solicitação de senha para continuar, que nesse caso será 123.



Para depurar o programa após a gravação não esqueça de ir em ferramentas e comutar a porta de gravação esp-sanusb para gravação para a COM serial disponível.



COMPILADOR ESP OPEN RTOS

Para utilizar o compilador da Espressif com implementação a RTOS, basta seguir os seguintes comandos de instalação em uma máquina Linux:

```
sudo apt-get install make unrar-free autoconf automake libtool gcc g++ gperf \
flex bison texinfo gawk ncurses-dev libexpat-dev python-dev python python-serial \
sed git unzip bash help2man wget bzip2 libtool-bin
```

```
git clone --recursive https://github.com/pfalcon/esp-open-sdk.git
```

```
cd esp-open-sdk
```

```
make
```

```
cd ..
```

```
export PATH=~/esp-open-sdk/xtensa-lx106-elf/bin:$PATH
```

```
git clone --recursive https://github.com/SuperHouse/esp-open-rtos.git
```

```
cd esp-open-rtos
```

Depois de instalado, é possível compilar e gravar um firmware como, por exemplo, o *simple* da pasta *examples*, bastando para isso digitar os seguintes comandos no terminal.

```
~/esp-open-rtos$ export PATH=/home/laese/esp-open-sdk/xtensa-lx106-elf/bin:$PATH
```

```
~/esp-open-rtos$ make flash -j3 -C examples/simple ESPPORT=/dev/ttyUSB0
```

Abaixo um código para emular um semáforo com contagem para passagem de pedestre através de um display de sete segmentos.

```
/* Desligar o display de 7 segmentos antes de gravar
   pois o mesmo usa as portas de gravação
*/
#include "espressif/esp_common.h"
#include "esp/uart.h"
#include "FreeRTOS.h"
#include "task.h"
#include "queue.h"
#include "esp8266.h"
static const uint8_t TabelaBit [] =
{
    //Segments
    // a b c d e f g
    // 1 3 15 13 12 14 2  ->No. pino editável BCM ou wiringPi(wPi)
    1, 1, 1, 1, 1, 0, // Plota 0
    0, 1, 1, 0, 0, 0, // Plota 1
    1, 1, 0, 1, 1, 0, 1, // Plota 2
    1, 1, 1, 1, 0, 0, 1, // Plota 3
    0, 1, 1, 0, 0, 1, 1, // Plota 4
    1, 0, 1, 1, 0, 1, 1, // Plota 5
    1, 0, 1, 1, 1, 1, 1, // Plota 6
    1, 1, 1, 0, 0, 0, 0, // Plota 7
    1, 1, 1, 1, 1, 1, 1, // Plota 8
    1, 1, 1, 1, 0, 1, 1, // Plota 9
    1, 1, 1, 0, 1, 1, 1, // Plota A
    0, 0, 1, 1, 1, 1, 1, // Plota b
    1, 0, 0, 1, 1, 1, 0, // Plota C
    0, 1, 1, 1, 1, 0, 1, // Plota d
    1, 0, 0, 1, 1, 1, 1, // Plota E
    1, 0, 0, 0, 1, 1, 1, // Plota F
    0, 0, 0, 0, 0, 0, 0, // Plota blank
};

//pinos do display
const int pinA = 1;
const int pinB = 3;
const int pinC = 15;
const int pinD = 13;
const int pinE = 12;
const int pinF = 14;
const int pinG = 2;
```

```

//pinos dos farois
const int verde = 4;
const int amarelo = 5;
const int vermelho = 16;

int flag = 0;

/* pin config */
const int gpio = 0; /* gpio 0 usually has "PROGRAM" button attached */
const int active = 0; /* active == 0 for active low */
const gpio_inttype_t int_type = GPIO_INTTYPE_EDGE_NEG;

void reset7() {
    gpio_write(pinA, 1);
    gpio_write(pinB, 1);
    gpio_write(pinC, 1);
    gpio_write(pinD, 1);
    gpio_write(pinE, 1);
    gpio_write(pinF, 1);
    gpio_write(pinG, 1);
    //gpio_write(pinDP, 1);
}

void set7 (int num) {
{
    uint8_t seg, Valor ;
    static const uint8_t pinoBCM [7] ={1,3,15,13,12,14,2}; //Simples de atualizar os pinos ligados aos display de 7
    segmentos

    for (seg = 0 ; seg < 7 ; ++seg)
    {
        Valor = TabelaBit [num * 7 + seg] ; //num * 7 indica a linha e seg indica a coluna
        gpio_write(pinoBCM[seg], !Valor) ; //escreve no segmento o bit 0 (LOW) ou 1 (HIGH) de acordo com a tabela
        TabelaBit
    }
}
}

void farol(void *pvParameters) {

QueueHandle_t *queue = (QueueHandle_t *)pvParameters;
while(1) {
    gpio_write(vermelho,0);
    gpio_write(verde, 1);
    vTaskDelay(1000 / portTICK_PERIOD_MS);
    gpio_write(verde, 0);
    gpio_write(amarelo, 1);
}
}

```

```

vTaskDelay(1000 / portTICK_PERIOD_MS);
gpio_write(amarelo, 0);
gpio_write(vermelho, 1);

//flag = 1; //aciona bandeira
if(flag==1){ //verifica bandeira
//vTaskDelay(1000 / portTICK_PERIOD_MS);
cronometro();
reset7();
flag=0; //libera bandeira
}

vTaskDelay(1000 / portTICK_PERIOD_MS);
//gpio_write(vermelho, 1);
//vTaskDelay(1000 / portTick_PERIOD_MS);
}

}

void cronometro() {
    int num;
    //vTaskDelay(5000 / portTICK_PERIOD_MS);
    for(num=9;num>=0;num--){
        set7(num);
        vTaskDelay(1000 / portTICK_PERIOD_MS);
    }
    printf("teste");
}

```

void gpio_intr_handler(uint8_t gpio_num);

/* This task configures the GPIO interrupt and uses it to tell
when the button is pressed.

The interrupt handler communicates the exact button press time to
the task via a queue.

This is a better example of how to wait for button input!

```

*/
void buttonIntTask(void *pvParameters) {
    printf("Waiting for button press interrupt on gpio %d...\r\n", gpio);
    QueueHandle_t *mainqueue = (QueueHandle_t *)pvParameters;
    gpio_set_interrupt(gpio, int_type, gpio_intr_handler);

    uint32_t last = 0;
    while(1) {
        uint32_t button_ts;
        xQueueReceive(*mainqueue, &button_ts, portMAX_DELAY);
    }
}

```

```
button_ts *= portTICK_PERIOD_MS;
if(last < button_ts-200) {
    printf("Button interrupt fired at %dms\r\n", button_ts);
    flag = 1; //aciona bandeira
    last = button_ts;
}
}

static QueueHandle_t mainqueue;

void gpio_intr_handler(uint8_t gpio_num){
    uint32_t now = xTaskGetTickCountFromISR();
    xQueueSendToBackFromISR(mainqueue, &now, NULL);
}

void user_init(void) {
    uart_set_baud(0, 115200);
    printf("SDK version:%s\n", sdk_system_get_sdk_version());
    mainqueue = xQueueCreate(10, sizeof(uint32_t));

    gpio_enable(1, GPIO_OUTPUT);
    gpio_write(1, 1);
    gpio_enable(2, GPIO_OUTPUT);
    gpio_write(2, 1);
    gpio_enable(3, GPIO_OUTPUT);
    gpio_write(3, 1);
    gpio_enable(12, GPIO_OUTPUT);
    gpio_write(12, 1);
    gpio_enable(13, GPIO_OUTPUT);
    gpio_write(13, 1);
    gpio_enable(14, GPIO_OUTPUT);
    gpio_write(14, 1);
    gpio_enable(15, GPIO_OUTPUT);
    gpio_write(15, 1);

    gpio_enable(verde, GPIO_OUTPUT);
    gpio_write(verde, 1);
    gpio_enable(amarelo, GPIO_OUTPUT);
    gpio_write(amarelo, 0);
    gpio_enable(vermelho, GPIO_OUTPUT);
    gpio_write(vermelho, 1);
    xTaskCreate(farol, "farol", 256, &mainqueue, 2, NULL);

//xTaskCreate(cronometro, "cronometro", 256, &mainqueue, 2, NULL);
```

```

gpio_enable(gpio, GPIO_INPUT);
xTaskCreate(buttonIntTask, "buttonIntTask", 256, &mainqueue, 2, NULL);
}

```

PlatformIO

Platformio é um software livre multiplataforma para compilação que funciona até no Raspberry Pi (<http://docs.platformio.org/en/latest/core.html>) e pode ser utilizado para compilar mais de 200 placas diferentes de microcontroladores: <http://platformio.org/boards>. Uma vantagem dessa plataforma é que cada projeto salva as características da placa em que o projeto foi desenvolvido, não necessitando reconfigurar a IDE caso a placa seja outra, como é o caso da IDE Arduino, por exemplo no caso de uma mudança de ESP8266 para ESP32.

Instalando a IDE platformIO

Para instalar o platformio por execução de linha de comando CLI, basta digitar o seguinte comando no terminal ou prompt de comando (Mac/Linux/Windows), após ter instalado o python 2.7 (<https://www.python.org/downloads/>) ou superior adicionando o *Python.exe to Path*:

pip install -U platformio

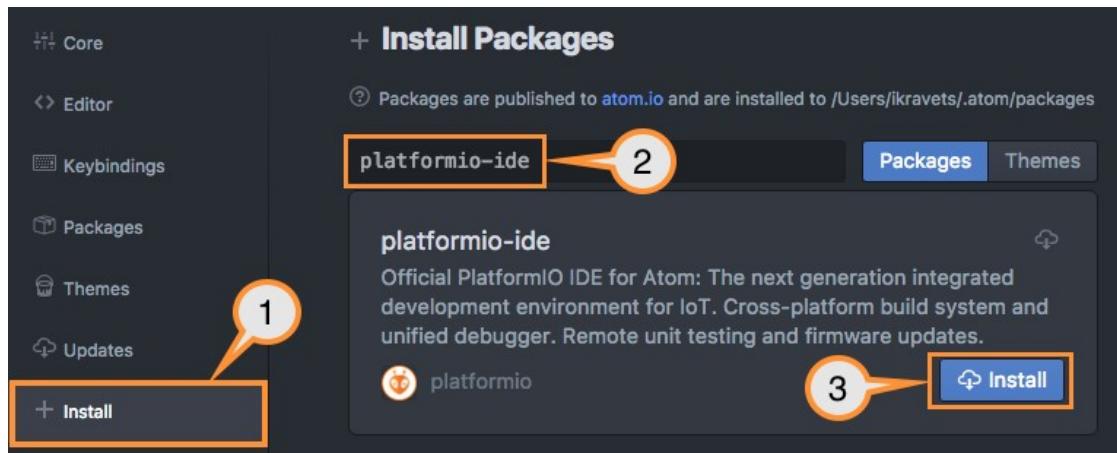


Outras formas de instalação em: <http://docs.platformio.org/en/latest/installation.html>

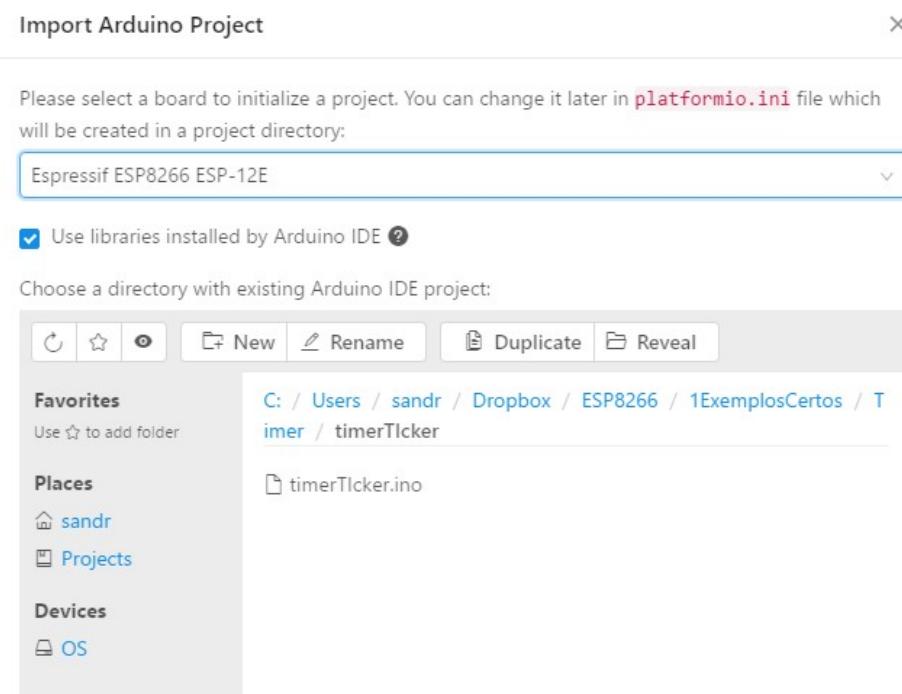
Agora baixe e instale a IDE PlatformIO Atom para o sistema operacional desejado em <http://platformio.org/platformio-ide>.

1. Após baixar a IDE Atom, acesse a aba Install:
 - o Windows, Menu: File > Settings > Install

- o macOS, Menu: Atom > Preferences > Install
 - o Linux, Menu: Edit > Preferences > Install
2. Procure o pacote oficial platformio-ide
 3. Instale a IDE PlatformIO .



A IDE Atom foi criada pela comunidade do GITHUB para edição de códigos. É importante salientar que não é necessário instalar o PlatformIO Core se for instalado a IDE PlatformIO Atom, pois o Core já é integrado ao IDE e é possível acessá-lo no Terminal do PlatformIO. Para realizar a migração de um firmware da IDE Arduino para essa plataforma mais veloz, selecione em *platformIO home: Import Arduino Project* e configure, como a Figura abaixo, o projeto a ser importado e clique em *Import*.



Essa plataforma compila também a linguagem *Wiring* de programação do Arduino, mas a linguagem de compilação padrão do platformIO é similar a wiring e mais rápida é a c++ (cpp). Ao importar um projeto baseado na IDE do Arduino, o Platformio cria uma

subpasta, por exemplo em *C:\Users\[NOME]\Documents\PlatformIO\Projects* e aloca todos os projetos gerados para essa IDE.

Após abrir o firmware clique em *Platformio -> Build*. Na primeira compilação irá baixar as bibliotecas a toolchain-xtensa da Espressif.

```

180202-123911-esp12e
lib
src
  timerTicker.ino
.gitignore
.travis.yml
platformio.ini

#include <Ticker.h>
//Biblioteca esp8266 que chama funções de interrupção
Ticker tickerNivelAlto;
Ticker tickerNivelBaixo;
void setPin(int state) {
    digitalWrite(13, state);
    Serial.print("Interrupcao ticker: ");
    Serial.println(state);
}
void setup() {
    Serial.begin(115200);
    pinMode(13, OUTPUT);
    digitalWrite(13, LOW);
}
// a cada 500ms, chama setPin(0)

platformio run
[02/02/18 12:40:09] Processing esp12e (platform: espressif8266; frameworks)
PackageManager: Installing toolchain-xtensa @ ~1.40802.0
Downloading...

```

A primeira compilação geralmente tem um tempo maior porque são gerados os objetos (.o) que serão utilizados para as próximas compilações. É possível abrir vários projetos em paralelo e para direcionar para compilação, basta clicar em um arquivo do projeto que era será selecionado com uma indicação de barra vertical azul. Note que o arquivo .ino dentro do projeto platformIO, ainda pode ser compilado pela IDE do Arduino, pois durante a compilação do platformIO, é gerado um arquivo temporário .cpp da conversão do .ino, que é compilado e depois de criado o binário, esse arquivo .cpp é apagado. Para verificar essa operação, basta acessar a pasta do projeto durante a compilação (*Build*).

Essa conversão de .ino para .cpp é basicamente a inserção da biblioteca (*#include <Arduino.h>*) e a prototipação *void setup();* e *void loop();* no código. Dessa forma, se o arquivo .ino for substituído pelo arquivo .cpp permanentemente no projeto, o tempo de compilação será ainda menor, pois não será mais necessário essa etapa para o PlatformIO gerar o arquivo binário localizado em *.pioenvs\esp12e\firmware.bin*. Esse

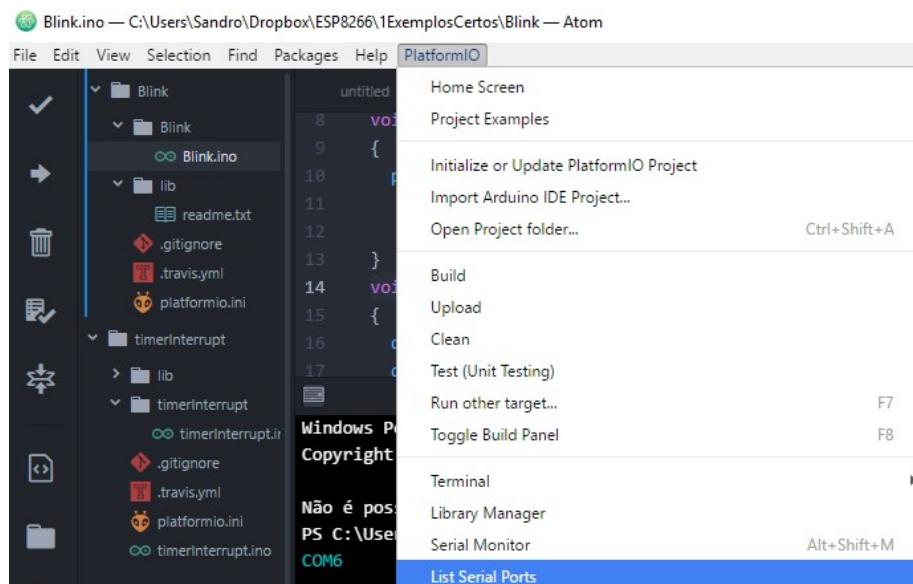
firmware.bin pode ser utilizado para gravar o ESP8266 via WiFi pelo método OTA (*over the air*).

```

Blink.ino — C:\Users\Sandro\Dropbox\ESP8266\1ExemplosCertos\Blink — Atom
File Edit View Selection Find Packages Help PlatformIO
Blink
  └── Blink
    ├── Blink.ino
    └── Blink.ino.cpp
  └── lib
    └── readme.txt
  └── timerInterrupt
    ├── lib
    └── timerInterrupt
      └── timerInterrupt.ino
  └── OnOffEspClientGet
    ├── lib
    └── OnOffEspClientGet
      └── OnOffEspClientGet.ino
  └── .gitignore
  └── .travis.yml
  └── platformio.ini
  └── timerInterrupt.ino
untitled
timerInterrupt.ino
Telemetry Consent
Welcome
OnOffEspClient
tim
1 // ****
2 Blink
3 Connected to pin D7 (GPIO13) ESP8266 NODEMCU
4 ****
5 #define ledPin 13 // #define LedPin D7
6 void setup()
7 {
8   pinMode(ledPin, OUTPUT);
9   pinMode(12, OUTPUT);
10  Serial.begin(115200);
11 }
12 void loop()
13 {
14   digitalWrite(ledPin, HIGH); digitalWrite(12, 0);
platformio run --target upload
[02/11/17 01:09:20] Processing esp12e (platform: espressif8266, board: esp12e, framework: arduino)
Verbose mode can be enabled via '-v, --verbose' option
Converting Blink.ino
Collected 28 compatible libraries
Looking for dependencies...
Project does not have dependencies
Compiling .pioenvs\esp12e\src\Blink.ino.o
Looking for upload port...
Auto-detected: COM6

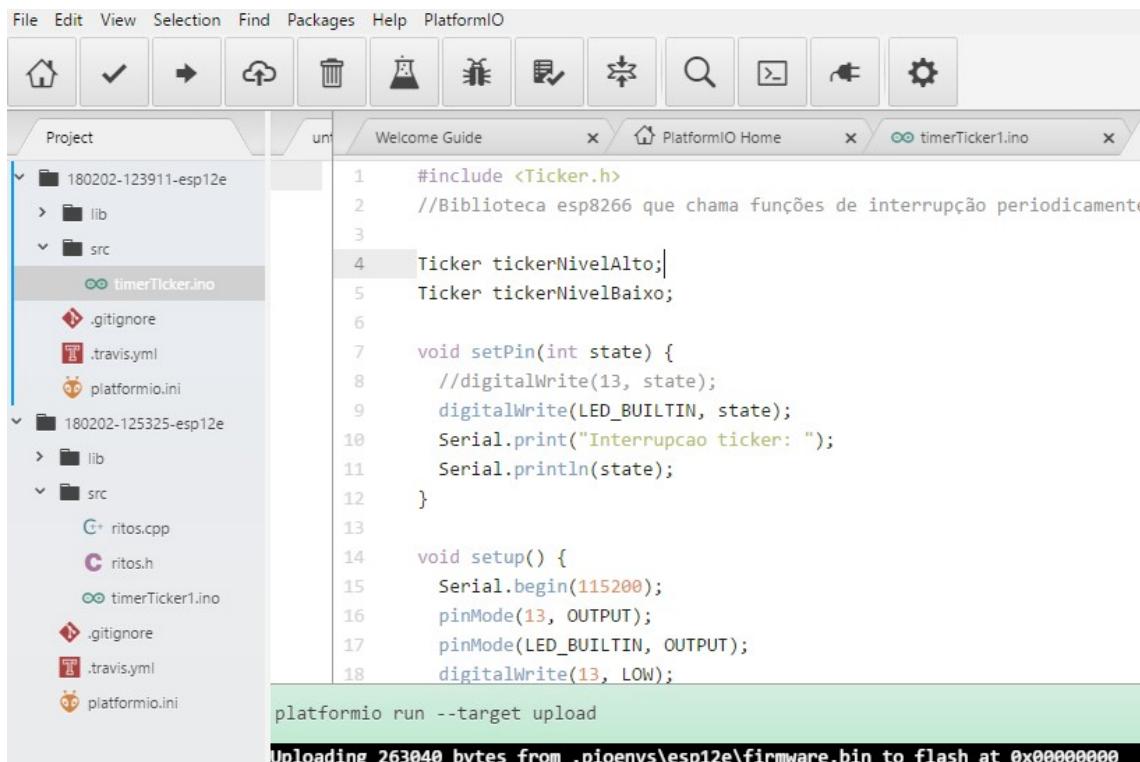
```

Para depurar pela serial, é possível ver a serial disponível instalada em *List Serial Ports*. Depois de verificar basta clicar em serial Monitor.

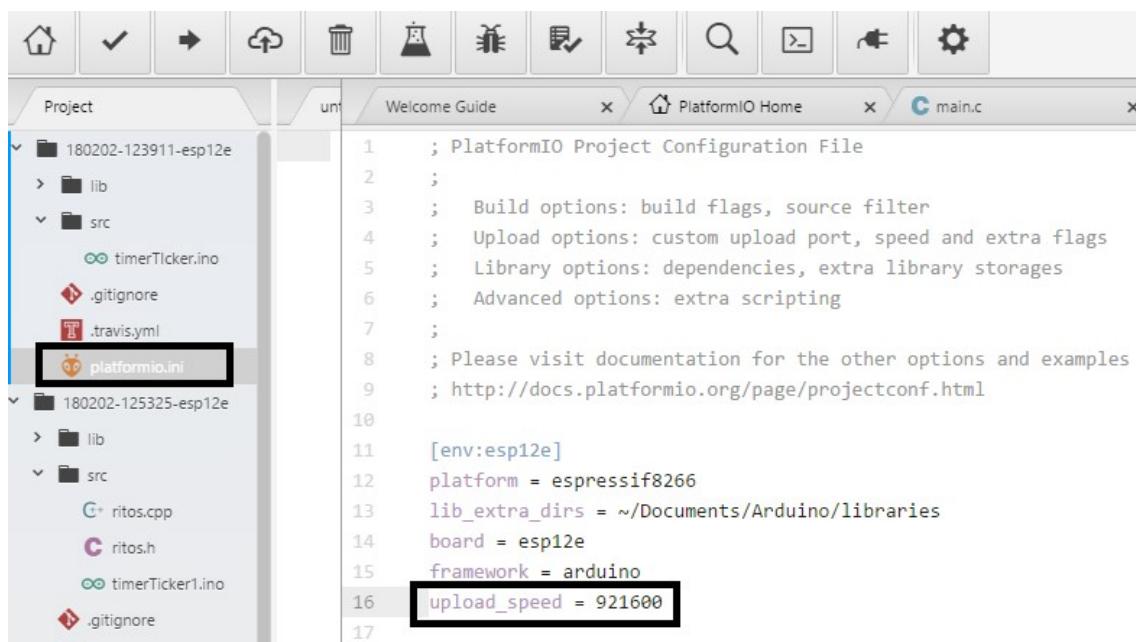


Se quiser salvar automaticamente todos os arquivos após clicar em build, basta ir em *build*: *PlatformIO > Settings > Build* e selecione: *Automatically save on build*. Esse ambiente gera o binário em **.pioenvs\esp12e\firmware.bin** que pode ser utilizado para gravação *Over the Air* através da biblioteca WebUpdater.

Para modificar a cor de fundo, basta clicar em *platformio-> settings _> Themes* e selecionar a cord a IDE (UI Theme) e a cor da sintaxe de edição (Syntax Theme).

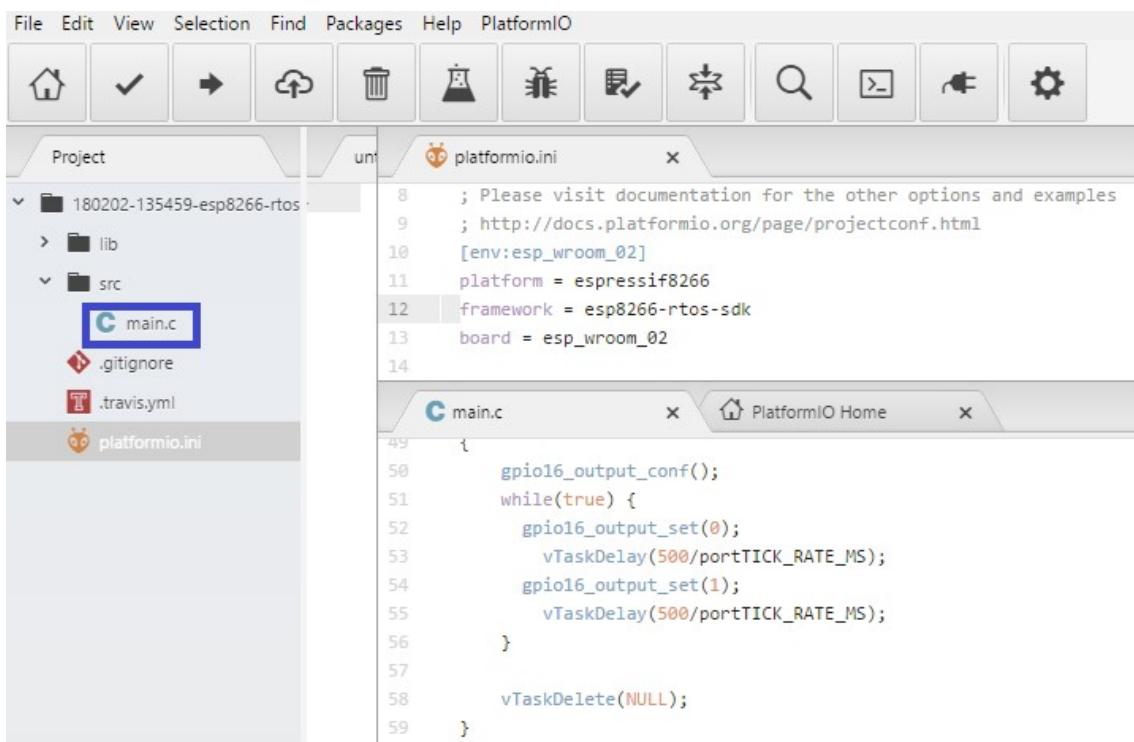


É possível alterar a taxa de transferência de gravação do ESP (pupload), acessando o arquivo *platformio.ini* e digitando *upload_speed = 921600*, como na Figura abaixo:



Caso aconteça falha na gravação, é recomendável deixar a taxa de transferência em 115200.

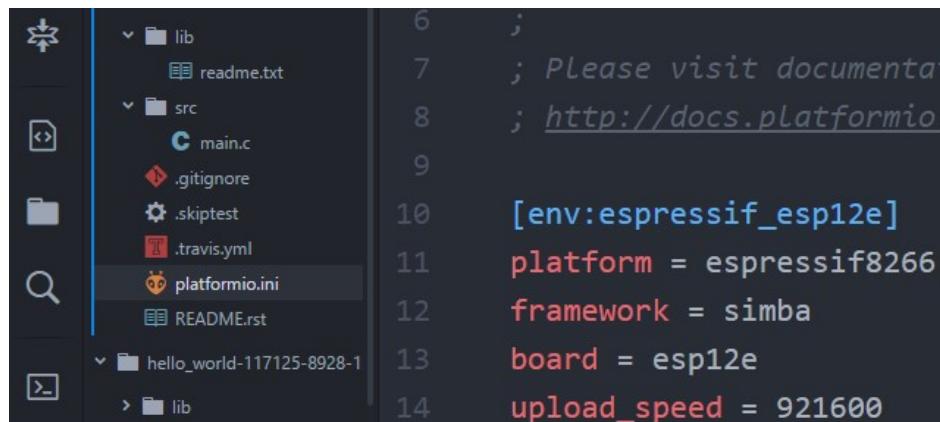
É possível também compilar projetos RTOS FreeRTOS, baixando o exemplo em Platformio Home -> Project Examples e selecionando *esp-rtos-sdk-blink*.



COMPILEADOR SIMBA NA IDE PLATFORMIO

Caso se deseje utilizar compilador Simba, basta importar um exemplo na Aba PlatformIO home -> Project Examples -> Simba\blink.

Nesse caso, no arquivo platformio.ini coloque somente a placa para qual deseja compilar e insira no final a taxa de gravação serial maxima upload_speed = 921600.



Teste também o exemplo do firmware abaixo:

```
#include "simba.h"

int main()
{
    struct pin_driver_t led;
    struct uart_driver_t uart;

    /* Start the system. */
    sys_start();

    uart_module_init();
    uart_init(&uart, &uart_device[0], 115200, NULL, 0);
    uart_start(&uart);
    sys_set_stdout(&uart.chout);

    /* Initialize the LED pin as output and set its value to 1. */
    pin_init(&led, &pin_led_dev, PIN_OUTPUT);
    pin_write(&led, 1);

    while (1) {
        /* Wait half a second. */
        thrd_sleep_ms(2000);

        std_printf(FSTR("pin toggle!\n"));
        /* Toggle the LED on/off. */
        pin_toggle(&led);
    }

    return (0);
}
```

MEMÓRIA NÃO VOLÁTIL

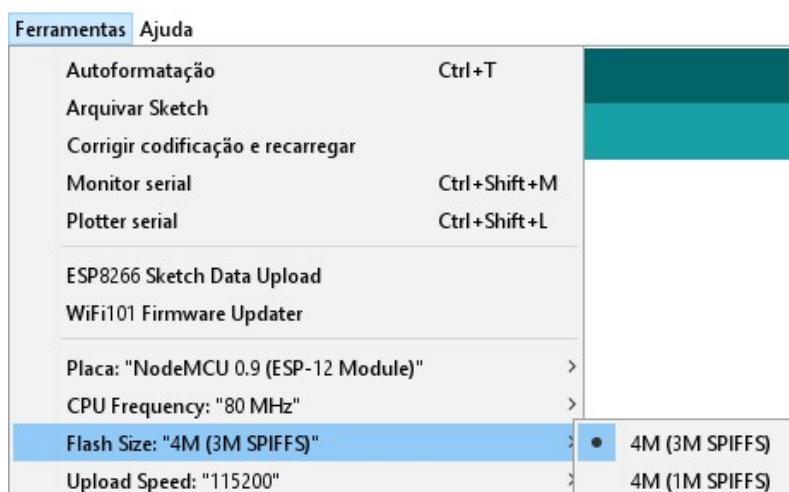
O ESP8266, não possui memoria não volátil interna, sendo necessário incluí-la externamente no módulo. No caso dos módulos do ESP8266, essa memoria é uma memoria flash que pode variar de 512kB até 4MB, que é o caso dos módulos NodeMCU e utiliza um barramento QSPI (*queued serial peripheral interface*).

Abaixo uma tabela mostra os diferentes tamanhos de memoria nos módulos do ESP.

Dá para utilizar a memoria flash do ESP para o armazenamento de informações do seu sistema, além das informações que já são armazenadas, como o código fonte da aplicação e todos os recursos como bibliotecas. Para armazenar as suas informações na memoria do ESP, existem duas formas:

SPIFFS (SPI Flash File System)

O SPIFFS é um tipo de sistema de arquivo, que possibilita a criação de arquivos e diretórios na memoria flash. Pode ser utilizado para sistema de armazenamento de dados mais complexos, como imagens e bases de dados. Essa memória dispõe de até 3M livre da memória flash.



No link <https://www.youtube.com/watch?v=wLPmOv9h988> é mostrado um tutorial de instalação e uso do *SPI Flash File System* na memória em um ESP8266. O firmware deste exemplo está abaixo.

```
//Video: https://www.youtube.com/watch?v=wLPmOv9h988
#include "FS.h"
```

```
void prepareFile(){

    Serial.println("Prepare file system");
    SPIFFS.begin(); //monta o filesystem

    File arquivo = SPIFFS.open("/test.txt", "r");
    if (!arquivo) {
        Serial.println("File doesn't exist yet. Creating it.");
    } else{
```

```

Serial.println("file open success:");

while (arquivo.available()) {
    Serial.write(arquivo.read());
}

arquivo.close();
}

void setup() {
    Serial.begin(115200);
    Serial.println("To read separate file in file system");
    prepareFile();
}

void loop() {
}

```

Para criar, ler e escrever um arquivo, é importante conhecer os parâmetros listados:

r Abre o arquivo de texto para leitura. O fluxo é posicionado no início do arquivo.

r+ Abre para leitura e escrita. O fluxo é posicionado no início do arquivo.

w Trunca o arquivo para comprimento zero ou cria um arquivo de texto para gravação. O fluxo é posicionado no início do arquivo.

w+ Abre para leitura e escrita. Se não existe, o arquivo é criado. Caso contrário ele é truncado. O fluxo é posicionado no início do arquivo.

a Abre para anexar (gravação no final do arquivo). O arquivo é criado se ele não existir. O fluxo é posicionado final do arquivo.

a+ Abre para leitura e anexação (gravação no fim do arquivo). O Arquivo é criado se ele não existir. O arquivo inicial de leitura é posicionado no início do arquivo, mas a saída é sempre anexado ao final do arquivo.

O firmware abaixo possibilita escrever e ler no arquivo .txt da memória Flash file System (SPIFFS) do ESP8266. Mais detalhes no vídeo em <https://youtu.be/28pCObPRrG4>.

```
#include "FS.h" //video: https://youtu.be/28pCObPRrG4
```

```
void setup() {
```

```

Serial.begin(115200);

bool result = SPIFFS.begin(); // função para montar o filesystem

//abre para leitura e escrita no final do arquivo
File arq = SPIFFS.open("/test.txt", "a+"); //arq está na RAM
arq.print('1'); //escreve byte
arq.print("23"); //escreve string

if (!arq) {
    Serial.println("Arquivo ainda não existe.");
}
else{
    arq.close();
    delay(500);
    arq = SPIFFS.open("/test.txt", "a+");//abre o arquivo para leitura
    Serial.println("arquivo criado com sucesso.");
    while(arq.available())
    {
        //Serial.write(arq.read());
        String line = arq.readStringUntil('\n');
        Serial.println(line);
    }
    arq.close();
}
}

void loop() {
}

```

O exemplo abaixo guarda os Logs da conexão Wifi em um arquivo .txt criado automaticamente na memória SPIFFS e exibe-o-em uma página .html que é mostrada quando o IP do ESP8266 é inserido no browser.

```
#include "FS.h" //video: https://youtu.be/28pCObPRrG4
#include <ESP8266WiFi.h>
```

```
const char* ssid = "SETA-AF82";
const char* password = "14502384";
String buf;
```

```
WiFiServer server(80);
```

```
void formatFS(void){
    SPIFFS.format();
```

```
}

void createFile(void){
    File arq;

    //Cria o arquivo se ele não existir
    if(SPIFFS.exists("/log.txt")){
        Serial.println("Arquivo ja existe!");
    } else {
        Serial.println("Criando o arquivo...");
        arq = SPIFFS.open("/log.txt","w+");

        //Verifica a criação do arquivo
        if(!arq){
            Serial.println("Erro ao criar arquivo!");
        } else {
            Serial.println("Arquivo criado com sucesso!");
        }
    }
    arq.close();
}

void deleteFile(void) {
    //Remove o arquivo
    if(SPIFFS.remove("/log.txt")){
        Serial.println("Erro ao remover arquivo!");
    } else {
        Serial.println("Arquivo removido com sucesso!");
    }
}

void writeFile(String msg) {

    //Abre o arquivo para adição (append)
    //Inclue sempre a escrita na ultima linha do arquivo
    File rFile = SPIFFS.open("/log.txt","a+");

    if(!rFile){
        Serial.println("Erro ao abrir arquivo!");
    } else {
        rFile.println("Log: " + msg);
        Serial.println(msg);
    }
    rFile.close();
}
```

```
void readFile(void) {  
    //Faz a leitura do arquivo  
    File rFile = SPIFFS.open("/log.txt","r");  
    Serial.println("Reading file...");  
    while(rFile.available()) {  
        String line = rFile.readStringUntil("\n");  
        buf += line;  
        buf += "<br />";  
    }  
    rFile.close();  
}  
  
void closeFS(void){  
    SPIFFS.end();  
}  
  
void openFS(void){  
    //Abre o sistema de arquivos  
    if(!SPIFFS.begin()){  
        Serial.println("Erro ao abrir o sistema de arquivos");  
    } else {  
        Serial.println("Sistema de arquivos aberto com sucesso!");  
    }  
}  
  
void setup(void){  
    //Configura a porta serial para 115200bps  
    Serial.begin(115200);  
  
    //Abre o sistema de arquivos (mount)  
    openFS();  
    //Cria o arquivo caso o mesmo não exista  
    createFile();  
  
    writeFile("Booting ESP8266...");  
    writeFile("Connecting to " + (String)ssid);  
  
    //Inicia a conexão WiFi  
    WiFi.begin(ssid, password);  
  
    while (WiFi.status() != WL_CONNECTED) {  
        delay(500);  
        Serial.print(".");  
    }  
  
    writeFile("WiFi connected");
```

```
Serial.println(WiFi.localIP());\n\n//Inicia o webserver\nserver.begin();\nwriteFile("Web Server started");\n}\n\nvoid loop(void){\n\n    //Tratamento das requisições http\n    WiFiClient client = server.available();\n    if (!client) {\n        return;\n    }\n\n    Serial.println("new client");\n\n    while(!client.available()){\n        delay(1);\n    }\n\n    String req = client.readStringUntil('\r');\n    Serial.println(req);\n    client.flush();\n\n    buf = "";\n\n    buf += "HTTP/1.1 200 OK\r\nContent-Type: text/html\r\n\r\n<!DOCTYPE HTML>\r\n<html>\r\n<h3 style=\"text-align: center;\">ESP8266 Web Log</h3>";\n    buf += "<p>";\n    readFile();\n    buf += "</p>";\n    buf += "<h4>Exemplo de Registro de Logs em SPIFFS</h4>";\n    buf += "</html>\r\n";\n\n    client.print(buf);\n    client.flush();\n\n    Serial.println("Client disconnected");\n}
```

EEPROM (emulação)

O ESP não tem EEPROM, mas sim uma emulação na memória flash. Isso significa que a vida útil de escritas na memória será a mesma da *flash*, cerca de 10.000 ciclos, e não de uma EEPROM que pode passar de 1 Milhão de ciclos de escrita (no mesmo endereço).

O tamanho desta zona que é escrita de byte a byte de memória pode ser de 4 à 4096 bytes (não é possível alocar menos de 4 bytes), o suficiente para a maior parte das aplicações. Caso precise de mais memória, sugiro utilizar o SPIFFS, que pode chegar a 3MB dependendo do modelo de ESP.

O endereço inicial da EEPROM emulada na flash é o **0x7b000**.

Como utilizar

Para utilizar a EEPROM emulada no ambiente do Arduino, devemos incluir a biblioteca EEPROM (`#include <EEPROM.h>`) na sketch.

O primeiro passo para a utilização é definir o tamanho da memória a ser utilizada com a função **EEPROM.begin(size);**, que poderá ser definida dentro do `setup()`.

Onde o tamanho (size) pode ser de 4 a 4096 bytes. Sem esta definição, não será possível utilizar a memória.

EEPROM.write(address, value);

Onde o primeiro argumento é o endereço (address) e o segundo o valor (value) que é um `unsigned8`. Para salvar as informações, devemos executar o comando:

EEPROM.commit();

E para liberar o recurso:

EEPROM.end();

O comando **EEPROM.end** já faz o comando `commit`, fazendo o uso do **EEPROM.commit** desnecessário quando utilizando o `.end`.

Leitura:

EEPROM.read(address);

A leitura tem apenas um argumento que é o endereço que será lido, o retorno é um byte `unsigned8`.

Exemplo:

Abaixo um exemplo de como fazer um incremento na posição de memória EEPROM de 0 a 512 e guardar esse número do incremento da memória dentro da própria memória. É possível também guardar o valor do conversor AD dividido por quatro para caber dentro de uma cada byte da posição de memória.

```
#include <EEPROM.h>

int addr = 0;

void setup()
{
    EEPROM.begin(512); // Aloca quantidade de bytes para EEPROM - máx.4096
}

void loop()
{
    // need to divide by 4 because analog inputs range from
    // 0 to 1023 and each byte of the EEPROM can only hold a
    // value from 0 to 255.
    int val = analogRead(A0) / 4;

    //EEPROM.write(addr, val); //Guarda o valor do conversor analogRead(A0) / 4;
    //Divide por 4 porque o valor do AD é de 0 a 1023 e o byte de memória cabe 255.
    EEPROM.write(addr, val); //guarda o número da posição de memória

    // Grava no endereço de 0 a 512.
    addr = addr + 1;
    if (addr == 512)
    {
        addr = 0;
        EEPROM.commit(); // o EEPROM.end(); possui o EEPROM.commit()
    }

    delay(100);
}
```

Abaixo a o *printscreen* da resposta serial de leitura com código abaixo do exemplo. E tocar com um fio ligado no A0 no Gnd e Vcc da placa NodeMCU, será percebido a variação do conversor AD armazenada na EEPROM.

```
#include <EEPROM.h>

// start reading from the first byte (address 0) of the EEPROM
int addr = 0, eeaddr=0;
byte value;
int val=0, eeval=0;
```

```
void setup()
{
    // initialize serial and wait for port to open:
    Serial.begin(115200);
    while (!Serial) {
        ; // wait for serial port to connect. Needed for Leonardo only
    }
    EEPROM.begin(512);
}

void loop()
{
    val = analogRead(A0) / 4;

    EEPROM.write(addr, addr); //guarda o número da posição de memória

    EEPROM.write(addr+1, val); //Guarda o valor do conversor analogRead(A0) / 4;
    //Divide por 4 porque o valor do AD é de 0 a 1023 e o byte de memória cabe 255.

    EEPROM.commit(); //Grava

    // read a byte from the current address of the EEPROM
    eeaddr = EEPROM.read(addr);
    Serial.print(eeaddr); Serial.print("\t"); //tab

    eeval = EEPROM.read(addr+1);
    Serial.print(eeval, DEC);
    Serial.println();

    // advance to the next address of the EEPROM
    addr = addr + 2;

    // there are only 512 bytes of EEPROM, from 0 to 511, so if we're
    // on address 512, wrap around to address 0
    if (addr == 256) addr = 0;

    delay(200);
}
```

COM6

104	104
105	105
106	106
107	107
108	108
109	109
110	110
111	111

Exemplo

Abaixo outro exemplo de como fazer um contador de inicialização do ESP, onde a cada boot é feito um incremento na memoria.

```
#include <EEPROM.h>

//Define o quanto sera alocado entre 4 e 4096bytes
#define MEM_ALOC_SIZE 8
uint8_t boot_num = 0;

void setup() {

    Serial.begin(115200);

    Serial.printf("Flash chip ID: %d\n", ESP.getFlashChipId());
    Serial.printf("Flash chip size (in bytes): %d\n", ESP.getFlashChipSize());
    Serial.printf("Flash chip speed (in Hz): %d\n", ESP.getFlashChipSpeed());

    EEPROM.begin(MEM_ALOC_SIZE);
    boot_num = EEPROM.read(0);
    boot_num++;
    EEPROM.write(0,boot_num);
    EEPROM.end();

    Serial.printf("Boot num: %d\n", boot_num);
}

void loop()
```

Salvar dados da memoria pode ser extremamente útil em diversas aplicações, como salvar o estado das GPIOs, sinalização de alarmes, tempo de uso entre outras informações simples. Já para casos mais complexos, ou que demandam de maior espaço, maior que 4098 bytes, o uso da EEPROM emulada é mais limitado, o ideal seria o uso de SPIFFS.

GET EM SERVIDOR

O programa exemplo abaixo mostra como realizar um get em um servidor para repassar valores que chegam pela serial através da interface de depuração ou qualquer outro dispositivo computacional.

```
#include <ESP8266WiFi.h>
const char* rede = "SETA-AF82"; //Cria o vetor de char rede
const char* senha = "14502384";
//const char* rede = "WIFI-8820"; //Cria o vetor de char rede
//const char* senha = "97D1158820";
const char* host = "sanusb.org";
String resposta, cod;
int incod;

WiFiClient cliente; //Cria o objeto cliente

void setup() {
    Serial.begin(115200); // inicializa a porta serial com 115200 baud
    Serial.println(rede);
    WiFi.begin(rede, senha); //conecta a Rede
    Serial.println("Aguardando conexão com o roteador...");
    while (WiFi.status() != WL_CONNECTED)//imprime na serial "-" enquanto o modulo não se conectar a rede
    {
        delay(500);
        Serial.print("-");
    }
    Serial.print("\nIP: ");
    Serial.println(WiFi.localIP()); //Imprime o IP
}

void loop() {
//*****
***

    Serial.print("Insira um valor para o servidor:"); //Solicita ao usuário entrar com o código da cidade
    while (!Serial.available()); //Aguarda os bytes de entrada serial
    while (Serial.available()) { //Guarda todos os caracteres da entrada da serial na String cod

        char c = Serial.read(); //receber cod pela serial sem final de linha \r\n
        cod = cod + c; //verifique na interface serial Nenhum final-de-linha
        delay(100);
    }
    incod = cod.toInt(); //Transforma a String cod em um inteiro incod
    Serial.println("\nValor inserido...");
```

```

//*****
***

while (!cliente.connect(host, 80))//Aguarda a conexão imprimindo |
{
    delay(500);
    Serial.print("|");
}

if (cliente.connect(host, 80))//
{
    Serial.println("Conectado e enviando requisicao HTTP");

    //Enviando para o host o valor lido.
//    cliente.print(String("GET /esp/get/key.php?s1=") + cod + //receber cod pela serial sem final de linha \r\n
//                " HTTP/1.1\r\n" + "Host: " + host + "\r\n" + "Connection: close\r\n\r\n");
/*
    cliente.print(String("GET /esp/get/key.php?s1=") + cod); //receber cod pela serial sem final de linha \r\n
    cliente.print (" HTTP/1.1\r\n");
    cliente.print ("Host: ");
    cliente.print (host);
    cliente.print ("\r\nConnection: close\r\n\r\n");
*/
    Serial.println("Aguardando resposta do servidor..."); 
    while (!cliente.available()) {
        Serial.print('*');
        delay(500);
    }
    while (cliente.available())//Lê a resposta da API e guarda todos os caractéres na String resposta
    {
        char c = cliente.read();
        Serial.print(c);
        resposta += c;
    }
}
delay(1000);
cod = ""; //Limpa a String
resposta = "";

}

```

INTRODUÇÃO A IoT COM MQTT

No modelo convencional de comunicação WEB, toda nova notificação somente será realizada se houver uma requisição do cliente. Isto torna o modelo ineficiente, pois em diversas situações é

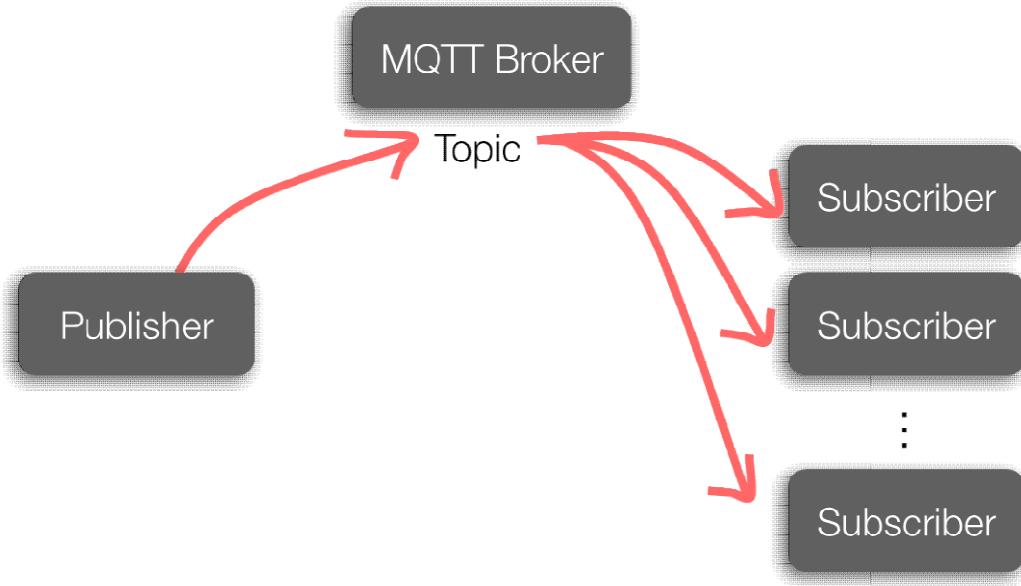
necessário atualizar o cliente no momento em que ocorreram modificações nos dados do servidor, sob pena da informação ficar depreciada, principalmente em redes sociais, bolsa de valores, informações climáticas entre outros. Outro problema do modelo, seria de que o usuário necessitaria enviar diversas requisições para o servidor, com a intenção de saber se existe alguma atualização da informação (*pulling*), aumentando assim o processamento do dispositivo do usuário e também gerando um desconforto, pois na maioria dos casos não há um tempo exato para a informação ser atualizada.

Uma solução é inverter os papéis, fazendo o servidor acordar o cliente, sempre que uma nova mensagem chegar. Este modelo é conhecido por *Push*. **Push refere-se a um modelo de comunicação entre cliente-servidor onde é o servidor que inicia a comunicação.** Neste modelo, é possível que um servidor envie dados para um cliente sem que ele tenha explicitamente solicitado.

Esta tecnologia de envio assíncrono de notificações por parte do servidor é o modelo mais implementado em IoT e também utilizado em diversos sistemas operacionais como o iOS, utilizando o *Apple Push Notification Service* (APNS); no Windows Phone com o *Microsoft Push Notification Service* (MPNS) e no Android usando o Google Cloud Messaging (GCM) ou o *Firebase Cloud Messaging* (FCM) que é a nova versão do GCM.

O MQTT (*significa Message Queuing Telemetry Transport*) é um protocolo leve, criado pela IBM em 1999, que permite a comunicação bidirecional (não como o protocolo HTTP, mas a seu modo específico) para coleta de dados e gerenciamento de dispositivos da Internet das Coisas.

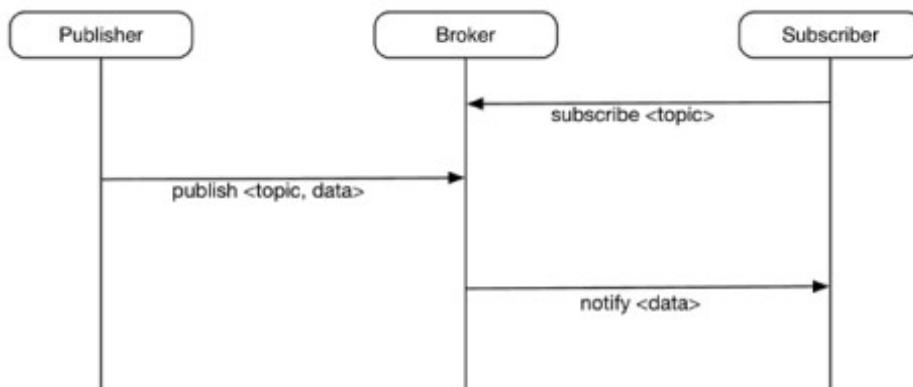
O MQTT permite o envio de mensagens para **filas**, e estas podem ser escutadas por outros dispositivos inscritos. O protocolo utiliza uma arquitetura *publish/subscribe*, em contraste ao *request/response* do protocolo web HTTP. A insustentável leveza desse ser consagrou-lhe como o protocolo oficial da Internet das Coisas (IoT) e das conversas entre máquinas (M2M), com seu uso largamente estimulado por gigantes como a Intel e IBM. Ele segue o padrão de projetos chamado observer, onde um grupo de objetos interessados assinam (*subscribe*) um determinado tópico. Outros objetos publicam informações nesses tópicos e os assinantes recebem.



O centralizador ou servidor de notificação do MQTT é chamado de *broker* que funciona como o servidor de notificação no método push. Ele que recebe e envia mensagens para todos os clients que estabelecerem conexão e todos os dispositivos devidamente inscritos para receber a notificação ou *payload*.

O protocolo MQTT é baseado no TCP/IP e ambos, cliente e broker, necessitam da pilha TCP/IP para o seu funcionamento. O MQTT está na mesma camada OSI que o HTTP, porém a maior diferença entre os dois protocolos é o tamanho do payload. No HTTP, o payload é maior, o que inviabiliza o seu uso em conexões de baixa qualidade, como GSM por exemplo.

O publisher é responsável por se conectar ao broker e publicar mensagens. Já o subscriber é responsável por se conectar ao broker e receber as notificações ou mensagens que ele tiver interesse. Na imagem abaixo possível observar a arquitetura do paradigma pub/sub.



Para os nossos testes usaremos o broker do Eclipse, que é publicado pelo site iot.eclipse.org. É muito fácil usá-lo, basta acessar o endereço iot.eclipse.org na porta 1883 com um cliente MQTT como o *paho* e pronto. Na realidade é um broker Mosquitto que a eclipse deixou aberto ao público.

O servidor mosquito está para o MQTT assim como o Apache está para o HTTP, ele fará a conexão como servidor broker entre o pub/sub.

Caso prático de uso:

Como foi visto, o MQTT (Message Queue Telemetry Transport) é um leve e eficiente protocolo voltado para IoT. Possui header extremamente compacto e exige pouquíssimos recursos de hardware, processamento e baixíssima banda para funcionar. Como é calcado no TCP, possui também portanto garantia de entrega das mensagens (algo essencial em IoT). Além disso, utilizando-se de um servidor (broker) na nuvem, a comunicação torna-se a nível global. Ou seja, é possível interagir via MQTT com um Raspberry Pi de qualquer lugar do planeta que possua acesso à Internet. Outro ponto interessante é que este protocolo possui implementações nos mais diversos sistemas operacionais, permitindo assim que dispositivos totalmente diferentes "conversem" pela Internet

Servidor Broker utilizado

Há muitos servidores brokers disponíveis na Internet (tanto grátis quanto pagos). Neste exemplo, utilizaremos o broker oficial do Eclipse (iot.eclipse.org), pois consiste de um broker grátis e com baixo downtime. Um exemplo de firmware testado para enviar dados para o broket mqtt e acompnahr por terminais inscritos como subscriber no smartphone e no google chrome é mostrado abaixo.

```
#include <ESP8266WiFi.h>
#include <PubSubClient.h>

const char* ssid = "SETA-AF82";
const char* password = "14502384";
const char* mqtt_server = "iot.eclipse.org";

WiFiClient espClient;
PubSubClient client(espClient);
```

```
const byte ledPin = 13; //D7

void callback(char* topic, byte* payload, unsigned int length) {
    Serial.print("Message arrived [");
    Serial.print(topic);
    Serial.print("] ");
    for (int i=0;i<length;i++) {
        char receivedChar = (char)payload[i];
        Serial.print(receivedChar);
        if (receivedChar == '1') digitalWrite(ledPin, HIGH);
        if (receivedChar == '0') digitalWrite(ledPin, LOW);
    }
    Serial.println();
}

void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        Serial.print("Attempting MQTT connection...");
        // Attempt to connect
        if (client.connect("ESP8266 Client")) {
            Serial.println("connected");
            // ... and subscribe to topic
            client.subscribe("sanusbmqtt"); //Topic
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(5000);
        }
    }
}

void setup()
{
    Serial.begin(115200);

    client.setServer(mqtt_server, 1883);
    client.setCallback(callback);

    pinMode(ledPin, OUTPUT);
}

void loop()
{
```

```

if (!client.connected()) {
    reconnect();
}
client.loop();
}

```

A aplicação será iniciada e todas as mensagens recebidas no tópico **sanusbmqtt** irão aparecer na tela dos clientes inscritos.

Para testar, utilize qualquer cliente MQTT como, por exemplo, o MQTTLens para PC em

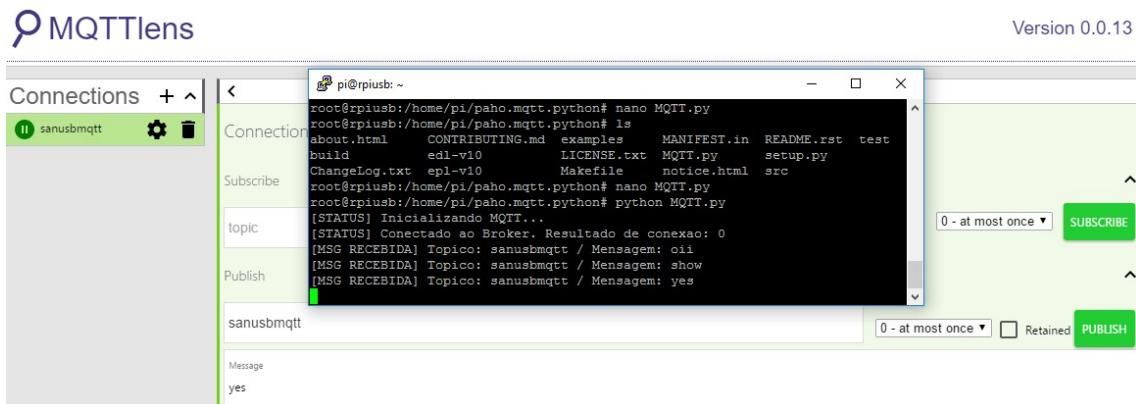
<https://chrome.google.com/webstore/detail/mqttlens/hemojaaeigabkcookmlgmdigohjobjm> ou o MyMQTT disponível para Android em https://play.google.com/store/apps/details?id=at.tripwire.mqtt.client&hl=pt_BR. Depois de instalar o cliente MQTT, conecte-se ao servidor broker iot.eclipse.org e publique no tópico **sanusbmqtt** uma mensagem qualquer. A mensagem irá aparecer na tela do terminal do Raspberry Pi, com ilustrados nas Figuras abaixo.

Connection Details

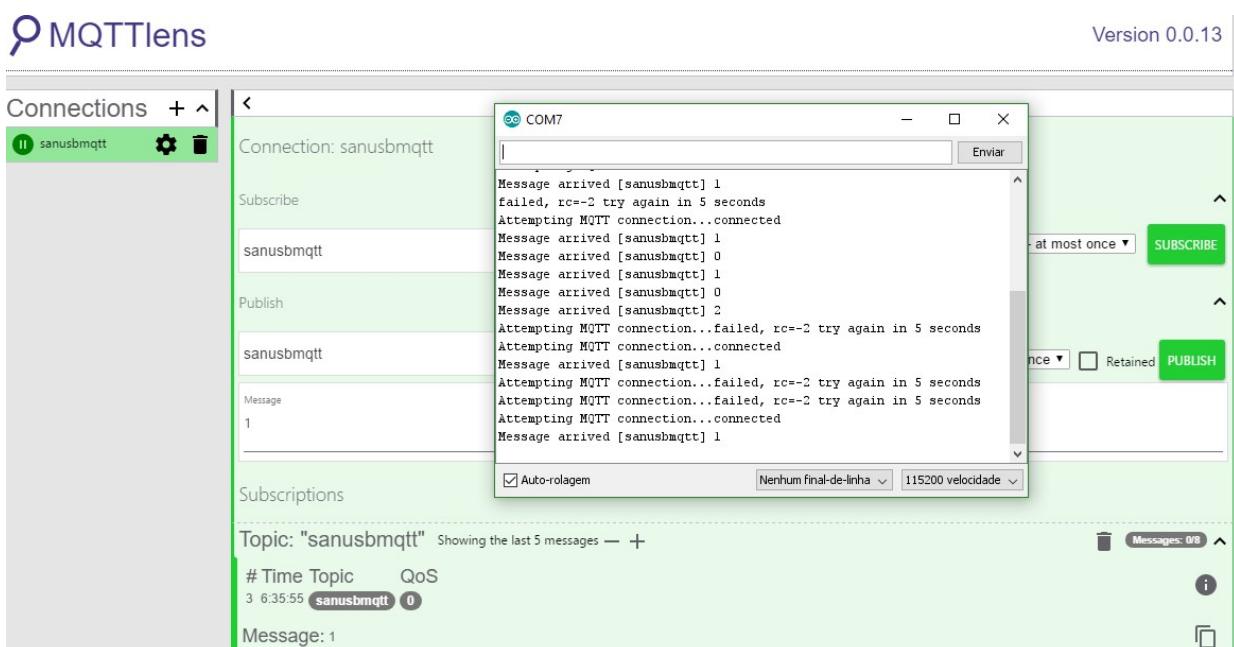
Connection name	Connection color scheme	
sanusbmqtt		
Hostname	Port	
tcp:// iot.eclipse.org	1883	
Client ID	Generate a random ID	
lens_VoxFWCTXJ1iVk6yoX60xzUu37AC	Generate a random ID	
Session	Automatic Connection	Keep Alive
<input checked="" type="checkbox"/> Clean Session	<input checked="" type="checkbox"/> Automatic Connection	120 seconds

Credentials

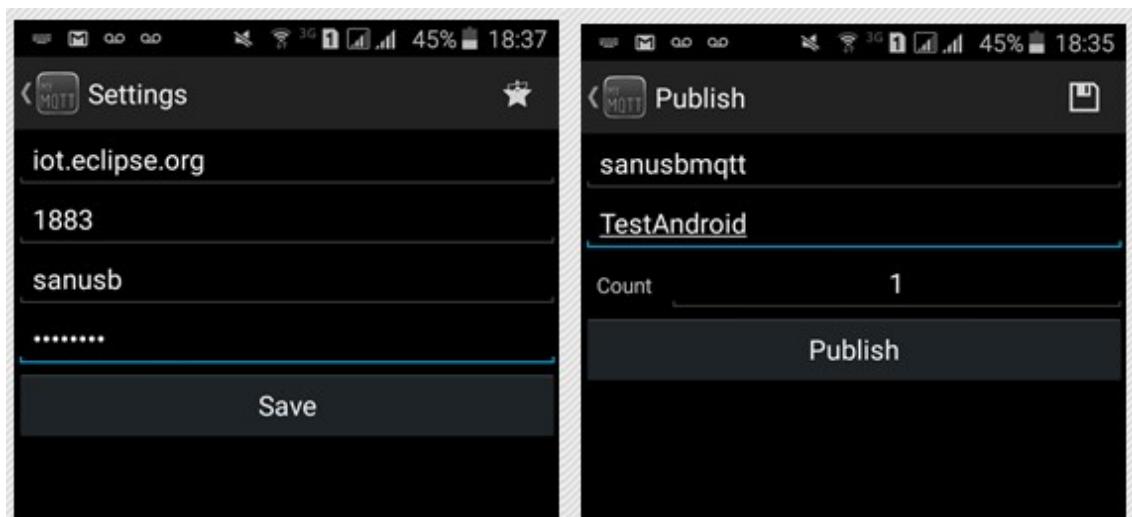
Username
sanusb
Password



O exemplo funcionando pe ilustrado na Figura abaixo.



Para sair da aplicação, pressione Ctrl + C. A configuração do Aplicativo Android é mostrada na Figura a seguir:



BOT TELEGRAM

O Telegram é um aplicativo de mensagens baseado em nuvem. O fato de ele ser baseado em nuvem permite que o usuário accesse suas mensagens em qualquer dispositivo. O Telegram também criptografa as mensagens, além de disponibilizar chats privados e prover a autodestruição de mensagens, o que dá segurança e privacidade aos usuários. Já para os desenvolvedores, existe a API do Telegram, a qual pode ser usada para o desenvolvimento de aplicação sobre a plataforma.

Por outro lado, bot nada mais é que a abreviação de robô, ou seja, um bot é um programa que atua em uma conta de usuário no Telegram que não é operada por um humano, mas sim, por um robô. O robô, tecnicamente chamado de software, pode falar sobre como está seu signo hoje, ajudar nas compras, procurar vídeos, servir para fazer pesquisas, entre outros.

Suponha que você esteja falando sobre algum tutorial do grupo SanUSB e gostaria de mostrar um dos vídeos no YouTube. Use um robô para isso. O bot irá buscar no YouTube os vídeos. Para isso, use no campo de escrita de mensagem o comando @vid sanusb ou @vid acompanhado do nome do assunto, para encontrá-lo. Veja, abaixo uma tabela de robôs que podem ser usados:

Robô ou bot	Descrição	Serviço	Parâmetros
gif	Busca e envia GIFs animados	giphy.com	
vid	Compartilha vídeos	youtube.com	
bing	Busca e envia imagens	bing.com	

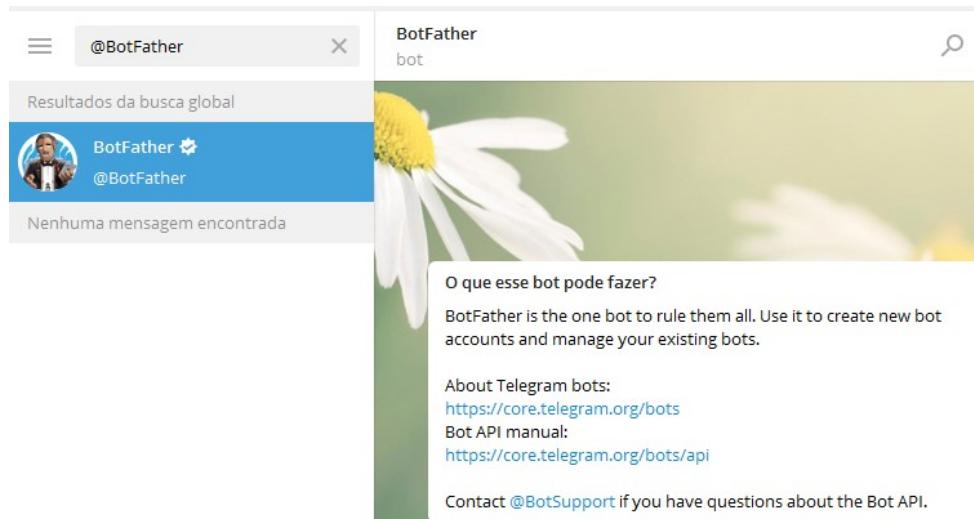
pic	Busca e envia imagens	yandex.com	
wiki	Compartilha artigos da Wikipedia	en.wikipedia.org	Use wiki para ver em português
imdb	Compartilha informações sobre filmes	imdb.com	
bold	Aplica formatação básica ao seu texto		Há um limite de 120 caracteres para a mensagem e você pode usar italic, como parâmetro para obter itálico

Loja de Bots do Telegram <https://storebot.me/>

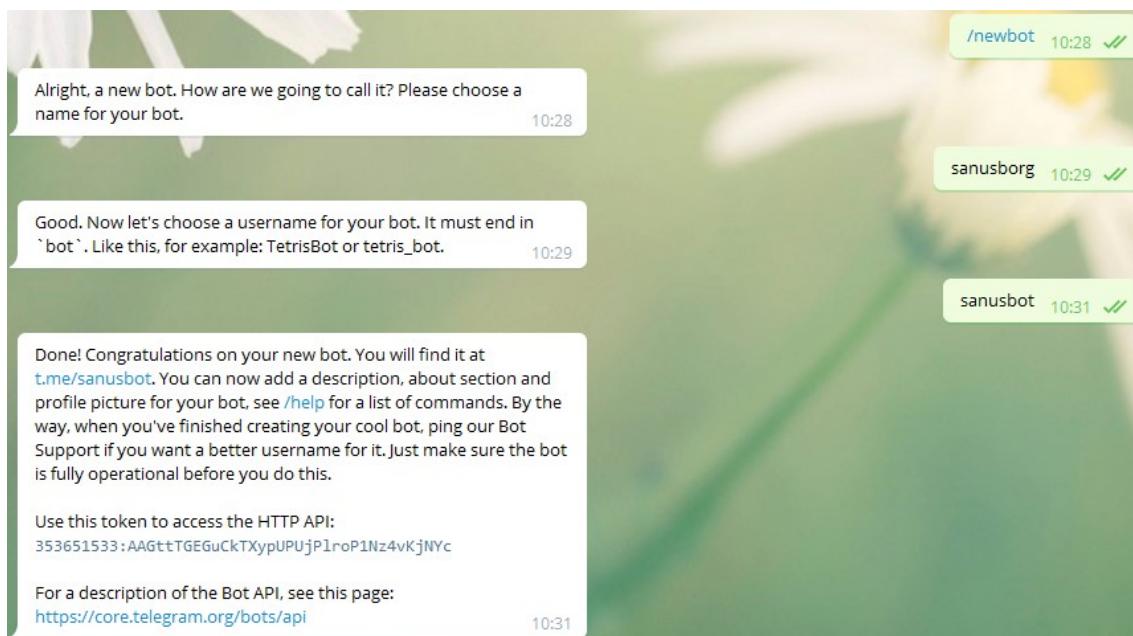
Criando um Bot

A forma de criar bots no telegram é bem simples:

1. Adicione um chat com o @BotFather no Telegram e toque em Iniciar.



2. Escreva: /newbot
3. Ele pedirá um nome para o seu bot. Escreva o nome.
4. Se o nome estiver disponível, você terá que adicionar um username com final bot. Digite o username. Após isso, inicie o bot, ele aparecerá na lista lateral como um contato. Se escrever algo clicando em cima do Bot criado, o comando *Getupdates* já apresentará o conteúdo em JSON. Isso é necessário para obter o número `chat_id` do bot.
5. Você receberá um token de acesso. Agora é só rodar o firmware de comunicação do ESP com o bot.



Como primeira aplicação com Telegram, veja o código do exemplo flashLedBot abaixo. Bibliotecas disponíveis em <https://github.com/SanUSB/TelegramESP>. O exemplo aponta para o Bot **sanusblaese**. Para utilizá-lo abra o telegram e procure por **sanusblaese** após rodar o código abaixo no ESP8266. O código e bibliotecas estão disponíveis em: <https://github.com/SanUSB/TelegramESP>.

Basicamente o que o código faz são os seguintes passos:

1. Conecta na rede WiFi configurada—Método `setupWifi`.
2. Configura os pinos.
3. Fica em loop consultando por novas mensagens no Telegram.
4. Ao receber novas mensagens, começa o tratamento de cada comando.

```
#include <ESP8266WiFi.h> //Video: https://www.youtube.com/watch?v=XWMOyjoGw_U
#include <WiFiClientSecure.h>
#include <ESPTelegramBOT.h>

// Initialize Wifi connection to the router
char ssid[] = "xxxxxxxxxxxxxx";
char password[] = "yyyyyyyyyyyy";

// Initialize Telegram BOT
#define BOTtoken "343986945:AAGuf50736pi2tmkMuWW35DhFmFl3xcp4Uo" //token of BOT
//search in Telegram the Bot: sanusblaese
```

```
TelegramBOT bot(BOTtoken);

int Bot_mtbs = 2000; //mean time between scan messages
long Bot_lasttime; //last time messages' scan has been done
bool Start = false;
int ledStatus = 0;

void setup() {
    Serial.begin(115200);

    // Set WiFi to station mode and disconnect from an AP if it was Previously
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(100);

    // attempt to connect to Wifi network:
    Serial.print("Connecting Wifi: ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(500);
    }
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    IPAddress ip = WiFi.localIP();
    Serial.println(ip);
    bot.begin(); // launch Bot functionalities
    pinMode(13, OUTPUT); // initialize digital pin 13 (D7) as an output.
    digitalWrite(13, HIGH);
}

void loop() {
    if (millis() > Bot_lasttime + Bot_mtbs) { //verifica tempo definido de leitura

        bot.getUpdates(bot.message[0][1]); // launch API GetUpdates up to xxx message (get update_id of last read
message)

        int conteudoIdAtual = bot.message[0][0].toInt() + 1; //conteudo id_update atual (anterior +1)
        // Serial.println(conteudoIdAtual);

        for (int i = 1; i < conteudoIdAtual; i++) { //se tiver conteudo

            String update_id = bot.message[i][0]; Serial.print("update_id: "); Serial.println(update_id);
```

```

String first_name = bot.message[i][1]; Serial.print("first_name: "); Serial.println(first_name);
String last_name = bot.message[i][2]; Serial.print("last_name: "); Serial.println(last_name);
String chat_id = bot.message[i][3]; Serial.print("chat_id: "); Serial.println(chat_id);
String date = bot.message[i][4]; Serial.print("date: "); Serial.println(date);
String text = bot.message[i][5]; Serial.print("text: "); Serial.println(text);

if (bot.message[i][5] == "/ledon") {
    ledStatus = 1;
    digitalWrite(13, HIGH); // turn the LED on (HIGH is the voltage level)
    bot.sendMessage(chat_id, "Led is ON", "");
}

//Escreve a URL
https://api.telegram.org/bot343986945:AAGuf50736pi2tmkMuWW35DhFmFl3xcp4Uo/sendMessage?chat\_id=309616823&text=Led\_is%20ON

if (text == "/ledoff") {
    ledStatus = 0;
    digitalWrite(13, LOW); // turn the LED off (LOW is the voltage level)
    bot.sendMessage(chat_id, "Led is OFF", "");
}

if (text == "/status") {
    if(ledStatus){
        bot.sendMessage(chat_id, "Led is ON", "");
    } else {
        bot.sendMessage(chat_id, "Led is OFF", "");
    }
}

if (text == "/start") {
    String welcome = "Welcome (/start) to SanUSB ESP8266 Telegram Bot library, " + first_name + ". Click /ledon : to switch the Led ON. /ledoff : to switch the Led OFF. /status : Returns current status of LED.";
    bot.sendMessage(chat_id, welcome, "");
}
}

bot.message[0][0] = ""; // reset new messages
//****************************************************************************
}

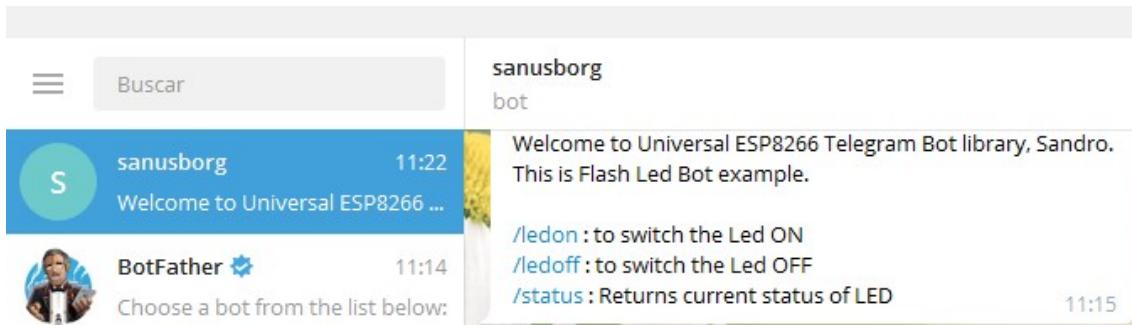
Bot_lasttime = millis();
}
}

```

Ao receber as mensagens, é possível colocar o tratamento para cada comando que você desejar, inclusive o Telegram tem suporte para enviar um teclado com os comandos pre-

definidos por você, como pode ser visto ao tratar o comando /options. Os comandos criados foram:

- **/start**: Comando enviado ao iniciar o chat Telegram.O ESP devolve um texto para o Telegram com todos os comandos disponíveis e o status atual do led.
- **/ledon e /ledoff**: Liga e desliga o LED.
- **/status**: O ESP devolve o status atual do led.



API de Bots do Telegram

O acesso de mensagens de Telegram é realizado utilizando objetos json com os comandos disponibilizados em <https://core.telegram.org/bots/api> .

A Bot API é uma interface baseada em HTTP criada para desenvolvedores interessados em construir bots para Telegram.Existem duas formas exclusivas de receber atualizações de seu bot - o método **getUpdates** por um lado e **Webhooks** do outro, que é um simples evento de notificação assíncrona via HTTP POST.As atualizações recebidas são armazenadas no servidor até que o bot as receba de qualquer forma, mas elas não serão mantidas por mais de 24 horas. Todas as formas retornam um objeto JSON.

getUpdates: este método é usado para receber atualizações e retorna uma matriz de objetos de atualização.

<https://api.telegram.org/bot<token>/getUpdates>

<https://api.telegram.org/bot353651533:AAAGttTGEGuCkTXypUPUjPlroP1Nz4vKjNYc/getUpdates>

```
{"ok":true,"result":[{"update_id":898314563,"message":{"message_id":157,"from":{"id":309616823,"first_name":"Sandro","last_name":"Juca"},"chat":{"id":309616823,"first_name":"Sandro","last_name":"Juca","type":"private"},"date":1491916566,"text":"/ledon","entities":[{"type":"bot_command","offset":0,"length":6}]}}]
```

<https://api.telegram.org/bot399661601:AAEngDHU1M3LPCiPwN7ZyQfvq6M1CplUIlfU/getUpdates>

Exemplos de outros comandos:

getMe:<https://api.telegram.org/bot<token>/getMe>

Exemplo:

<https://api.telegram.org/bot343986945:AAGuf50736pi2tmkMuWW35DhFmFl3xcp4Uo/getMe>

<https://api.telegram.org/bot399661601:AAEnqDHU1M3LPCiPwN7ZyQfVq6M1CpIUIfU/getMe>

Reposta:

```
{"ok":true,"result":{"id":353651533,"first_name":"sanusborg","username":"sanusbot"}}}
```

```
{"ok":true,"result":{"id":343986945,"first_name":"sanusblaese","username":"sanusblaesebot"}}}
```

```
{"ok":true,"result":{"id":399661601,"first_name":"SanUSB","username":"sanusbbot"}}}
```

Enviando Mensagens:

Através do id obtido do comando *getUpdates*, basta construir a URL abaixo:

https://api.telegram.org/bot<token>/sendMessage?chat_id=yyyyyyyy&text>Hello

https://api.telegram.org/bot343986945:AAGuf50736pi2tmkMuWW35DhFmFl3xcp4Uo/sendMessage?chat_id=309616823&text>Hello

https://api.telegram.org/bot399661601:AAEnqDHU1M3LPCiPwN7ZyQfVq6M1CpIUIfU/sendMessage?chat_id=309616823&text>Hello

Utilizando o Webhook

O Webhook serve para integrar o bot com uma página php que define o seu comportamento. Dessa forma, quando um comando é enviado ao bot, a API do Telegram dispara um evento, o qual despacha a mensagem via HTTP para URL da página configurada junto ao Webhook.

Para isso, a API do Telegram possui o método *setWebhook*. Ele recebe como parâmetro a URL da página que “responde” pelo bot. Para que essa URL seja considerada válida pelo método *setWebhook*, é preciso que ela suporte o protocolo HTTPS. Sabendo disso, para setar uma página php como bot utilizando o método *setWebhook* é só abrir o browser e digitar a seguinte URL:

[https://api.telegram.org/bot\(SEU_TOKEN\)/setwebhook?url=https://\(SUA_URL\)/seubot.php](https://api.telegram.org/bot(SEU_TOKEN)/setwebhook?url=https://(SUA_URL)/seubot.php). Exemplo:

<https://api.telegram.org/bot399661601:AAEnqDHUIM3LPCiPwN7ZyOfVq6M1CpIUIfU/setwebhook?url=https://sanusb.tk/diego/nodejs/static/sanusbot.php>

Após acessar a URL na formatação acima, a resposta da API do Telegram será a seguinte:

```
{"ok":true,"result":true,"description":"Webhook was set"}
```

Nesse exemplo em questão, é utilizado o robô SanUSB. Abra o Telegram e busque pelo Robô SanUSB. Digite para ele o comando /usb. O robô encaminhará para um script em um servidor php (sanusbbot.php) que tratará essa informação e responderá o ID do de quem enviou a mensagem para ele.



Vale salientar, que por motivo de segurança, o Telegram mata todos os processos a cada 24 horas, sendo necessário realizar um get nessa URL com setwebhook todos os dias para manter o processo ativo. Para deletar o webhook e voltar a utilizar a função getUpdates, basta digitar na URL:

[https://api.telegram.org/bot\(SEU_TOKEN\)/setwebhook?url=](https://api.telegram.org/bot(SEU_TOKEN)/setwebhook?url=)

<https://api.telegram.org/bot399661601:AAEnqDHUIM3LPCiPwN7ZyOfVq6M1CpIUIfU/setwebhook?url=>

```
{"ok":true,"result":true,"description":"Webhook was deleted"}
```

Configurando o Webhook

Ok, então já é possível enviar e receber nossas mensagens através do link da API. Em seguida, é necessário obter essas mensagens de encaminhamento para o nosso próprio servidor através do método setWebhook (<https://core.telegram.org/bots/api#setwebhook>). Uma característica do setWebhook é que o Telegram exige que seu servidor use SSL, em outras palavras, que seu site tenha um certificado e pode ser visto corretamente via **https**.

É possível encontrar o script php comentado para Webhook no GitHub [aqui](#).

INTERRUPÇÃO DE TEMPORIZADORES

Os temporizadores associados com interrupção são muito utilizados para disparar eventos de forma assíncrona, de tal modo que o programa mantém uma execução normal e no estouro do temporizador, o programa salta para o evento para atender a interrupção. Isso permite parar dentro de uma função no loop para fazer uma verificação. Além dessa liberdade, é possível maior precisão no tempo, porque se o firmware tiver delay em qualquer função esse delay é impreciso.

A utilização de timer se prevalece de um recurso independente do fluxo principal do programa e no caso específico do ESP8266, você encontra 2 tipos diferentes, a Interrupção por hardware e a interrupção por software. Trataremos aqui de apenas um desses tipo, mas vamos fazer uma introdução de ambos.

INTERRUPÇÃO DE TEMPORIZADOR POR HARDWARE

Nesse caso, é utilizado um temporizador real configurado no microcontrolador ESP para gerar interrupções programadas. No exemplo abaixo, é utilizado o periférico interno timer 0.

```
#define LED 16

bool toggle = 0;

void timer0_ISR (void) {
    if (toggle) { digitalWrite(LED, HIGH); toggle = 0;
    } else { digitalWrite(LED, LOW); toggle = 1;
    }

    timer0_write(ESP.getCycleCount() + 80000000L); // para 80MHz -> conta 80000000 = 1sec
    Serial.println("timer0 interrompeu");
}

void setup() {
    Serial.begin(115200);
    pinMode(LED, OUTPUT);
```

```

noInterrupts();
timer0_isr_init(); //ISR = Interrupt service routine
timer0_attachInterrupt(timer0_ISR);
timer0_write(ESP.getCycleCount() + 80000000L); // para 80MHz, 80000000L = 1sec
interrupts();
}

void loop() {}

```

A instrução `ESP.getCycleCount()` conta os ciclos de instrução durante o processamento do firmware dentro da interrupção com uma variável interna de 32 bits, passível de estourar a cada 56 segundos. Com o processamento de 80MHz, o período de um microsegundo possibilita o processamento de 80 ciclos de instrução.

INTERRUPÇÃO DE TEMPORIZADOR POR SOFTWARE

Esse temporizador é baseado em software e pode suportar a geração de vários timers concorrentes. Por ser baseado em software, já pode-se deduzir que ele tem menos acuidade do que o temporizador por hardware. Os fatores que podem prejudicar sua acuidade são os recursos periféricos como o próprio WiFi e a percepção dessa imprecisão pode ser maior em menores intervalos. A função que trata o evento deve apenas setar uma flag e o tratamento deve ser feito no loop do programa. Mas em condições de programação de quem programa com threads e nesses casos específicos é possível fazer uso de chamada de funções dentro de interrupções.

```

extern "C"{
#include "user_interface.h"
}

os_timer_t mTimer;
bool _timeout = false;

//Nunca execute nada na interrupção, apenas setar flags!
void tCallback(void *tCall){
    _timeout = true;
}

void setup() {

```

```

Serial.begin(115200);
Serial.println();

//habilita e configura a interrupcao
os_timer_setfn(&mTimer, tCallback, NULL);
os_timer_arm(&mTimer, 1000, true); //1000 ms

}

void loop() {
//verificacao no loop
if (_timeout==true){
    Serial.println("cuco!");
    _timeout = false;
}
yield(); //um microsegundo pra respirar

}

```

A BIBLIOTECA DE INTERRUPÇÃO TICKER

Essa Biblioteca esp8266 é baseada em interrupção de temporizador por software e chama funções de interrupção periodicamente. A biblioteca Ticker suporta funções **attach_ms** (variável *int32* em milissegundo) e **attach** (variável *float* em segundo) tendo um argumento. Este exemplo abaixo executa dois tickers que ambos chamam uma função **setPin()**.

```

#include <Ticker.h>
//Biblioteca esp8266 que chama funções de interrupção periodicamente por software

Ticker tickerNivelAlto;
Ticker tickerNivelBaixo;

void setPin(int state) {
    digitalWrite(13, state);
    Serial.print("Interrupcao ticker: ");
    Serial.println(state);
}

void setup() {
    Serial.begin(115200);
    pinMode(13, OUTPUT);
    digitalWrite(13, LOW);
}

```

```
// a cada 500ms, chama setPin(0)
tickerNivelBaixo.attach_ms(700, setPin, 0);

// a cada 700 ms, chama setPin(1)
tickerNivelAlto.attach_ms(1500, setPin, 1);
}

void loop() {}
```

TEMPORIZADOR CÃO DE GUARDA (WATCHDOG TIMER)

Este exemplo é baseado na biblioteca de interrupção temporizada com a biblioteca ticker (sinalizador),

```
#include <Ticker.h>
//Biblioteca esp8266 que chama funções de interrupção periodicamente por software

Ticker TickSegundo;

volatile int watchdogCount = 0;

void ISRwatchdog() {
    watchdogCount++;
    Serial.println("interrompeu!");
    if ( watchdogCount == 10 ) {
        Serial.println("O cachorro mordeu!");
        ESP.reset();
    }
}

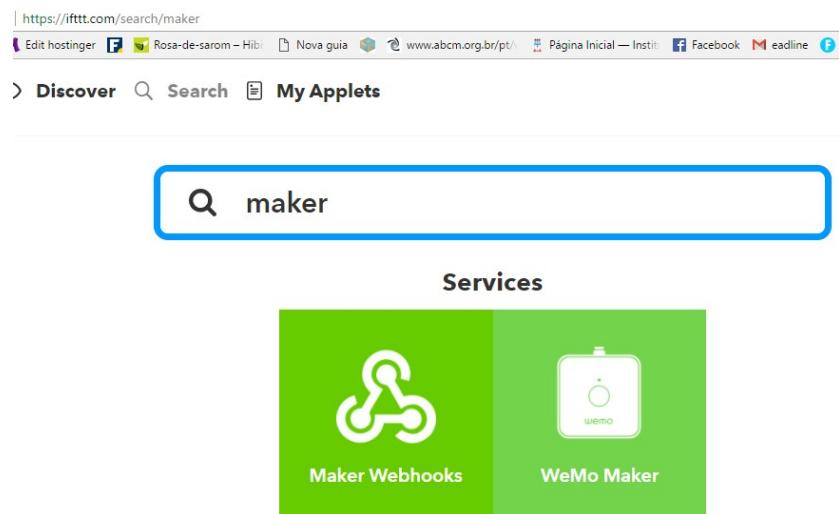
void setup() {
    Serial.begin(115200);
    Serial.println("Starting");
    TickSegundo.attach(1, ISRwatchdog);
}

void loop() {
    Serial.print("Watchdog counter = ");
    Serial.println(watchdogCount);
    watchdogCount = 0; //Se zerar o watchdogCount no loop nunca vai resetar
    delay(1000);
}
```

IFTTT (Se isso, então faça aquilo)

O IFTTT, em <https://ifttt.com/>, dispõe gratuitamente de um serviço IoT intermediário para notificação assíncrona de eventos.

Para configurar, Procure pelo canal maker e depois selecionar Maker Webhook e clique em Connect. Após conectado no Maker Webhook, existem duas possibilidades: Triggers e Actions.



Trigger

Receber um pedido da Web: Este trigger dispara sempre que o serviço Maker recebe uma solicitação da Web para notificá-lo de um evento. Para obter informações sobre o desencadeamento de eventos, acesse as configurações do serviço do Maker e, em seguida, a URL listada (web) ou toque em seu nome de usuário.

Actions

Faça uma WEB request: Essa ação fará uma solicitação da web para uma URL acessível publicamente. NOTA: Solicitações podem ser taxa limitada.

Integre outros serviços no IFTTT com seus projetos de DIY. Você pode criar Applets que funcionam com qualquer dispositivo ou aplicativo que possa fazer ou receber uma solicitação da Web. Veja como os outros estão usando o serviço Maker Webhooks e pode compartilhar seu projeto em hackster.io.

Clicando em **Documentations**, é mostrado a chave e os procedimentos para disparar um evento. Para disparar um Evento: Faça uma solicitação Web com POST ou GET para:

<https://maker.ifttt.com/trigger/{event}/with/key/cy2D03A6s0e6AwFWJos1zw>

Com um corpo {event} JSON opcional de:

```
{ "value1" : "", "value2" : "", "value3" : "" }
```

Os dados são completamente opcionais e você também pode passar valor1, valor2 e valor3 como parâmetros de consulta ou variáveis de formulário. Esse conteúdo será repassado para a ação em sua receita.

Você também pode testar com curl a partir de linha de comando.

curl -X POST

<https://maker.ifttt.com/trigger/{event}/with/key/cy2D03A6s0e6AwFWJos1zw>

Exemplos:

https://maker.ifttt.com/trigger/sanusb_on/with/key/cy2D03A6s0e6AwFWJos1zw

Exemplo com parâmetros:

https://maker.ifttt.com/trigger/sanusb1/with/key/hVaKHhVLemdaaRv4Wc_YlgtDeYrpeCU3rQz6wtkQc_M?value1=keki&value2=etev&value3=Juli

Agora clique em **MyApplets** e crie uma clicando em NewApplet, clique em +this e insira o canal que está utilizando que no caso é o Maker Webhook, clique em Receive Web request e insira o nome do evento.

<https://ifttt.com/create/if-receive-a-web-request?sid=7>

Step 2 of 6

Receive a web request

This trigger fires every time the Maker service receives a web request to notify it of an event. For information on triggering events, go to your Maker service settings and then the listed URL (web) or tap your username (mobile)

Event Name (required)

sanusb_on

Depois clique em that. Selecione Notification, instale o Ifttt no smartphone e coloque o mesmo nome do evento. Configure a notificação:

Step 5 of 6

Send a notification from the IFTTT app

This action will send a notification to your devices from the IFTTT app.

Notification (required)

O evento "{{EventName}}" pisca o Led pra voce! Em {{OccurredAt}}

Add ingredient

EventName

Value1
Value2
Value3

OccurredAt

Pronto, o *Applet* está configurado para interagir como Webhook, ou seja, um serviço intermediário assíncrono entre o processo IoT. O código exemplo abaixo envia a notificação quando o ESP é resetado.

```
#include <IFTTTMaker.h>
```

```
#include <ESP8266WiFi.h>
#include <WiFiClientSecure.h>

//----- Replace the following! -----
char ssid[] = "SETA-AF82";      // your network SSID (name)
char password[] = "14502384"; // your network key
#define KEY "cy2D03A6s0e6AwFWJos1zw" // Get it from this page https://ifttt.com/services/maker/settings
#define EVENT_NAME "sanusb_on" // Name of your event name, set when you are creating the applet

WiFiClientSecure client;
IFTTTMaker ifttt(KEY, client);

void setup() {

    Serial.begin(115200);

    // Set WiFi to station mode and disconnect from an AP if it was Previously
    // connected
    WiFi.mode(WIFI_STA);
    WiFi.disconnect();
    delay(100);

    // Attempt to connect to Wifi network:
    Serial.print("Connecting Wifi: ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print(".");
        delay(500);
    }
    Serial.println("");
    Serial.println("WiFi connected");
    Serial.println("IP address: ");
    IPAddress ip = WiFi.localIP();
    Serial.println(ip);

    //triggerEvent takes an Event Name and then you can optional pass in up to 3 extra Strings
    if(ifttt.triggerEvent(EVENT_NAME, ssid, ip.toString())){
        Serial.println("Successfully sent");
    } else {
        Serial.println("Failed!");
    }
}
```

```
}
```

```
void loop() {}
```

OUTRAS FORMAS DE PROGRAMAR O ESP8266

A Espressif empresa possui um repositório no GitHub (<https://github.com/espressif>), onde disponibiliza exemplos de código para firmwares com RTOS e comandos AT , e sua SDK (<https://github.com/esp8266/esp8266-wiki/tree/master/sdk>), por exemplo. Como mencionado também, a fabricante libera em seu GitHub o código-fonte do firmware que trata os comandos AT (https://github.com/espressif/esp8266_at) e faz a interface com WiFi correspondente aos comandos, ou seja, você pode baixar este código-fonte e modificá-lo para se adequar aos seus requisitos de projetos, ou simples interesses pessoais.

A arquitetura do módulo não é ARM, é Xtensa. Portanto, para compilar um código-fonte nativo para o ESP8266 é preciso ter um compilador adequado para a sua arquitetura de máquina.

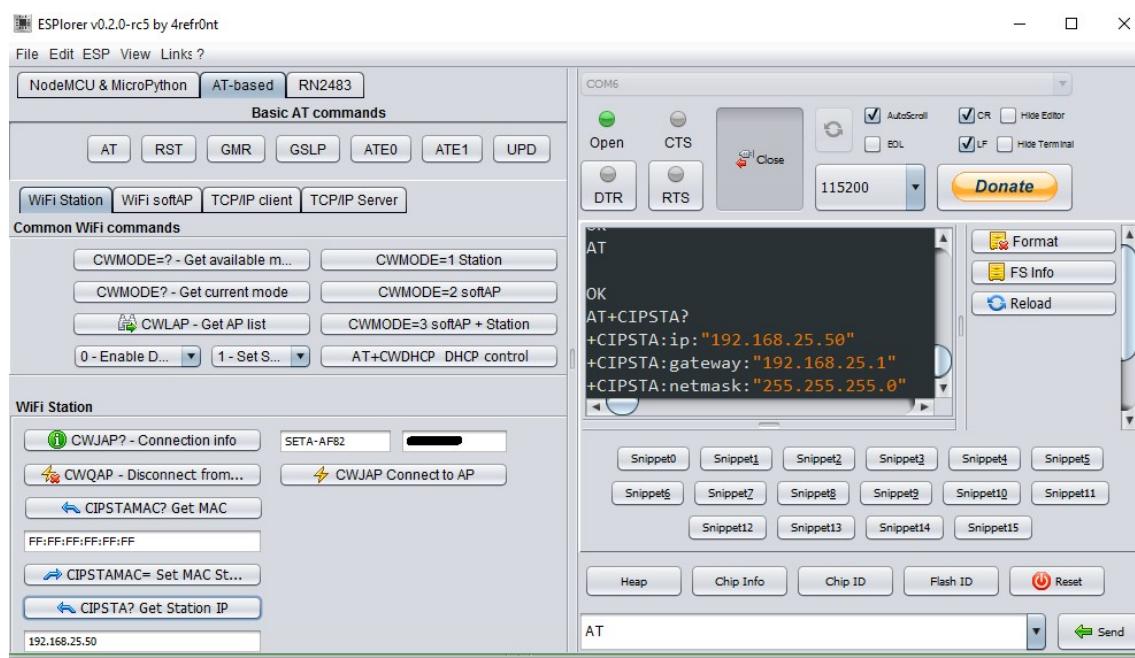
A Espressif também libera um Toolchain (<https://github.com/esp8266/esp8266-wiki/wiki/Toolchain>), contendo o compilador e demais ferramentas necessárias para compilar código nativo para o módulo. Ela também fornece detalhes de configuração e setup. Além disso, a Espressif liberou umKit de Desenvolvimento de software(SDK) para permitir criar códigos nativos, o que inclui uma *toolchain* com gcc adaptado para compilar código para a arquitetura Xtensa do módulo, além de outras ferramentas que permitem fazer o *upload* do código para o módulo. Há até mesmo uma máquina virtual (<http://bbs.espressif.com/viewtopic.php?f=5&t=2>) em formato VirtualBox com Linux Ubuntu contendo essas ferramentas já previamente instaladas e configuradas.

ESPLORER

Para aqueles que querem uma maior comodidade na hora de programar o ESP8266 com comandos AT com NodeMCU, é indicado o ESPlorer disponível em <http://esp8266.ru/esplorer/>. Este software ajuda muito na hora de configurar o módulo, pois já vem com interface serial e muitos comandos estão prontos como também é possível criar e salvar comandos pré-definidos, os chamados snippets.

Com o Esplorer, é possível configurar o módulo, enviar scripts, ler dados de configuração e outras funcionalidades.

As plataformas compatíveis são Windows(x86, x86-64), Linux(x86, x86-64, ARM soft & hard float), Solaris(x86, x86-64) e Mac OS X(x86, x86-64, PPC, PPC64). Segue um printscreenshot da interface:

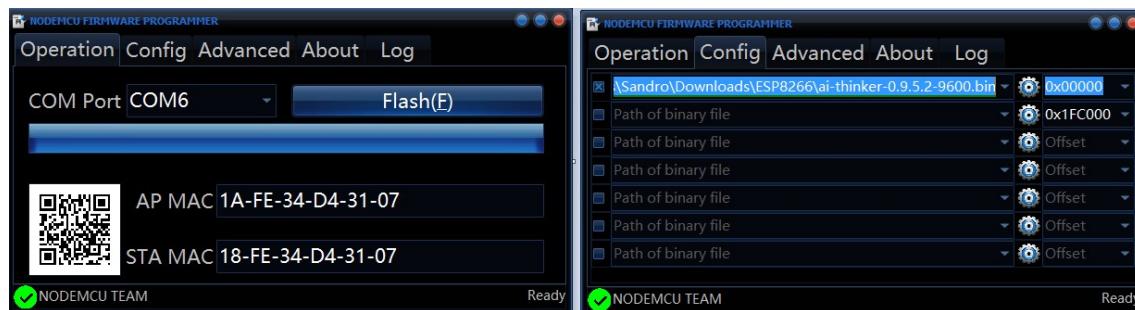


Como foi visto, existem diversas formas de programar o microcontrolador ESP8266 para IoT e as mais comuns são utilizando IDE do Arduino, SDK (Kit de desenvolvimento de software) da fabricante Espressif ou IDEs de desenvolvimento como a PlatformIO. É possível também interagir pela interface serial com outros microcontroladores ou processadores utilizando comandos AT.

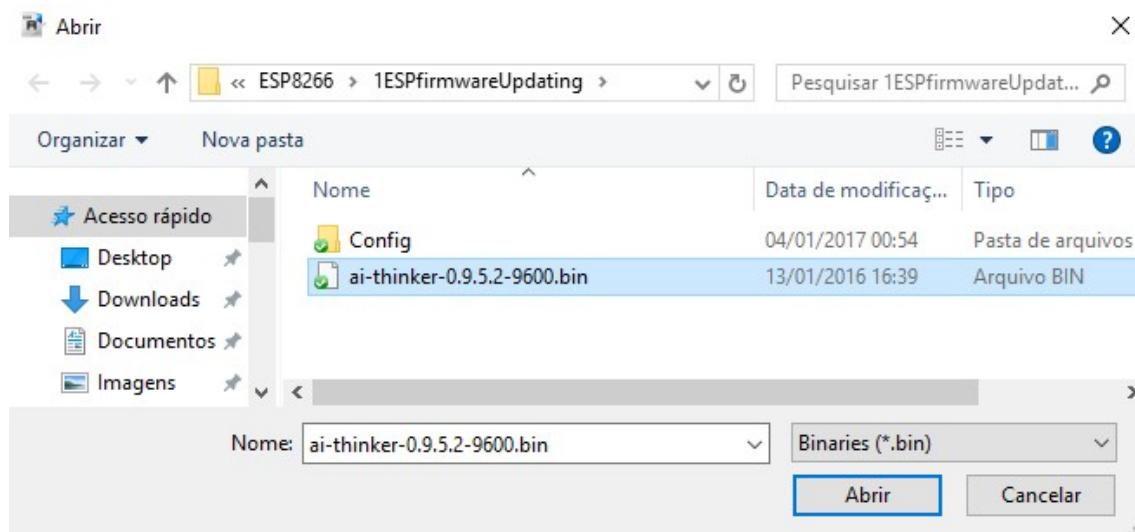
COMO RESTAURAR O FIRMWARE ORIGINAL COM COMANDOS AT

Se você precisar restaurar o firmware do ESP8266, siga as instruções abaixo:

- 4) Faça o download do software flasher <https://github.com/nodemcu/nodemcu-flasher> ou baixe a pasta com o executável e o binário de comandos AT em sanusb.org/esp/flasher/FirmwareAT.zip.
- 5) Descompacte arquivos em uma pasta (por exemplo, C:\ESP8266)
- 6) Execute o programa " nodemcu-flasher.exe " .



- 8) Clique na Aba Config e selecione, na posição de memória 0x00000, o arquivo " ai-thinker-0.9.5.2-9600.bin ".

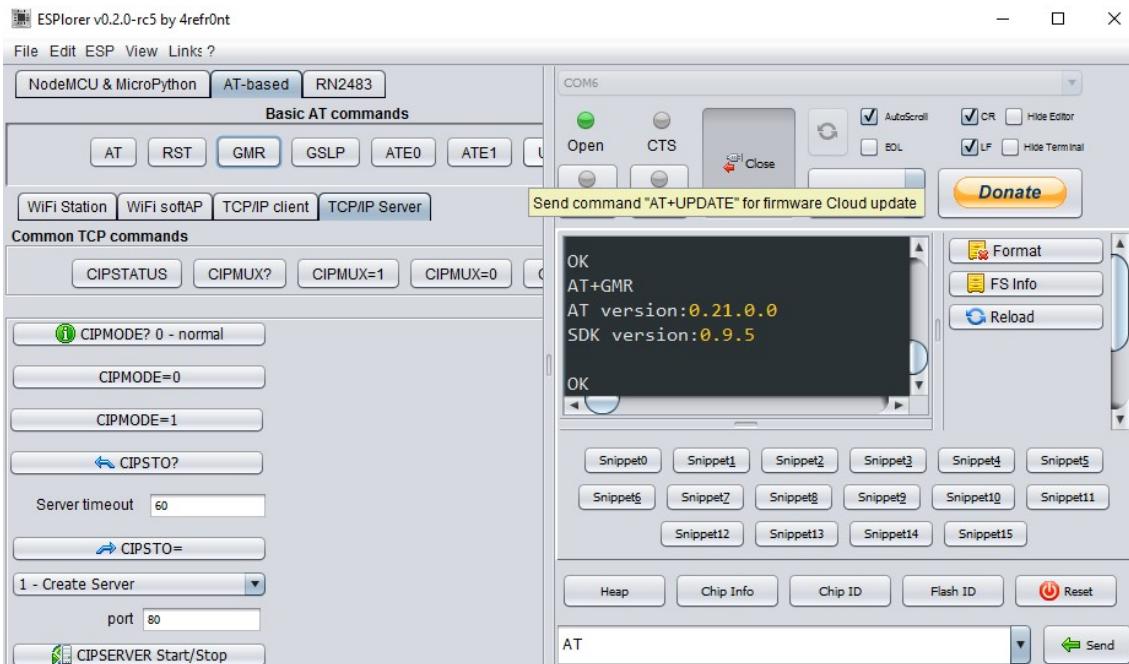


- 9) Clique na Aba Operation, Selecione a porta COM e clique no botão "Flash":

10) Aguarde enquanto o firmware é transferido para o ESP:

11) Feche a janela quando o download estiver concluído:

12) Você pode testar o firmware digitando "AT" na aba AT-based do ESPlorer:



Se aparecer "OK", o firmware do comando AT está instalado corretamente.

ESP8266 AT Command Set

Function	AT Command	Response
Working	AT	OK
Restart	AT+RST	OK [System Ready, Vendor:www.ai-thinker.com]
Firmware version	AT+GMR	AT+GMR 0018000902 OK
List Access Points	AT+CWLAP	AT+CWLAP +CWLAP:(4,"RochefortSurLac",-38,"70:62:b8:6f:6d:58",1)+CWLAP:(4,"LiliPad2.4",-83,"f8:7b:8c:1e:7c:6d",1)OK
Join Access Point	AT+CWJAP? AT+CWJAP="SSID","Password"	Query AT+CWJAP? +CWJAP:"RochefortSurLac" OK
Quit Access Point	AT+CWQAP=? AT+CWQAP	Query OK
Get IP Address	AT+CIFSR	AT+CIFSR 192.168.0.105 OK
Set Parameters of Access Point	AT+ CWSAP? AT+ CWSAP= <ssid>,<pwd>,<chl>, <ecn>	Query ssid, pwd chl = channel, ecn = encryption
WiFi Mode	AT+CWMODE? AT+CWMODE=1 AT+CWMODE=2 AT+CWMODE=3	Query STA AP BOTH
Set up TCP or UDP connection	AT+CIPSTART=? (CIPMUX=0) AT+CIPSTART = <type>,<addr>,<port> (CIPMUX=1) AT+CIPSTART= <id><type>,<addr>, <port>	Query id = 0-4, type = TCP/UDP, addr = IP address, port= port
TCP/UDP Connections	AT+ CIPMUX? AT+ CIPMUX=0 AT+ CIPMUX=1	Query Single Multiple
Check join devices' IP	AT+CWLIF	
TCP/IP Connection Status	AT+CIPSTATUS	AT+CIPSTATUS? no this fun
Send TCP/IP data	(CIPMUX=0) AT+CIPSEND=<length>; (CIPMUX=1) AT+CIPSEND= <id>,<length>	
Close TCP / UDP connection	AT+CIPCLOSE=<id> or AT+CIPCLOSE	
Set as server	AT+ CIPSERVER= <mode>[,<port>]	mode 0 to close server mode; mode 1 to open; port = port
Set the server timeout	AT+CIPSTO? AT+CIPSTO=<time>	Query <time>0~28800 in seconds
Baud Rate*	AT+CIOBAUD? Supported: 9600, 19200, 38400, 74880, 115200, 230400, 460800, 921600	Query AT+CIOBAUD? +CIOBAUD:9600 OK
Check IP address	AT+CIFSR	AT+CIFSR 192.168.0.106 OK
Firmware Upgrade (from Cloud)	AT+CIUPDATE	1. +CIPUPDATE:1 found server 2. +CIPUPDATE:2 connect server 3. +CIPUPDATE:3 got edition 4. +CIPUPDATE:4 start update
Received data	+IPD	(CIPMUX=0): +IPD, <len>; (CIPMUX=1): +IPD, <id>, <len>; <data>
Watchdog Enable	AT+CSYSWDTENABLE	Watchdog, auto restart when program errors occur: enable
Watchdog Disable	AT+CSYSWDTDISABLE	Watchdog, auto restart when program errors occur: disable

POSTANDO EM UM SERVIDOR THINGSPEAK COM COMANDOS AT:

<https://thingspeak.com/channels/22418>

<https://thingspeak.com/channels/22418#dataimport>

Entre em uma conta ThingSpeak

Crie um novo canal indo na página de canais e selecione Create New Channel

Enviar dados para o canal via URL:

https://api.thingspeak.com/update?api_key=API_KEY_DO_SEU_CANAL&field1=7

Exemplo:

<http://api.thingspeak.com/update?key=LLYF7WQB7ZCQAENI&field1=1&field2=2&field3=3&field4=47>

Ver os campos do canal:

http://api.thingspeak.com/channels/YOUR_CHANNEL_ID/feeds.json

<http://api.thingspeak.com/channels/22418/feed.json?key=LLYF7WQB7ZCQAENI>

<https://api.thingspeak.com/channels/22418/feeds.json>

<https://api.thingspeak.com/channels/22418/feeds.csv> (**Baixa o arquivo para Excel**)

Iframe gráfico

dinâmico: <http://api.thingspeak.com/channels/22418/charts/1?width=600&height=600&results=60&dynamic=true>

Utilizando o ESP8266 como cliente para envio de dados para um servidor na nuvem (método GET):

```
AT+CWJAP="ssid","senha"      // AT+CWJAP="HOTNET_SanUSB","sanusbelaese"
```

```
AT+CIPMUX=1
```

```
AT+CIPSTART=4,"TCP","184.106.153.149",80    //api.thinkpeak.com
```

```
AT+CIPSEND=4,44
```

```
GET /update?key=LLYF7WQB7ZCQAENI&field1=05
```

```
AT+CIPCLOSE
```

Resposta:

```
AT+CIPSTART=4,"TCP","184.106.153.149",80
```

```
4,CONNECT
AT+CIPSEND=4,44
OK

> GET /update?key=LLYF7WQB7ZCQAENI&field1=75
SEND OK

https://api.thingspeak.com/channels/22418/

+IPD,4,2:594,CLOSED
```

SERVIDOR IOT PUSH

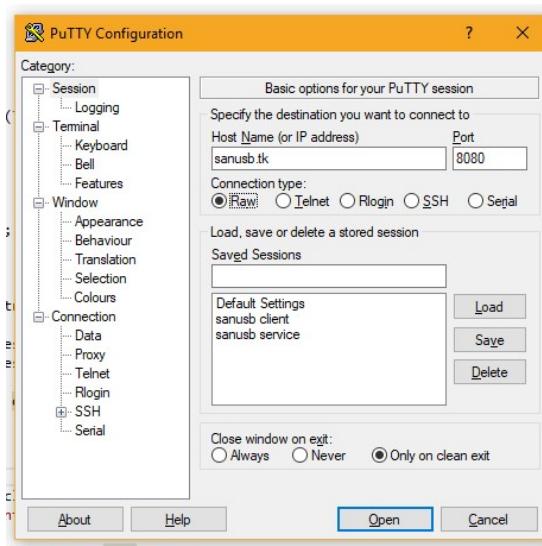
O IoT push se trata de um software instalado em um servidor. Esse software escuta conexões de clientes denominados dispositivo e serviço, e permite que serviços enviem mensagens para dispositivos que são identificados por códigos únicos (*strings*). Via TCP/IP, os serviços se conectam via na porta 8081 e os dispositivos se conectam na porta 8080.

Conexão via Putty

O tutorial a seguir utiliza o software Putty para se conectar via TCP/IP no servidor Push, mas qualquer software ou linguagem de programação com suporte a conexões TCP/IP podem ser utilizados.

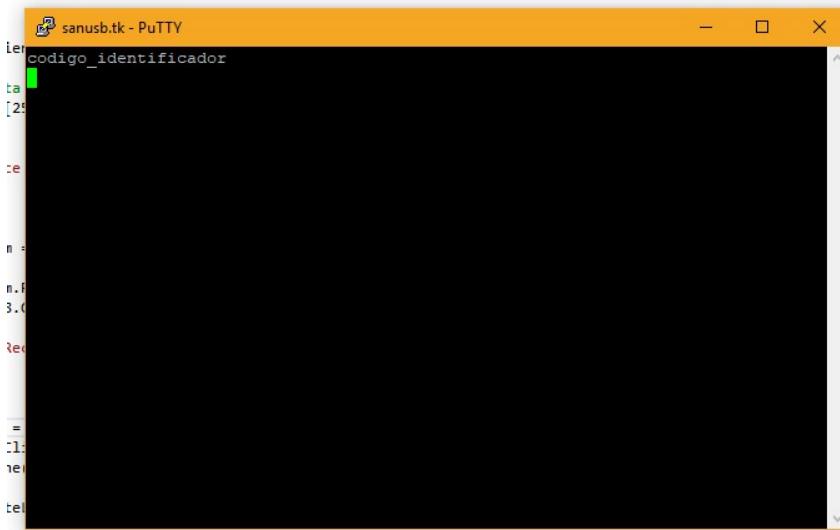
Para conexão é necessário seguir os passos a seguir:

1. O dispositivo deve se conectar via TCP/IP no endereço `sanusb.tk` na porta TCP 8080, conforme a figura abaixo.

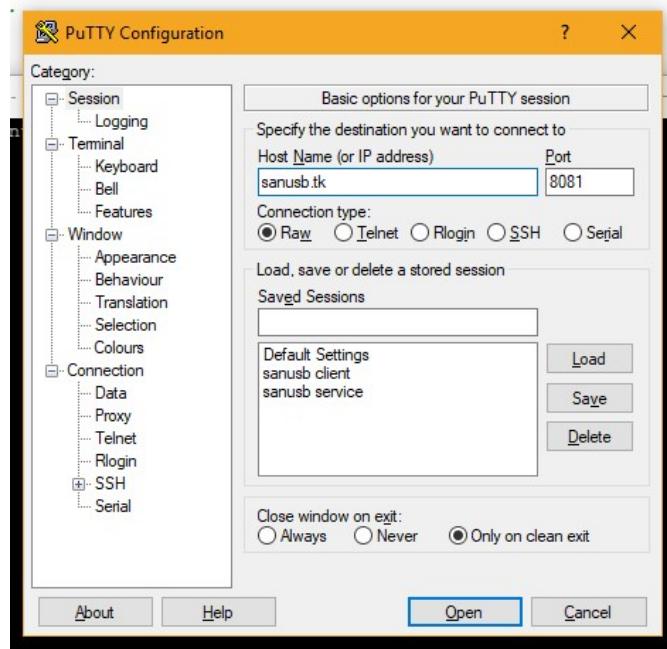


Logo ao se conectar na porta 8080, o dispositivo deve se identificar (*device_id*) enviando somente uma string única (no caso do Putty, digite a string e pressione

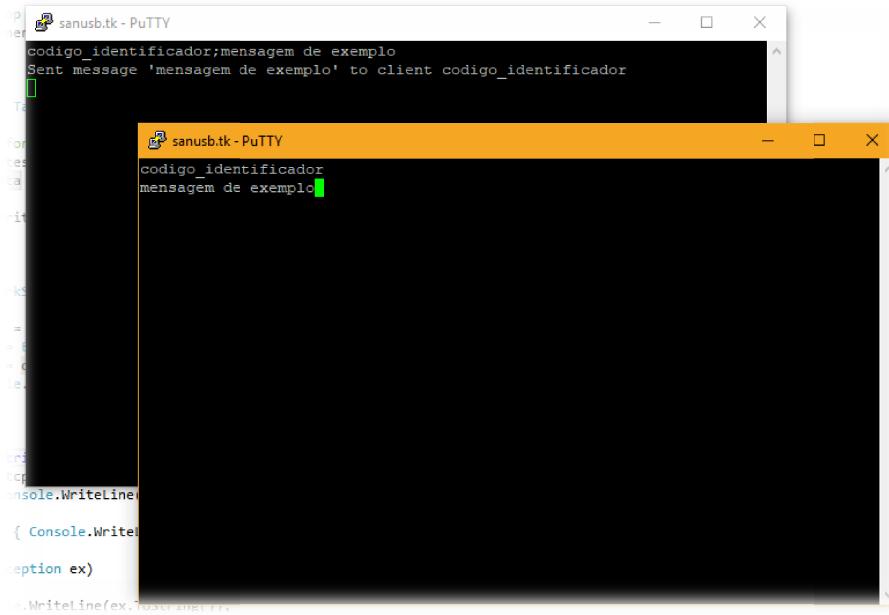
Enter). Obs.: Caso a string não seja única, a operação vai resultar na desconexão de um dispositivo previamente identificado por essa mesma string.



- Para se conectar como serviço, crie uma nova sessão e se conecte no mesmo endereço `sanusb.tk`, dessa vez na porta `8081` para enviar mensagens para dispositivos de acordo com sua identificação.



- Para enviar mensagens para um dispositivo, envie por esta conexão uma string no formato “identificador;mensagem” (sem as aspas), onde identificador (no exemplo, `código_identificador`) é o identificador do dispositivo de destino e mensagem é a mensagem que você deseja enviar para tal dispositivo (no caso do Putty, digite o texto e pressione Enter). Caso o dispositivo exista e esteja conectado, o resultado será semelhante ao mostrado a seguir:



CONEXÃO COM PIC E ESP8266

O tutorial a seguir descreve como se conectar no IoT Push como dispositivo utilizando um microcontrolador da família PIC16F com bootloader SANUSB instalado e módulo wireless ESP8266 conectado através da porta SERIAL UART.

Tal tutorial assume que o módulo ESP8266 esteja configurado como estação com comandos AT e que esteja conectado em uma rede WiFi que seja capaz de alcançar o servidor onde está o IoT Push. É possível realizar tal configuração através dos comandos AT suportados pelo módulo, documentados no seguinte link:https://www.espressif.com/sites/default/files/documentation/4b-esp8266_at_command_examples_en.pdf

No exemplo, o microcontrolador se conecta como dispositivo e exibe as mensagens recebidas através do IoT Push em um display LCD, mas tais mensagens podem ser consumidas de qualquer outra forma. Por conveniência foi utilizada uma biblioteca para controle do display LCD, mas seu uso não é obrigatório e a mesma pode ser removida caso não seja necessária.

Para executar o exemplo, siga os passos a seguir:

1). Instale o gravador SanUSB, disponível nos seguintes links:

- Windows 10: <https://www.microsoft.com/store/apps/9N26BFCFTGKV>
- Outras plataformas(incluindo Windows, Ubuntu e OSX): <http://www.sanusb.org/>

2). Instale a IDE MPLABX e o compilador XC8, ambos disponíveis em <https://www.microchip.com/mplab/mplab-ide-home>

3). Baixe o arquivo Chamou-PIC.X.zip. Este arquivo contém um projeto da IDE MPLABX, que contém o código necessário para a conexão do microcontrolador com o IoP Push.

O arquivo pode ser baixado através do link <https://github.com/SanUSB/Chamou-PIC.X>

4). Abra o projeto no MPLABX e modifique as variáveis de configuração de maneira que o microcontrolador possa se conectar no IoT Push.

As variáveis a ser modificadas são as que seguem:

- **ip_addr**: string contendo o nome DNS ou endereço IP do servidor onde está o IoT Push (no exemplo, sanusb.tk)
- **ip_port**: inteiro indicando a porta IP pela qual o IoT Push recebe conexões de dispositivos (por padrão, **8080**)
- **device_id**: Código identificador do dispositivo. Pode ser uma string qualquer, desde que seja única (no exemplo, “código”)

Veja as variáveis na imagem a seguir:

```

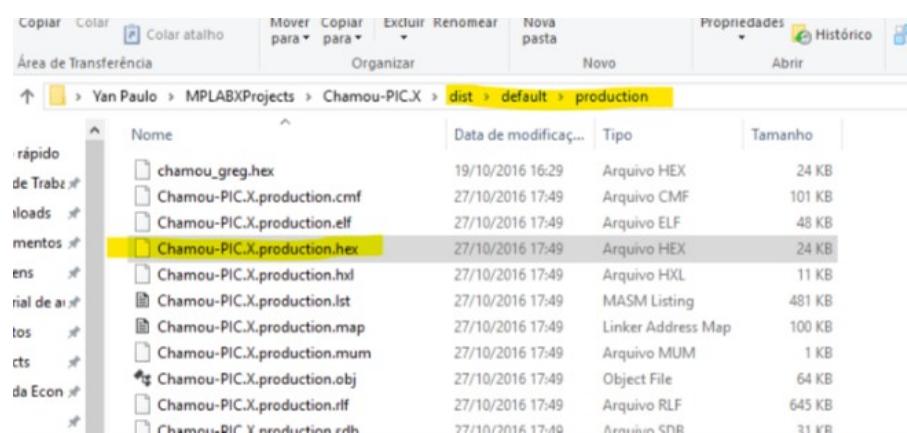
1  #include <string.h>
2  #include <stdlib.h>
3  #include "SanUSB1X.h"
4  #include "lcd.h"
5
6  #define BUFFER_SIZE 64
7
8  //INFORMAÇÃO: Manipule o valor recebido do servidor PUSH na linha 124!!!!!!
9
10 void clear_local_buffer(void);
11
12 typedef struct {
13     char items[BUFFER_SIZE];
14     int first;
15     int last;
16     int count;
17 } ring_buffer;
18
19 //Configurations
20 char ip_addr[64] = "sanusb.tk";
21 int ip_port = 8080;
22 char device_id[8] = "codigo";
23 //flags
24 char wifi_connected = 0, receive_enabled = 0;

```

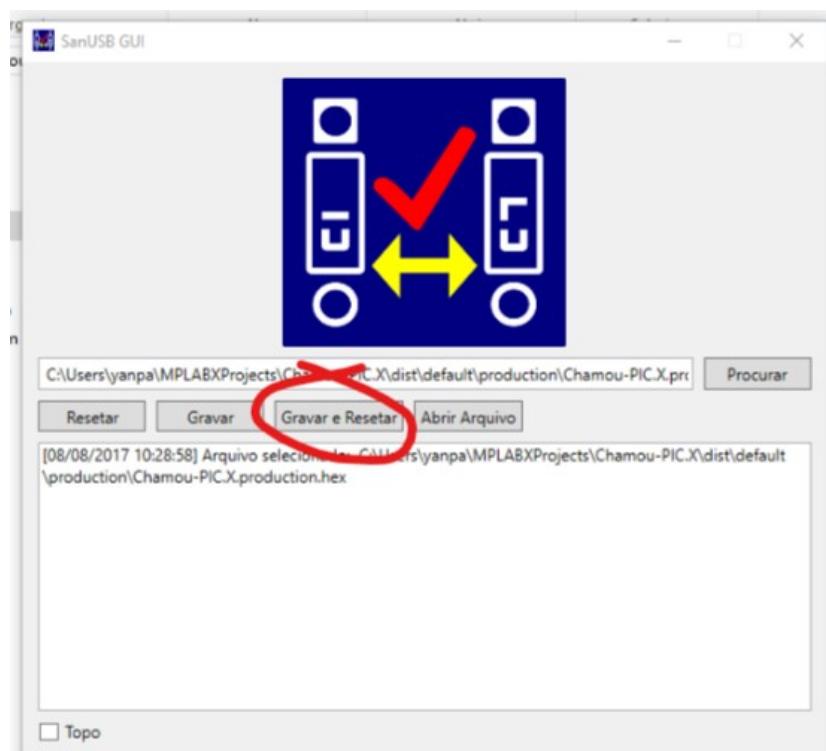
5). (Opcional) O comportamento padrão do projeto é receber o conteúdo e exibir em um display LCD conectado ao microcontrolador. Caso deseje alterar este comportamento, pule para a linha 124 e altere o código pelo que for conveniente. O trecho a ser alterado é exibido na figura a seguir:

```
120 //MAIN - MAIN.c
121
122 blink = 5;
123
124 str = strtok(NULL, ":");
125 /*-----'str' contém o texto recebido do servidor-----*/
126 /*-----Manipule 'str' aqui-----*/
127 Lcd_Cmd(LCD_CLEAR);
128
129 lcd_pos = size <= 18 ? (19 - size) / 2 : 1;
130 lcd_escreve(1, lcd_pos, str);
131 /*-----Pare de manipular 'str' aqui-----*/
132
133
```

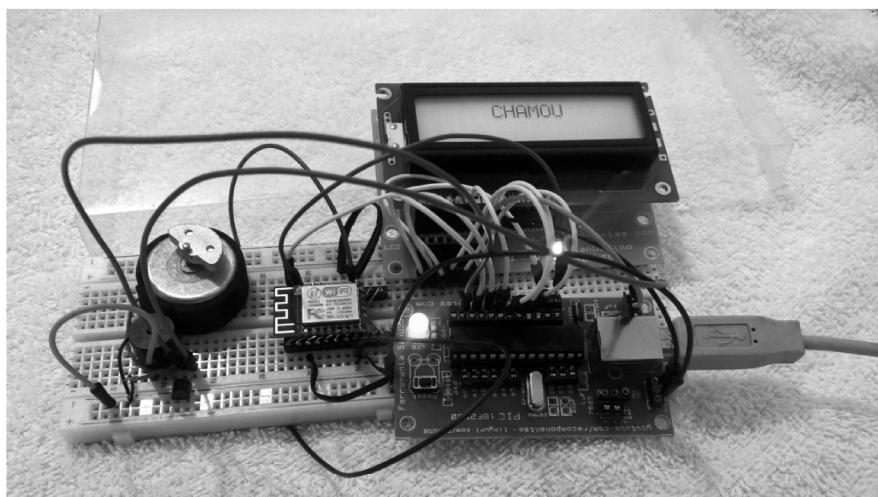
6). Compile o projeto e localize o arquivo “Chamou-PIC.X.production.hex” no diretório “dist/default/Production” do projeto (veja figura a seguir).



7). Utilize a ferramenta SanUSB para gravar o arquivo “Chamou-PIC.X.production.hex” no microcontrolador, lembrando de utilizar a opção “gravar e resetar”, conforme a figura a seguir:



8) Ao reiniciar, o microcontrolador deve automaticamente se conectar ao IoT Push com o id especificado e escutar por mensagens enviadas pelo mesmo. O código presente a partir da linha 124 deve executar automaticamente sempre que uma mensagem chegar.



LUA NO ESP8266 COM O NODEMCU

O NodeMCU está intimamente ligado à linguagem Lua que, curiosamente, é um projeto brasileiro nascido em 1993 na PUC-Rio, criado para ser fácil de embutir em outros ambientes, exatamente como ocorre no NodeMCU.

Por ser uma linguagem de programação voltada a aplicações especiais, é possível que você nunca tenha ouvido falar nela – mas saiba que ela é popular em seu nicho, sendo a linguagem de extensão escolhida para softwares de grande popularidade, como o Adobe Lightroom, o World of Warcraft e o Angry Birds. Ela é frequentemente apontada como a mais rápida entre as linguagens interpretadas, é open source, pequena, altamente portável nas mais diversas plataformas, e tem um manual de referência gratuito e bastante completo em <http://www.lua.org/manual/5.2/pt/>.

A linguagem Lua foi criada no Brasil na década de 1990, e é popular para acrescentar recursos e expansibilidade em uma série de plataformas, incluindo softwares como o World of Warcraft e o Adobe Lightroom.

Uma API bastante completa em https://github.com/nodemcu/nodemcu-firmware/wiki/nodemcu_api_en, com suporte a WiFi, TCP/IP, UDP, MQTT, i2c, SPI, tratamento de arquivos, timer, gpio, socket, one-wire, JSON e mais, está disponível no NodeMCU e facilmente acessível pela Lua.

Para exemplificar a sintaxe, vou mostrar uma forma de lidar com pinos de I/O digitais no NodeMCU. As operações são similares às que ocorrem no Arduino: você define o modo do pino, e aí envia o estado HIGH ou LOW para ele:

```
gpio.mode(2, gpio.OUTPUT)  
gpio.write(2, gpio.HIGH)  
gpio.write(2, gpio.LOW)
```

Em seu modo mais simples, o firmware NodeMCU transforma o ESP8266 em um ambiente interativo que você acessa por meio de um terminal serial, digitando expressões e comandos em Lua e vendo seu resultado, on-line. A partir daí, é possível aumentar a complexidade transferindo programas em Lua para execução pelo NodeMCU, ou mesmo instalando uma IDE tradicional.

É interessante perceber uma das diferenças profundas entre o modelo de desenvolvimento do Arduino e do NodeMCU: no Arduino, o seu programa é transferido para o microcontrolador na forma de firmware, e no NodeMCU o próprio ambiente NodeMCU é o firmware, e o seu programa é enviado a esse firmware para ser executado por ele.

INICIANDO COM O ESP32 DEVKIT

As principais plataformas de desenvolvimento de aplicações para o ESP32 são :

ESP-IDF – Framework de desenvolvimento IoT oficial da ESPRESSIF para o ESP32. Ele é o mais completo, mas é complicado de usar. Disponível em:

IDE Arduino: é o Ambiente de desenvolvimento mais conhecido de todos e bem mais fácil de usar. ESP32 – Arduino IDE

PlatformIO : é um ambiente similar ao do Arduino. Ainda não posso afirmar se é melhor. ESP32 – PlatformIO

Sobre o uso do ESP32 com a IDE Arduino , sugiro que veja nesse tutorial :

ESP32 – Passos com IDE Arduino:

Passos com IDE Arduino:

1. Abra a IDE Arduino;
2. Acesse a página da Web <https://github.com/espressif/arduino-esp32> e siga as instruções em instruções de instalação para a IDE Arduino IDE do sistema operacional que está usado.

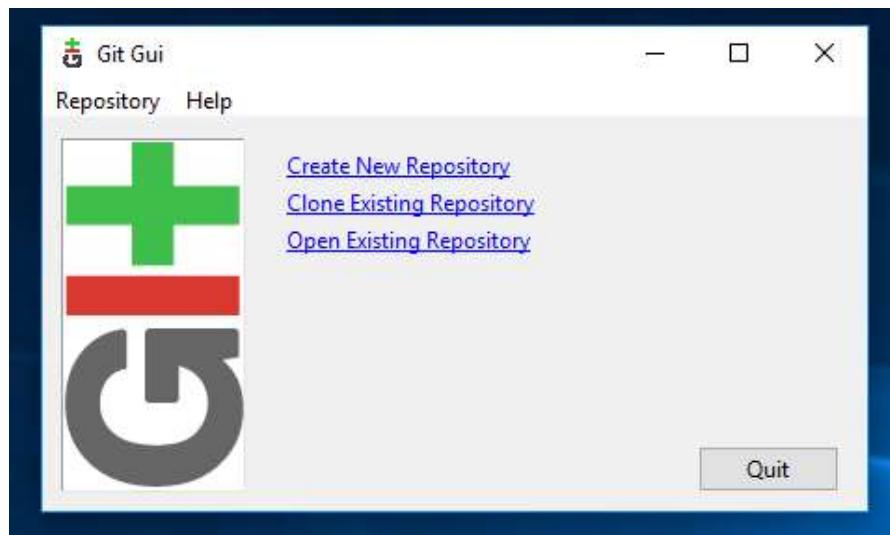
No caso do Linux, apenas copie e cole essas linhas no terminal para instalar:

```
sudo usermod -a -G dialout $USER && \
sudo apt-get install git && \
wget https://bootstrap.pypa.io/get-pip.py && \
sudo python get-pip.py && \
sudo pip install pyserial && \
mkdir -p ~/Arduino/hardware/espressif && \
cd ~/Arduino/hardware/espressif && \
git clone https://github.com/espressif/arduino-esp32.git esp32 && \
cd esp32/tools/ && \
python get.py
```

Depois inicie a IDE do Arduino e selecione em ferramentas a placa ESP32. Nesse caso é a DOIT ESP32 DEVKIT1.

No caso do Windows:

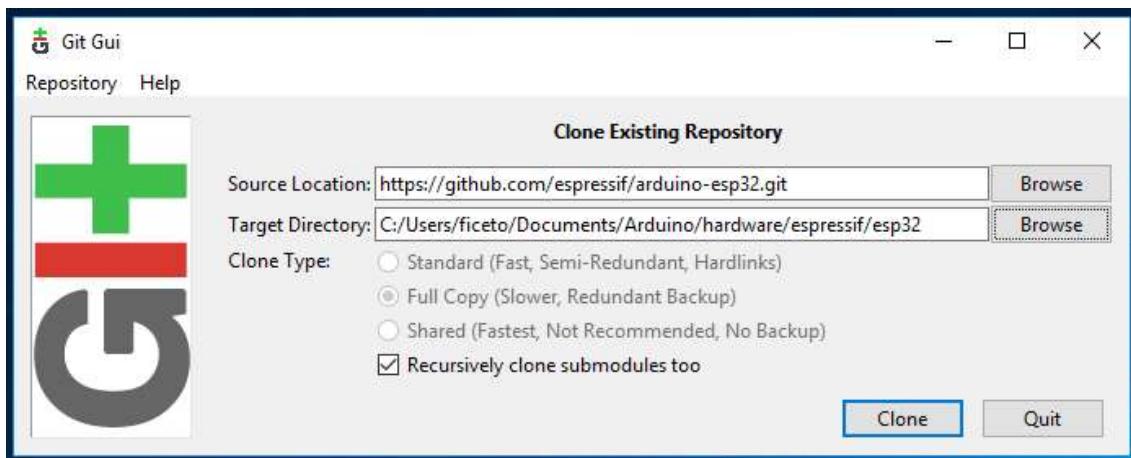
- Baixe e instale o Git de <https://git-scm.com/download/win>
- Inicie o Git GUI e execute as seguintes etapas:
 - Selecione o item de clone repositório existente (*Clone Existing Repository*):



- Selecione a origem e o destino:

Fonte: <https://github.com/espressif/arduino-esp32.git>

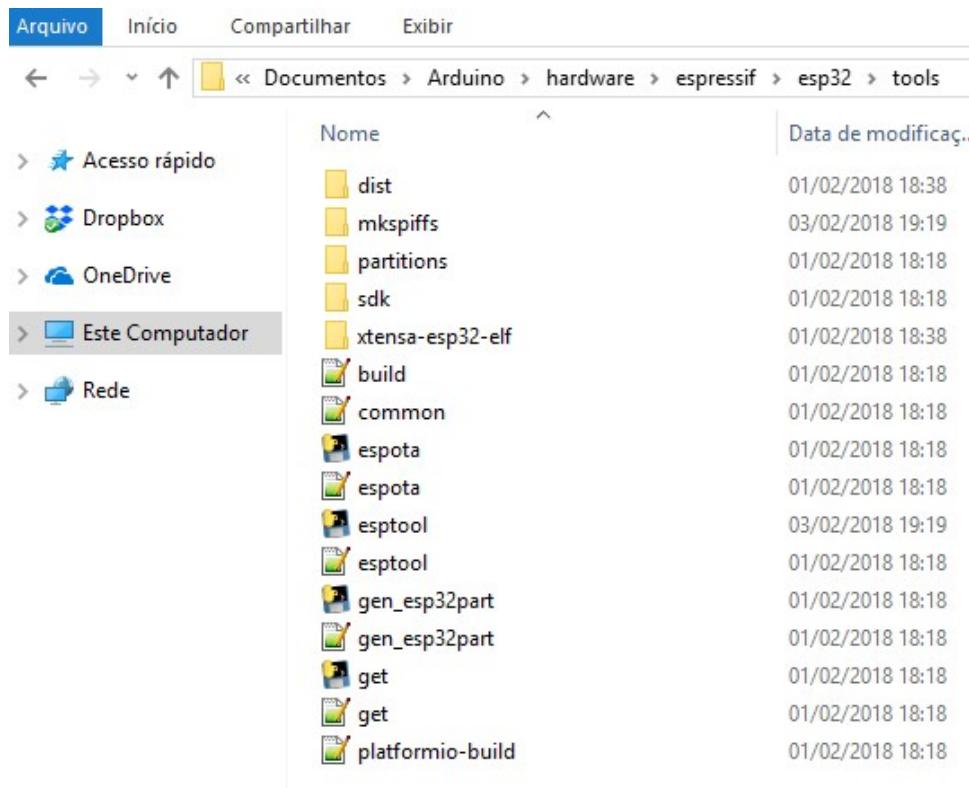
Destino: *C:\Users\[usuário]\Documents\Arduino\hardware\espressif\esp32*



- Clique em Clone para iniciar a clonagem do repositório.
- Abra *C:\Users\[usuário]\Documents\Arduino\hardware\espressif\esp32\tools* e clique duas vezes em *get.exe*. Irá aparecer a informação na tela abaixo.

```
C:\Users\sandr\Documents\Arduino\hardware\espressif\esp32\tools\get.exe
System: Windows, Info: Windows-10-10.0.16299
Platform: i686-mingw32
Tool xtensa-esp32-elf-win32-1.22.0-80-g6c4433a-5.2.0.zip already downloaded
Extracting xtensa-esp32-elf-win32-1.22.0-80-g6c4433a-5.2.0.zip
```

Quando o get.exe terminar, deverá surgir os seguintes arquivos na pasta *tools*:



3. Instale o driver para o chip serial usado para conectar a placa ao seu PC por USB, a partir da página

<http://www.silabs.com/products/mcu/pages/usbtouartbridgevcpdrivers.aspx>

4. Conecte a placa ao seu PC com um cabo USB a micro USB

Em Arduino IDE, Selecione -> Novo e cole o programa de teste abaixo:

```
void setup() {
    pinMode(2, OUTPUT); //pino com LED desfault na placa ESP32 DEVKIT
```

```

}

void loop() {

    digitalWrite(2, HIGH);

    delay(500);

    digitalWrite(2, LOW);

    delay(500);

}

```

- Selecione Ferramentas -> Placa DOIT ESP32 DEVKIT1

- Selecione Ferramentas -> Porta Instalada e clique em ***Upload***.

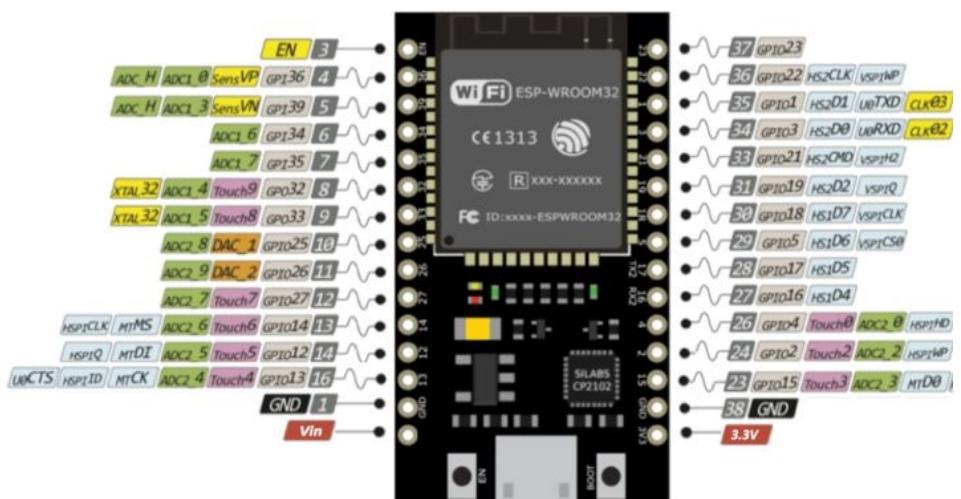
Obs: ***Mantenha o botão BOOT pressionado, após a compilação e durante todo o upload, mas isso pode não seja necessário;***

- Abra o Ferramentas -> Serial Monitor e selecione 115200 baud.

A figura abaixo mostra o número e a posição dos pinos GPIO e ser colocado dentro da função ***digitalWrite()***.

ESP32 PINOUT

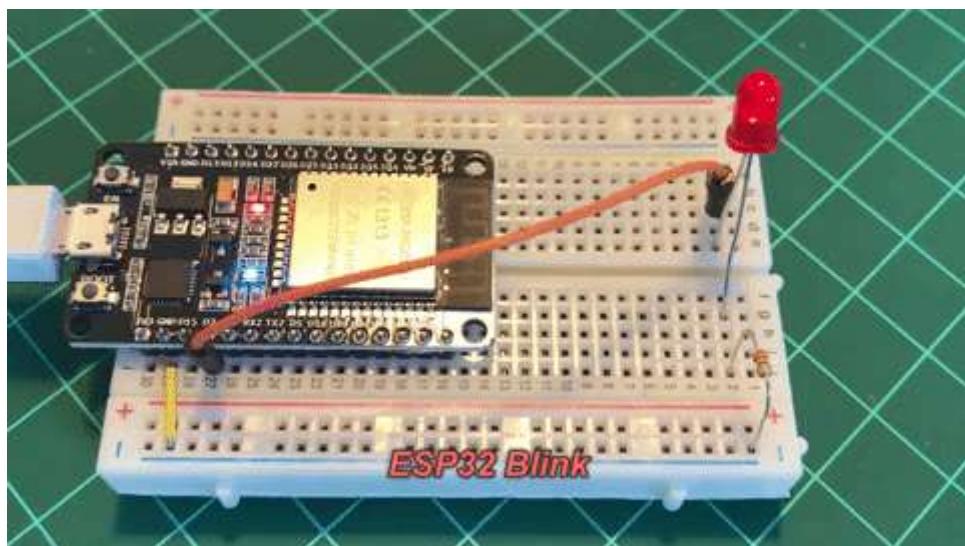
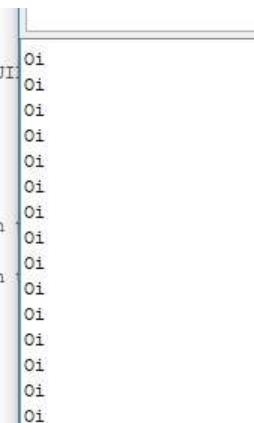
ESP32 DEVELOPMENT BOARD DUAL CORE ESP-32 & ESP-32S BOARD



O ESP32 DevKit possui um LED interno conectado ao GPIO 2. É importante verificar se "LED_BUILTIN" é automaticamente reconhecido pelo IDE. Caso contrário, é necessário adicionar o à linha de código como feito acima. Após a gravação, o Led

interno da placa ficará piscando e o ESP32 enviará **Oi** para serial como mostrado na ilustração abaixo.

```
1
2 void setup() {
3     // initialize digital pin LED_BUILTIN
4     pinMode(2, OUTPUT);
5     Serial.begin(115200);
6 }
7
8 void loop() {
9     digitalWrite(2, HIGH);    // turn LED on
10    delay(500);
11    digitalWrite(2, LOW);    // turn LED off
12    delay(500);
13    Serial.write("Oi\n");
14 }
```



Esse repositório <https://github.com/espressif/arduino-esp32> também contém bibliotecas de suporte para implementação de sistema operacional em tempo real FreeRTOS.

RTOS ESP32

RTOS é um sistema operacional projetado para funcionar em microcontroladores, ou seja, é um sistema operacional em tempo real, que é um tipo particular de sistemas operacionais. Para entender o que é um sistema operacional em tempo real, é necessário considerar uma das principais características dos sistemas operacionais de propósito geral, como o Windows ou o Linux, em que é possível ter vários processos e aplicativos em execução ao mesmo tempo parecendo que eles estão todos funcionando em paralelo.

No entanto, para um computador com um único núcleo apenas um processo pode ser executado a cada tempo. Dessa forma, o que os processadores fazem é mudar a execução entre tarefas em alta velocidade de modo que o usuário final tem a sensação de que as tarefas estão ocorrendo em paralelo.

Normalmente, não se pode prever o fluxo de execução dos múltiplos processos nos sistemas operacionais de propósito geral, que não são deterministas. Por outro lado, um sistema operacional em tempo real é construído para ter um padrão de execução determinista, ou seja, a execução em tempo real significa deve atender as tarefas em um prazo determinado e não significa necessariamente em alta velocidade. Além disso, é realizado o controle dessas tarefas, pois é possível programar o fluxo de execução de acordo com suas prioridades.

As tarefas geralmente são os blocos de distintos de operação dos sistemas operacionais em tempo real . Em FreeRTOS, as tarefas são implementadas como funções C e seguem um protótipo pré-definido, como pode ser visto abaixo:

```
void taskImplementingFunction( void * parameter )
```

Exemplo:

O firmware abaixo realiza a comutação de um led no loop infinito e o processamento de duas tarefas concorrentes que imprimem textos na execução e a comutação de um Led no *loop* infinito.

```
void setup() {
    Serial.begin(112500);
    delay(1000);
    pinMode(2, OUTPUT); //pino com LED default na placa ESP32 DEVKIT

    xTaskCreate(
        taskUm,      /* Ponteiro a função da task criada*/
        "TaskUm",    /* nome da task. */
        10000,       /* tamanho da pilha em words. */
        NULL,        /* Parâmetro passado como input da task */
        1,           /* Prioridade da task. */
        NULL);       /* Identificador da última task */

    xTaskCreate(taskDois,"TaskDois",10000,NULL,1,NULL);
}
```

```

void loop() {
    digitalWrite(2, HIGH);
    delay(500);
    digitalWrite(2, LOW);
    delay(500);
}

void taskUm( void * parameter )
{
    for( int i = 0;i<10;i++ ){

        Serial.println("task SanUSB 1");
        delay(1000);
    }

    Serial.println("Final da task SanUSB 1");
    vTaskDelete( NULL );
}

void taskDois( void * parameter)
{
    for( int i = 0;i<10;i++ ){

        Serial.println("task SanUSB 2");
        delay(1000);
    }

    Serial.println("Final da task SanUSB 2");
    vTaskDelete( NULL );
}

```

Os argumentos para as funções são descritos abaixo:

TaskCode: Ponteiro para a função que implementará a tarefa. Vamos criar duas funções, **TaskUm** e **TaskDois**, que vamos definir mais tarde e serão passadas neste argumento. As tarefas devem ser implementadas para nunca retornar (ou seja, loop infinito).

TaskName: O nome da tarefa, em uma string. Usaremos "TaskUm" e "TaskDois".

StackDepth: O tamanho da pilha da tarefa, especificado como o número de variáveis que pode conter (não o número de bytes). Não existe uma maneira simples de

determinar o tamanho da tarefa, embora alguns cálculos possam ser feitos. Neste exemplo simples, passaremos um valor suficientemente grande.

Parameter: Ponteiro para um parâmetro que a função da tarefa pode receber. Ele precisa ser do tipo `(void *)` [2]. Nesse caso, por simplicidade do código, não o usaremos, então passamos `NULL` no argumento.

Priority: Prioridade da tarefa. As duas tarefas terão com a mesma prioridade.

TaskHandle: retorna um identificador de referência da última tarefa em chamadas em funções (por exemplo, para excluir uma tarefa ou alterar sua prioridade). Além disso, para este exemplo simples, não vamos usá-lo, então será `NULL`.

Uma task pode ser excluída no FreeRTOS chamando ao final de sua execução a função `vTaskDelete(NULL)`.

Como foi visto, um RTOS tem a habilidade de ser *multitask*, ou, multitarefa. Criando tasks (como pode ser visto nos 2 artigos anteriores) escrevemos bem menos código e desperdiçamos menos processamento, uma vez que determinado código só será processado quando requisitado, ao invés de ter uma execução infinita em um loop. Mas quanto mais recursos, mais controle é necessário ter. Por exemplo, temos duas tarefas distintas que compartilham dados de uma mesma variável. Para que não haja colisão das tarefas, é necessário sinalizar que a variável está sendo manipulada e para isso podemos utilizar semáforos, semáforos binários ou MUTEX.

MUTEX

A palavra **MUTEX** significa ***MUTual EXclusion***. Quando um recurso faz uso do mutex, deve-se ter garantido o acesso através de um block no mutex. Quando termina-se o uso desse recurso, devolve-se a permissão de acesso ao recurso, caso contrário, nunca mais ninguém faz o acesso a outro código ou variável, como se fosse o uso de um bastão em uma corrida em grupo (*token de prioridade máxima*).

O MUTEX é simples de usar e resolve a grande maioria dos problemas. Já firam desenvolvidos programas bem complexos com threads controlando variáveis compartilhadas através de MUTEX.

Na realidade a principal diferença entre um mutex e um semáforo binário é a possibilidade de causar um *deadlock*, por isso que mutexes devem ser usados para guardar e processar urgente variáveis, enquanto semáforos são somente para realizar sincronia entre *tasks*.

Um mutex sempre deve ser adquirido primeiro e depois liberado sempre na mesma task (e sempre nessa ordem), quando a *task* adquire o mutex ela se torna dona daquele recurso, caso outra task seja bloqueada tentando adquirir o mutex, a task que é dona do mutex naquele momento tem a sua prioridade elevada para o mesmo nível da task que tenta adquirir, evitando o *deadlock*.

Deadlock acontece quando uma *task* trava esperando a outra e vice versa. Quando se tem dois mutexes e duas tasks, por exemplo: task 1 pega o mutex 1; task 2 pega o mutex 2; task 1 espera mutex 2 ser liberado pra liberar o mutex 1; task 2 espera o mutex 1 ser liberado pra liberar o mutex 2. Isso é deadlock e é um erro de design da aplicação.

Como mutexes não podem ser adquiridos em uma interrupt, eles por consequencia não podem ser liberados dela, já que a interrupt não pode adquirir um recurso.

Já semáforos são mecanismos de sincronização, nenhuma task é dona de um recurso e semáforos podem ser adquiridos dentro de uma task e liberados em uma interrupção.

Mutexes são usadas pra impedir que duas threads usem um mesmo recurso ao mesmo tempo, e acabem uma corrompendo a outra. ***Para uma porta serial, por exemplo, isso garante que uma task mande a mensagem completa antes de outra task começar***, e evita "Oi João!" e "Vai pra China Zé!" de sair: " Oi vai pra China João Zé". O procedimento comum é trava, usa e libera na mesma *task*. Se a mutex estiver ocupada, porém, a segunda task é obrigada a ficar esperando, o que é proibido em interrupções. Essa é a razão que não faz sentido usar mutex em interrupção.

Assim, no FreeRTOS mutex não pode ser utilizado dentro de funções de interrupção. Para criação e interação com o mutex, temos 3 funções:

- `xSemaphoreCreateMutex(void)`
- `xSemaphoreTake`
- `xSemaphoreGive`

Com a primeira, criamos um mutex. Com a segunda, obtemos o lock e com a terceira, liberamos o mutex. O código abaixo aplica um exemplo que altera uma variável global **my_shared_var** e aplica um delay global através de mutex, tanto em uma mesma task **teste**, quanto em uma task diferente **teste3**.

```

byte my_shared_var = 1;
SemaphoreHandle_t SanMutex;

void teste(void *pvParameters){
    while (true){
        xSemaphoreTake(SanMutex,portMAX_DELAY); //instancia o mutex que tambem é um semáforo
        Serial.print("Task Teste: ");
        Serial.println(my_shared_var);
        //my_shared_var = my_shared_var > 1 ? my_shared_var-1 : my_shared_var+1;
        if (my_shared_var > 1) {--my_shared_var;} else {++my_shared_var;}
        //Serial.println((char*)pvParameters);
        delay(1000);//Trava um segundo segurando o token do Mutex
        xSemaphoreGive(SanMutex);

    }
}

void teste3(void *pvParameters){
    while (true){
        xSemaphoreTake(SanMutex,portMAX_DELAY); //instancia o mutex que tambem é um semáforo
        Serial.print("Task Teste3: ");
        Serial.println(my_shared_var);
        if (my_shared_var > 1) {--my_shared_var;} else {++my_shared_var;}
        Serial.println((char*)pvParameters);
        delay(2000);//Trava um segundo segurando o token do Mutex
        xSemaphoreGive(SanMutex);

    }
}

void setup(){
    Serial.begin(115200);
    pinMode(2, OUTPUT); //pino com LED desfault na placa ESP32 DEVKIT
    SanMutex = xSemaphoreCreateMutex();
    if(SanMutex != NULL){
        xTaskCreatePinnedToCore(teste, "Print1", 10000,(void *) "pvParametros da task1", 3, NULL,0);
        xTaskCreatePinnedToCore(teste, "Print2", 10000,(void *) "pvParametros da task2", 3, NULL,0); //usa a mesma
        função
        xTaskCreatePinnedToCore(teste3, "Print3", 10000,NULL, 3, NULL,0); // Para criar com mutex:
        xTaskCreatePinnedToCore
    }
}

```

```

}

void loop(){
    digitalWrite(2, HIGH);
    delay(1000);
    digitalWrite(2, LOW);
    delay(1000);
    Serial.println("SanUSB");//Executa sem o delay global do mutex
}

```

The screenshot shows the Arduino IDE interface. The top menu bar includes Arquivo, Editar, Sketch, Ferramentas, and Ajuda. The title bar says "Rtos1ESP32Mutex". The code editor contains the provided C++ code for an RTOS task. The serial monitor window on the right displays the following text:

```

SanUSB
Task Teste: 1
SanUSB
Task Teste: 2
Task Teste3: 1

SanUSB
Task Teste: 2
SanUSB
Task Teste: 1
Task Teste3: 2

SanUSB
Task Teste: 1
SanUSB

```

Auto-rolagem

<http://www.dobitaobyte.com.br/como-colocar-o-esp32-com-rtos-na-ide-do-arduino/>

<http://www.dobitaobyte.com.br/passando-parametros-atraves-de-tasks-no-esp32/>

<http://www.dobitaobyte.com.br/selecionar-uma-cpu-para-executar-tasks-com-esp32/>

Instalação do Toolchain para Windows

Vamos começar instalando e configurando a **ToolChain** (conjunto de ferramentas). Baixe o pacote de programas zipados do link abaixo (560MB aproximadamente).

Windows all-in-one toolchain & MSYS2 zip

Descompacte o arquivo zipado em um diretório que achar melhor. Recomendo que seja em um subdiretório do raiz (para não ficar com um caminho muito grande e complexo). Após descompactá-lo, será criado um diretório `/msys32` . No meu caso escolhi o diretório `C:/` . Portanto o meu toolchain ficou no `C:/msys32` .

Na pasta `/msys32` , abra o aplicativo `mingw32.exe` . Crie uma pasta para a instalação da IDF com os comandos abaixo :

`pwd` => para identificar o diretório

`mkdir esp` => para criar o diretório esp

`cd esp` => para acessar o diretório esp

```
M ~/esp
sandr@DESKTOP-24QNIMS MINGW32 ~
$ pwd
/home/sandr

sandr@DESKTOP-24QNIMS MINGW32 ~
$ mkdir esp

sandr@DESKTOP-24QNIMS MINGW32 ~
$ cd esp

sandr@DESKTOP-24QNIMS MINGW32 ~/esp
$ pwd
/home/sandr/esp

sandr@DESKTOP-24QNIMS MINGW32 ~/esp
$ |
```

Baixando o ESP-IDF

O ESP-IDF compreende Aplicações e Bibliotecas, que devem ser baixadas do site da ESPRESSIF. Usando o aplicativo `mingw32`, dê os comandos abaixo, para baixar o IDF, na pasta `~/esp` (a pasta `/esp-idf` será criada automaticamente):

ESP-IDF será baixada na pasta `~/esp/esp-idf`
`git clone --recursive https://github.com/espressif/esp-idf.git`

```
M ~/esp
sandr@DESKTOP-24QNIMS MINGW32 ~
$ mkdir esp

sandr@DESKTOP-24QNIMS MINGW32 ~
$ cd esp

sandr@DESKTOP-24QNIMS MINGW32 ~/esp
$ pwd
/home/sandr/esp

sandr@DESKTOP-24QNIMS MINGW32 ~/esp
$ git clone --recursive https://github.com/espressif/esp-idf.git
Cloning into 'esp-idf'...
remote: Counting objects: 44175, done.
remote: Compressing objects: 100% (246/246), done.
Receiving objects: 30% (13253/44175), 25.61 MiB | 787.00 KiB/s
```

Configurando o caminho para ESP-IDF

Após o download da ESP-IDF na pasta `~/esp/esp-idf`, será necessário configurar o caminho (path). Os programas do toolchain acessam o ESP-IDF usando a variável de ambiente `IDF_PATH`. Os scripts do perfil de usuário estão contidos na pasta `~/msys32/etc/profile.d/`. Eles são executados toda vez que você abre uma janela MSYS2.

- Crie um novo arquivo de script na pasta `~/msys32/etc/profile.d/`. O arquivo deverá se chamar `export_idf_path.sh`.
- Identifique (com o Gerenciador de arquivos) o caminho para o diretório `esp-idf`. É específico para a configuração do seu sistema e pode parecer algo, como por exemplo : `C:\msys32\home\sandr\esp\esp-idf`

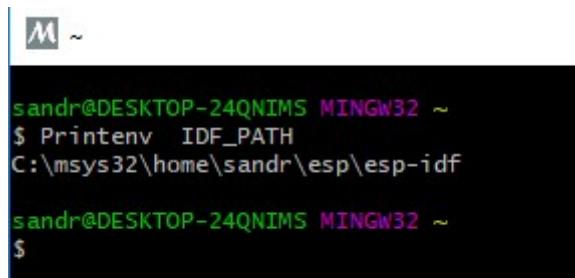
Usando qualquer editor (no meu caso, usei NotePad++), digite o comando `export` abaixo no *script* do arquivo, por exemplo:

```
export IDF_PATH="C:\msys32\home\sandr\esp\esp-idf"
```

- Lembre-se de substituir por barras normais no Path do Windows. Salve o arquivo do script como `export_idf_path.sh`
- Feche a janela MSYS2 e abra-a novamente. Verifique se `IDF_PATH` está configurado corretamente , digitando:

Printenv IDF_PATH

O caminho previamente inserido no arquivo de script deverá ser impresso.



```
sandr@DESKTOP-24QNIMS MINGW32 ~
$ Printenv IDF_PATH
C:\msys32\home\sandr\esp\esp-idf

sandr@DESKTOP-24QNIMS MINGW32 ~
$
```

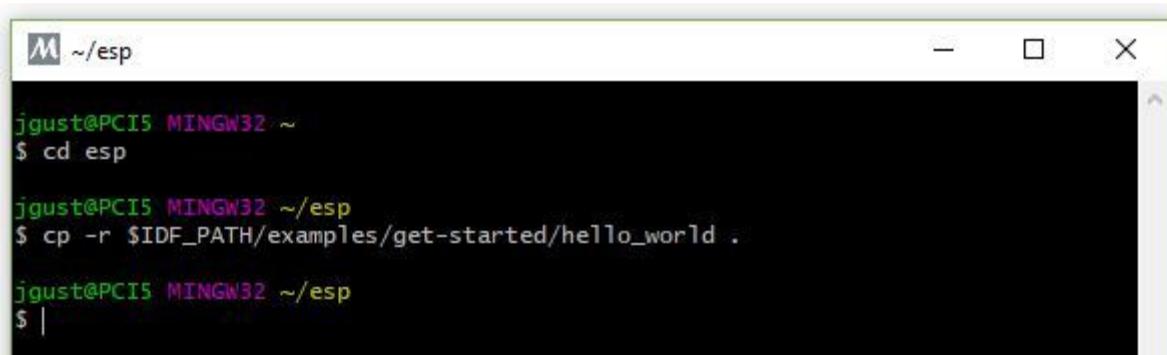
Testando um Projeto exemplo

O ambiente no seu PC, está pronto para testar uma aplicação para o ESP32. O Programa ***get-started/hello_world*** da pasta de examples da ESP-IDF, será usado para teste.

Copie a pasta do exemplo ***hello_word*** para a pasta ***~/esp***, com o comando abaixo usando a janela do **mingw32**. (não se esqueça do ponto no final do comando)

```
cd esp
```

```
cp -r $IDF_PATH/examples/get-started/hello_world .
```



```
jgust@PCI5 MINGW32 ~
$ cd esp

jgust@PCI5 MINGW32 ~/esp
$ cp -r $IDF_PATH/examples/get-started/hello_world .

jgust@PCI5 MINGW32 ~/esp
$ |
```

Configurando a porta COM na ESP-IDF

Conecte o seu ESP32 na porta USB do seu PC. Identifique a porta COM usada. O procedimento é o mesmo deste tutorial :

Configurando a ARDUINO IDE p/ o ESP32 :

Abra a janela do programa **mingw32**, e dê os seguintes comandos para configurar a porta COM na ESP-IDF:

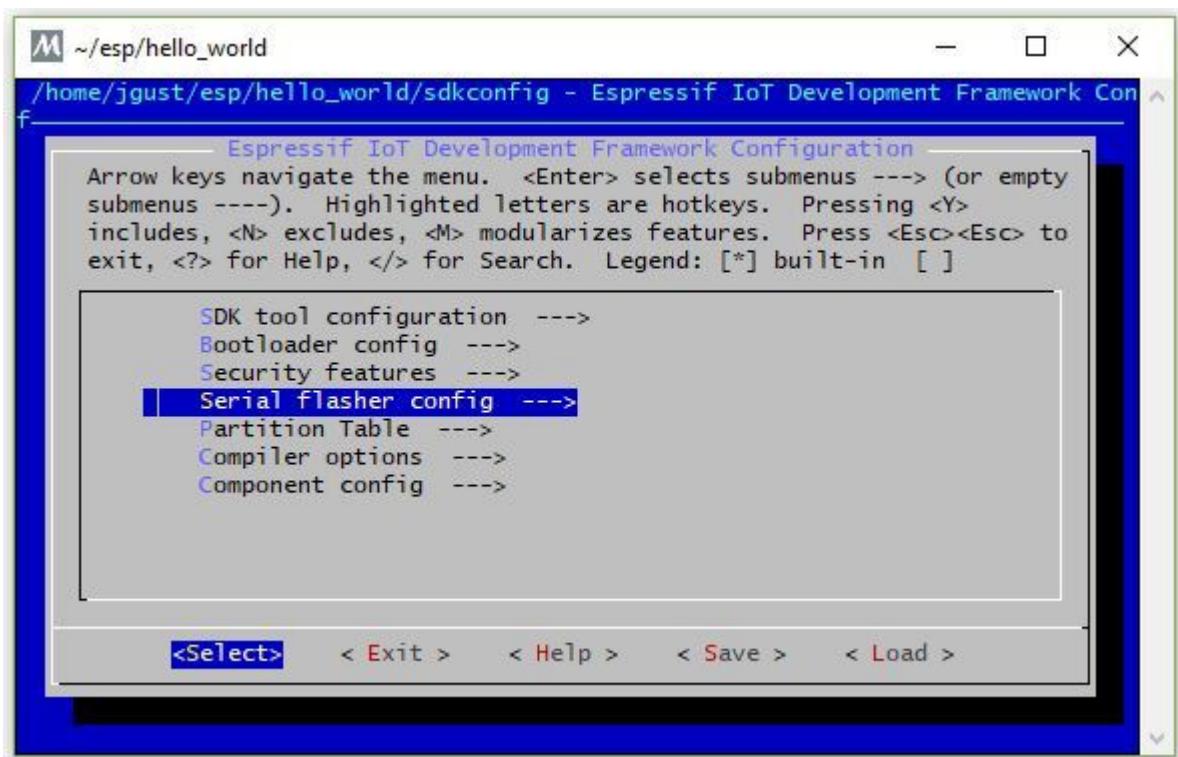
```
cd ~/esp/hello_world
```

```
make menuconfig
```

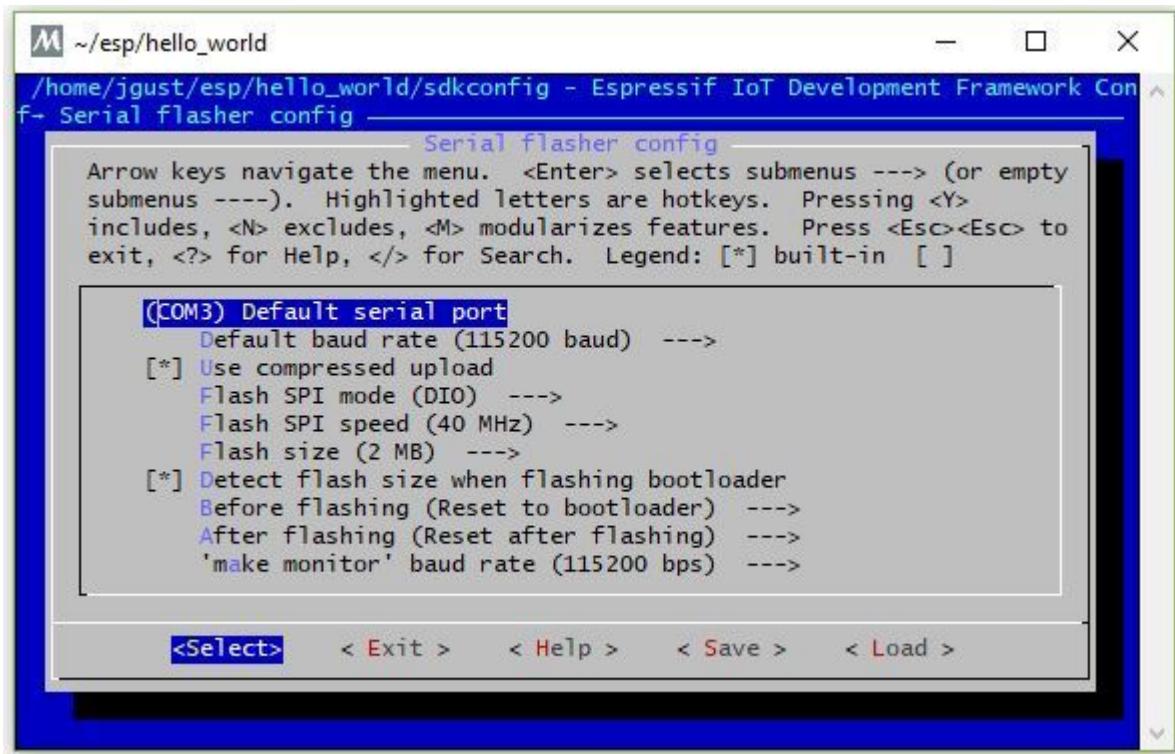


```
M ~/esp/hello_world
jgust@PC15 MINGW32 ~
$ cd ~/esp/hello_world
jgust@PC15 MINGW32 ~/esp/hello_world
$ make menuconfig|
```

Uma outra janela de configuração deverá aparecer . Usando as **teclas de cursor** e a tecla **Enter** , selecione **Serial flasher config** :



Configure a porta COM do seu PC para a interface serial-USB do seu ESP32 : (edite o campo)



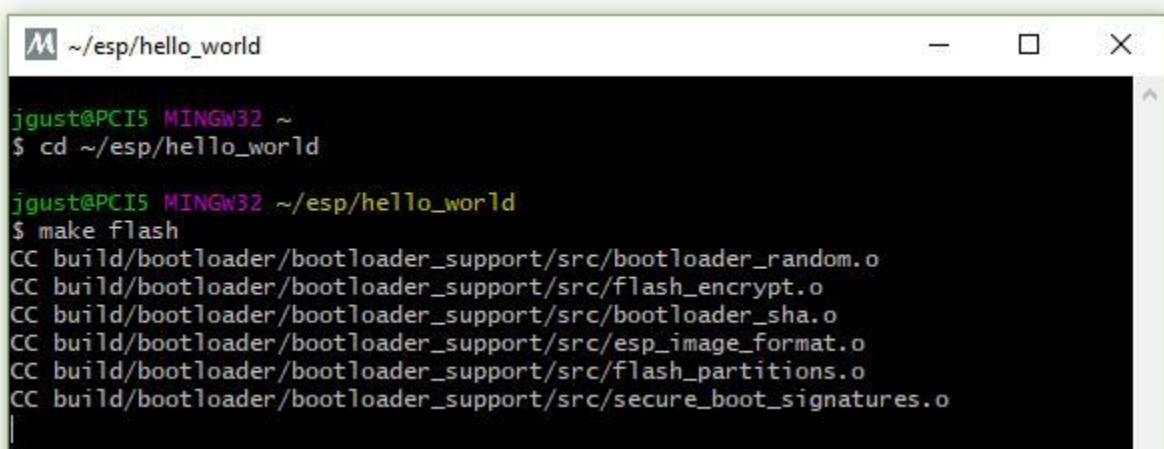
Depois selecione **Exit** duas vezes e **Salve** a configuração .

Compilando e Gravando na Flash do ESP32

Mantenha a placa ESP32 conectada no seu PC. Abra a janela do **mingw32**. Para compilar o aplicativo e todos os componentes do ESP-IDF, gerar o bootloader, a tabela de partição, os binários do aplicativo e gravar esses arquivos na memória Flash da placa ESP32, dê os comandos abaixo.

```
cd ~/esp/hello_world
```

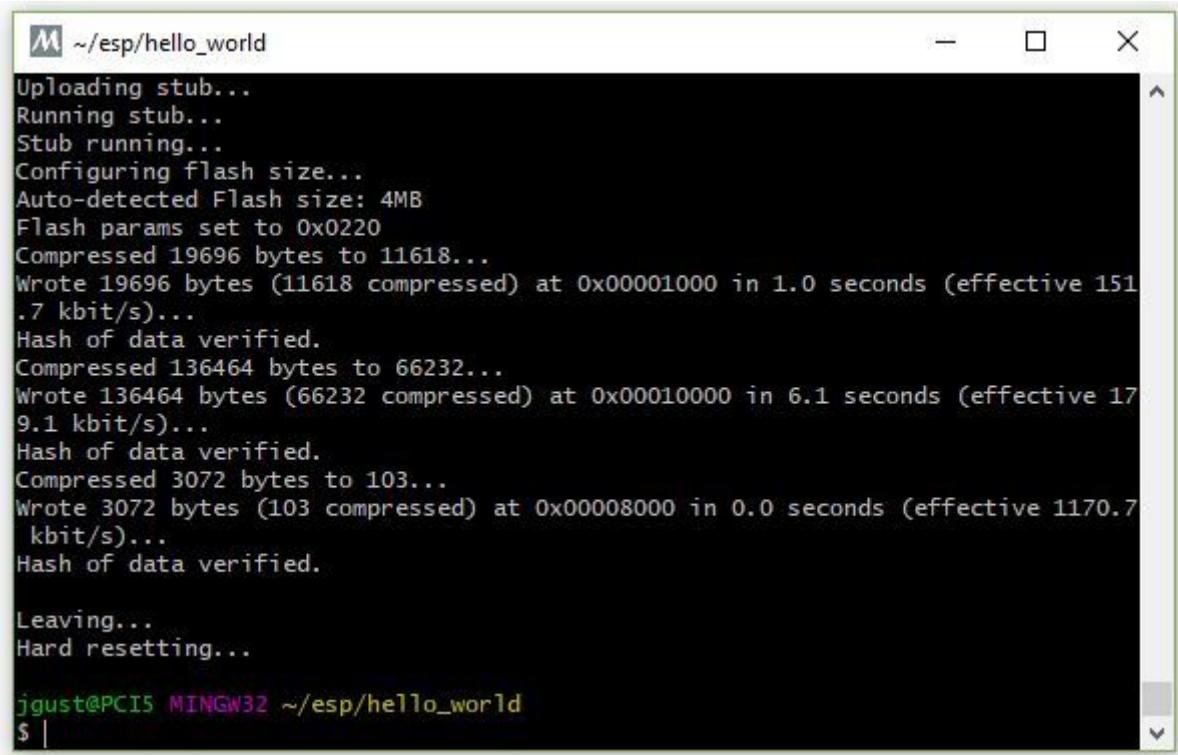
```
make flash
```



```
jgust@PC15 MINGW32 ~
$ cd ~/esp/hello_world

jgust@PC15 MINGW32 ~/esp/hello_world
$ make flash
CC build/bootloader/bootloader_support/src/bootloader_random.o
CC build/bootloader/bootloader_support/src/flash_encrypt.o
CC build/bootloader/bootloader_support/src/bootloader_sha.o
CC build/bootloader/bootloader_support/src/esp_image_format.o
CC build/bootloader/bootloader_support/src/flash_partitions.o
CC build/bootloader/bootloader_support/src/secure_boot_signatures.o
```

Após alguns minutos (demora mesmo) e se tudo correr bem , deverá aparecer essas mensagens abaixo. A placa será resetada e a aplicação **hello_world** começará.



```
Uploading stub...
Running stub...
Stub running...
Configuring flash size...
Auto-detected Flash size: 4MB
Flash params set to 0x0220
Compressed 19696 bytes to 11618...
Wrote 19696 bytes (11618 compressed) at 0x00001000 in 1.0 seconds (effective 151.7 kbit/s)...
Hash of data verified.
Compressed 136464 bytes to 66232...
Wrote 136464 bytes (66232 compressed) at 0x00010000 in 6.1 seconds (effective 179.1 kbit/s)...
Hash of data verified.
Compressed 3072 bytes to 103...
Wrote 3072 bytes (103 compressed) at 0x00008000 in 0.0 seconds (effective 1170.7 kbit/s)...
Hash of data verified.

Leaving...
Hard resetting...
```

Pronto, a aplicação já está gravada na Flash do ESP32. Agora vamos acessar a porta serial-USB , através de uma console (pode usar um outro programa – velocidade 115200 Bps). Na janela do **mingw32**, dê o comando para abrir a console da IDF :

make monitor

```
jgust@PC15 MINGW32 ~/esp/hello_world
$ make monitor|
```

Essa aplicação Hello_world, fica em loop enviando a mensagem Hello world ! e depois dá um reboot. Somente isso . Parabéns ! Você conseguiu usar a ESP-IDF com o ESP32 .

```
0x400d0018: _flash_cache_start at ???:?

I (181) boot: Loaded app from partition at offset 0x10000
I (181) boot: Disabling RNG early entropy source...
I (181) cpu_start: Pro cpu up.
I (185) cpu_start: Starting app cpu, entry point is 0x40080e24
0x40080e24: call_start_cpu1 at C:/gustavo/msys32/home/jgust/esp/esp-idf/components/esp32/cpu_start.c:215

I (177) cpu_start: App cpu up.
I (196) heap_init: Initializing. RAM available for dynamic allocation:
I (203) heap_init: At 3FFAE6E0 len 00001920 (6 KiB): DRAM
I (209) heap_init: At 3FFB2990 len 0002D670 (181 KiB): DRAM
I (215) heap_init: At 3FFE0440 len 00003BC0 (14 KiB): D/IRAM
I (221) heap_init: At 3FFE4350 len 0001BCB0 (111 KiB): D/IRAM
I (228) heap_init: At 400888E0 len 00017720 (93 KiB): IRAM
I (234) cpu_start: Pro ccpu start user code
I (252) cpu_start: Starting scheduler on PRO CPU.
I (0) cpu_start: Starting scheduler on APP CPU.
Hello world!
This is ESP32 chip with 2 CPU cores, WiFi/BT/BLE, silicon revision 1, 4MB external flash
Restarting in 10 seconds...|
```

Vejam o help do comando make :

\$ make help

REFERÊNCIAS:

Mjrovai. (2017) “Playing with the esp32 on arduino IDE”, <http://www.instructables.com/id/IOT-Made-Simple-Playing-With-the-ESP32-on-Arduino-/>

Minatel, P. (2016) “Arquivos sobre ESp8266”, <http://pedrominatel.com.br> .

Embarcados. (2016) “Módulo ESP8266”, <https://www.embarcados.com.br/modulo-esp8266> .

Do bit ao byte (2016) <http://dobitaobyte.com.br/timer-com-esp8266-na-ide-do-arduino>

Esp8266 Libraries. <https://github.com/esp8266/Arduino/tree/master/libraries/Ticker>