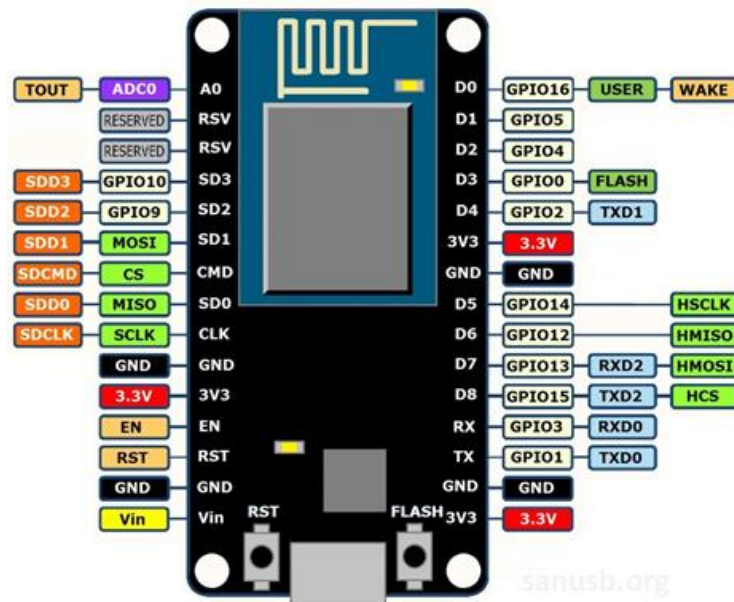
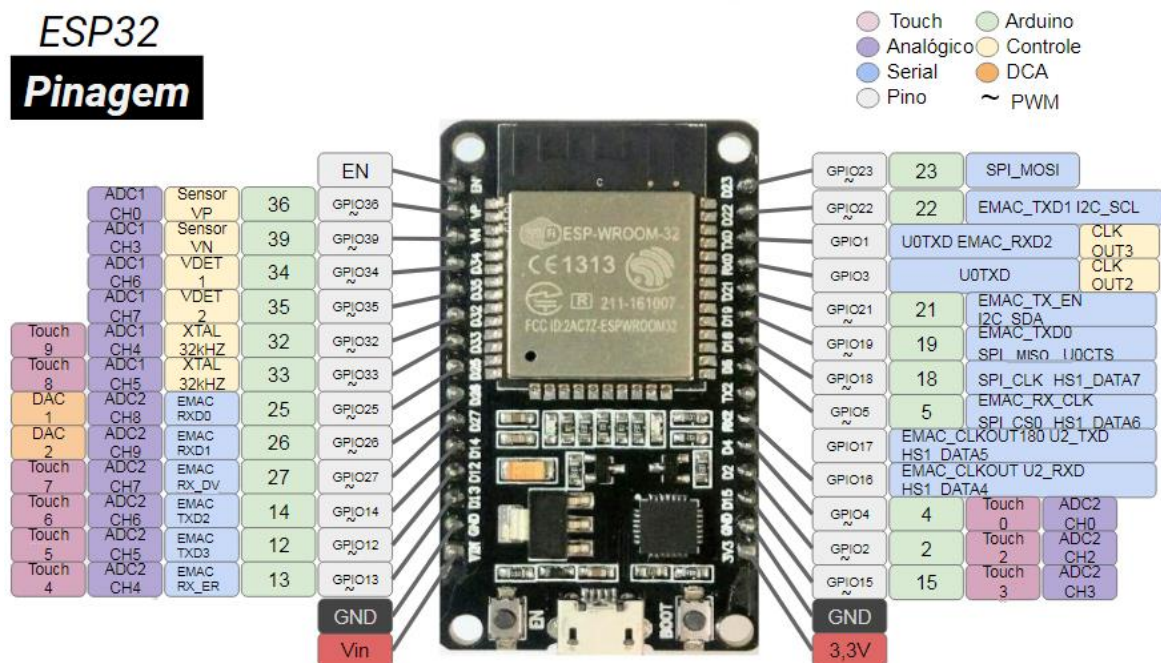


Aplicações Práticas de ESP8266 e ESP32 em IoT



ESP32

Pinagem



SanUSB.org

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

Sumário

1. INTRODUÇÃO	4
1.1. Instalação da placa ESP32.....	7
Hardware do módulo ESP8266.....	15
Integrando o ESP à IDE Arduino.....	16
Piscando o LED	19
ATUALIZAÇÃO WIFI PELO BROWSER FUNCIONAL	41
Instalando a IDE platformIO	56
Memória SPIFFS:.....	70
Como utilizar.....	71
INTRODUÇÃO A IoT COM MQTT	75
Caso prático de uso:	77
COMO RESTAURAR O FIRMWARE ORIGINAL COM COMANDOS AT	106
SERVIDOR IOT PUSH	111
Conexão via Putty.....	111
Instalação do Toolchain esp32 para Windows.....	122
.....	123
Configurando o caminho para ESP-IDF	123
Testando um Projeto exemplo	124
Configurando a porta COM na ESP-IDF.....	125
Compilando e Gravando na Flash do ESP32	127

INTRODUÇÃO

Para O ESP32 é um microcontrolador altamente versátil, amplamente utilizado em projetos de IoT (Internet das Coisas), automação e sistemas embarcados. Desenvolvido pela Espressif Systems, ele combina processamento, conectividade e baixo consumo de energia, sendo uma evolução do ESP8266, oferecendo melhorias significativas tanto em performance quanto em funcionalidades.

Características Técnicas

Processador:

O ESP32 possui um ou dois núcleos de CPU Xtensa LX6, com frequências que variam de 160 MHz a 240 MHz, dependendo do modelo. Isso proporciona uma boa capacidade de processamento para aplicações embarcadas e de IoT.

Conta com uma unidade de ponto flutuante (FPU) para cálculos mais precisos e uma unidade de aceleração criptográfica para garantir segurança nas comunicações.

Memória:

O ESP32 geralmente possui 520 KB de RAM SRAM integrada. Além disso, pode contar com uma quantidade variável de memória flash (normalmente entre 4 MB e 16 MB), que é usada para armazenar o firmware e outros dados do programa.

Conectividade:

Wi-Fi: Um dos principais diferenciais do ESP32 é o suporte a Wi-Fi 802.11 b/g/n, que possibilita a comunicação com redes sem fio, tornando-o ideal para aplicações IoT.

Bluetooth: Outro destaque é o suporte a Bluetooth 4.2 e Bluetooth Low Energy (BLE), o que permite a integração com uma ampla gama de dispositivos Bluetooth, como sensores e smartphones.

Comunicação Serial: Além disso, ele possui interfaces padrão de comunicação, como UART, I2C, SPI, I2S, PWM, e CAN, que facilitam a integração com sensores, atuadores e outros dispositivos eletrônicos.

Baixo Consumo de Energia:

O ESP32 oferece modos de economia de energia, como o modo de sono profundo (deep sleep) e o modo de hibernação, sendo capaz de operar com uma corrente extremamente baixa, o que o torna ideal para aplicações movidas a bateria.

Pinos de Entrada/Saída (GPIO):

Com cerca de 34 pinos de GPIO configuráveis, o ESP32 pode ser usado para leitura de sensores, controle de motores, LEDs, e uma variedade de dispositivos. Muitos desses pinos são multifuncionais, permitindo que o desenvolvedor escolha a função desejada para cada um deles (pino digital, PWM, etc.).

Sensor Hall e Touch Capacitivo:

Alguns modelos possuem um sensor de campo magnético Hall integrado e entradas capacitivas, permitindo a construção de interfaces sensíveis ao toque sem a necessidade de hardware adicional.

Aplicações Práticas do ESP32

Internet das Coisas (IoT):

A capacidade de conectar-se diretamente à internet via Wi-Fi torna o ESP32 ideal para aplicações de IoT, como automação residencial, controle de dispositivos inteligentes, monitoramento de sensores remotos e sistemas de coleta de dados.

Exemplos incluem: sensores de temperatura e umidade que enviam dados para a nuvem, sistemas de irrigação inteligentes controlados remotamente, e dispositivos wearable (vestíveis) com conectividade Bluetooth ou Wi-Fi.

Automação Residencial e Industrial:

Graças ao suporte a vários protocolos de comunicação, o ESP32 é amplamente utilizado em projetos de automação para controle de lâmpadas, climatização, monitoramento de consumo de energia, e acionamento de dispositivos elétricos, seja localmente ou via internet.

Controle de Dispositivos com Bluetooth:

O ESP32 também pode ser usado para criar sistemas que se integram a dispositivos móveis via Bluetooth, como controles de robôs, fechaduras eletrônicas ou dispositivos de fitness que sincronizam dados com smartphones.

Aplicações com Baixo Consumo de Energia:

Para projetos em que a economia de energia é crítica, como sensores em ambientes externos ou wearables, o ESP32 é ideal, pois pode permanecer em modo de "deep sleep" por longos períodos, despertando apenas para enviar ou receber dados.

Sistemas Embarcados e Automação de Pequenos Equipamentos:

Projetos que envolvem controle de motores, displays, leitura de múltiplos sensores ou execução de tarefas específicas de forma autônoma podem ser geridos eficientemente pelo ESP32, aproveitando sua capacidade de processamento e conectividade.

Redes Mesh e Comunicação entre Dispositivos:

O ESP32 suporta redes Mesh, onde múltiplos dispositivos ESP32 podem se conectar entre si para criar uma rede distribuída. Isso é útil para cenários em que se precisa cobrir uma grande área, como em sistemas de iluminação pública ou monitoramento ambiental em larga escala.

Vantagens do ESP32:

Custo-benefício: Apesar de suas muitas funcionalidades, o ESP32 é um microcontrolador de baixo custo, o que o torna uma excelente escolha para projetos que precisam de conectividade e processamento de dados a um preço acessível.

Comunidade e Suporte: O ESP32 tem uma vasta comunidade de desenvolvedores, o que facilita encontrar exemplos de código, bibliotecas e documentação. Além disso, ele é compatível com a plataforma Arduino, simplificando ainda mais o desenvolvimento de projetos.

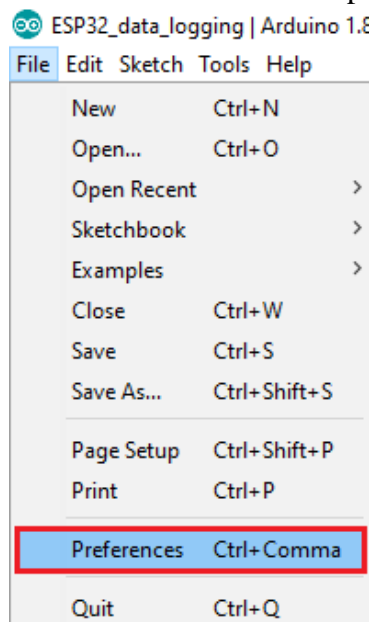
Flexibilidade de Programação: Além de ser suportado na Arduino IDE e PlatformIo, o ESP32 também pode ser programado utilizando outras plataformas como o Espressif IoT Development Framework (ESP-IDF), MicroPython e FreeRTOS.

Em resumo, o ESP32 se destaca por ser um microcontrolador multifuncional, eficiente e acessível, com uma combinação única de conectividade, capacidade de processamento e baixo consumo energético, aplicável em uma vasta gama de projetos tecnológicos e inovadores.

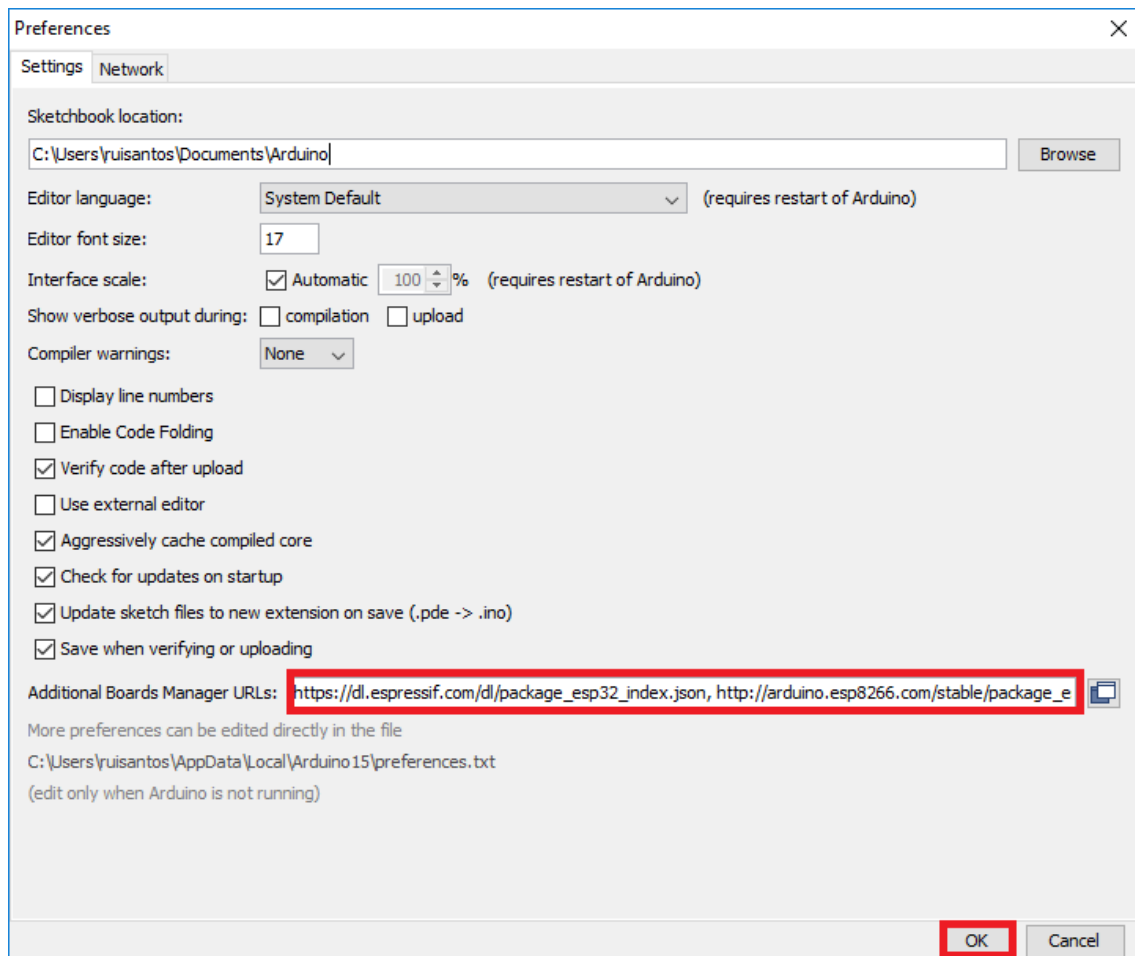
Instalação da placa ESP32

Para instalar a placa ESP32 no seu Arduino IDE, siga estas instruções:

- 1) Abra a janela de preferências no IDE do Arduino. Vá para Arquivo > **Preferências**.



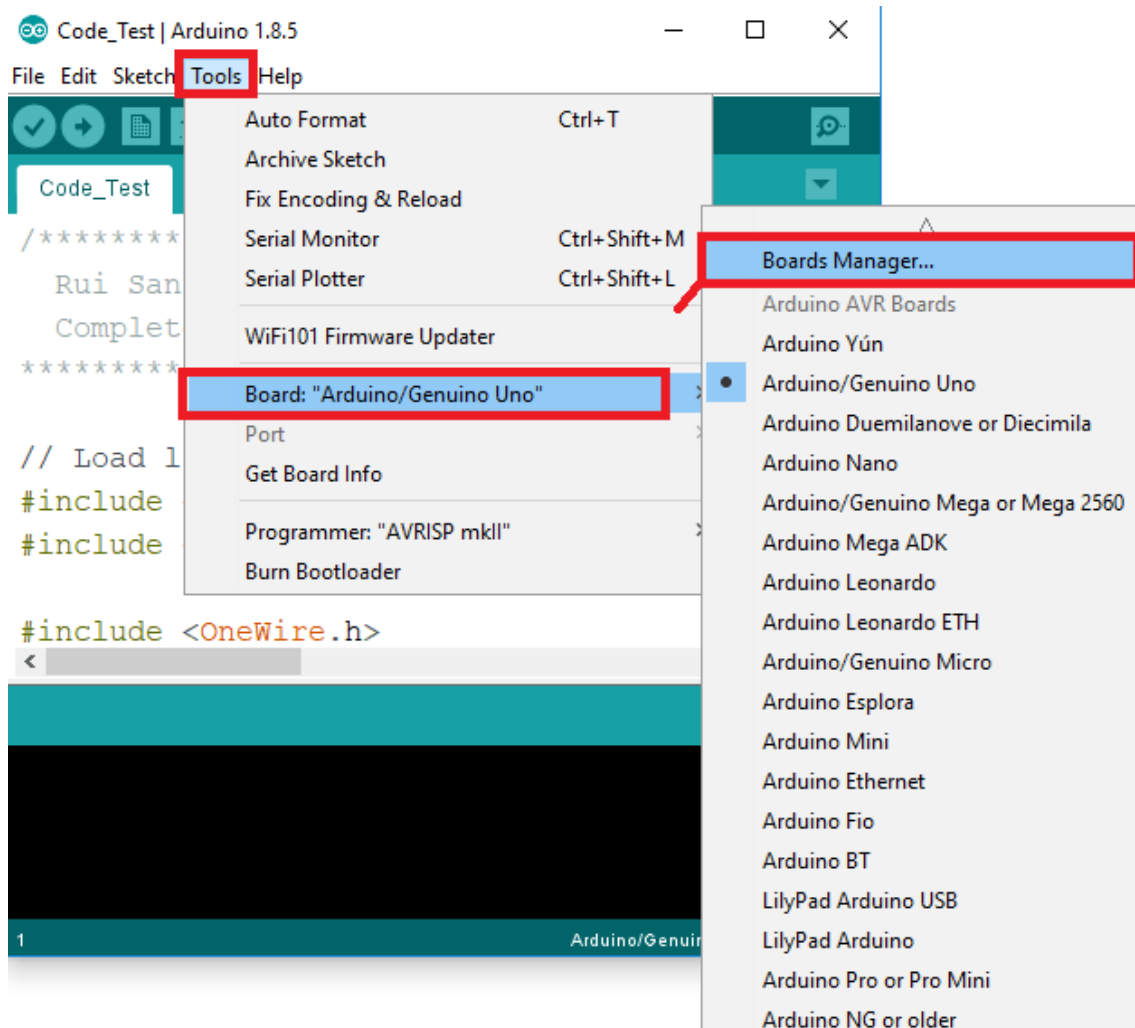
- 2) Verifique a biblioteca https://dl.espressif.com/dl/package_esp32_index.json no campo "Additional Board Manager URLs", como mostra a figura abaixo. Em seguida, clique no botão "OK":



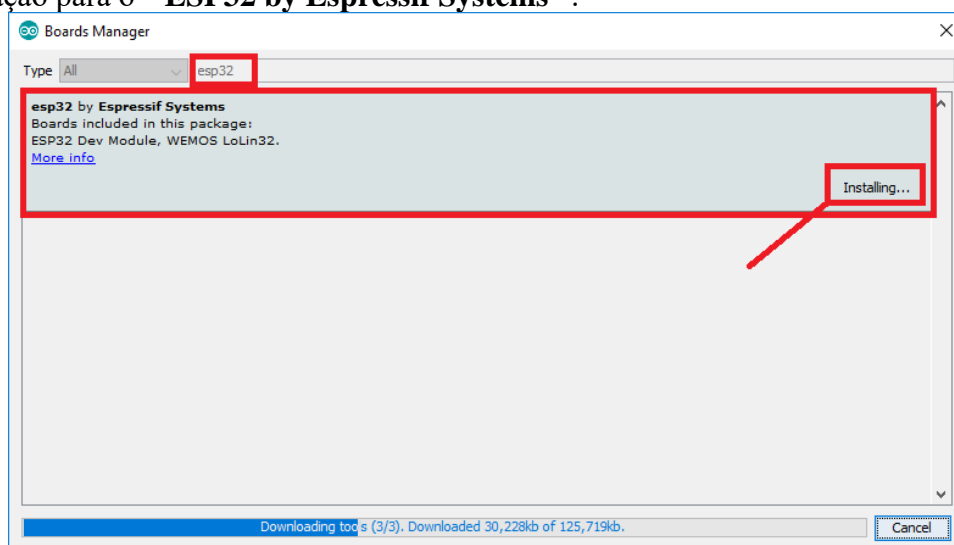
Nota: se você já tem o URL das placas ESP8266, você pode separar as URLs com uma vírgula da seguinte forma:

https://dl.espressif.com/dl/package_esp32_index.json,
http://arduino.esp8266.com/stable/package_esp8266com_index.json

3) Gerenciador de placas abertas. Vá para **Ferramentas > Placa > Gerenciador de Placas ...**



4) Se ainda não tiver instalado o ESP32, procure por ESP32 e pressione o botão de instalação para o “**ESP32 by Espressif Systems**”:

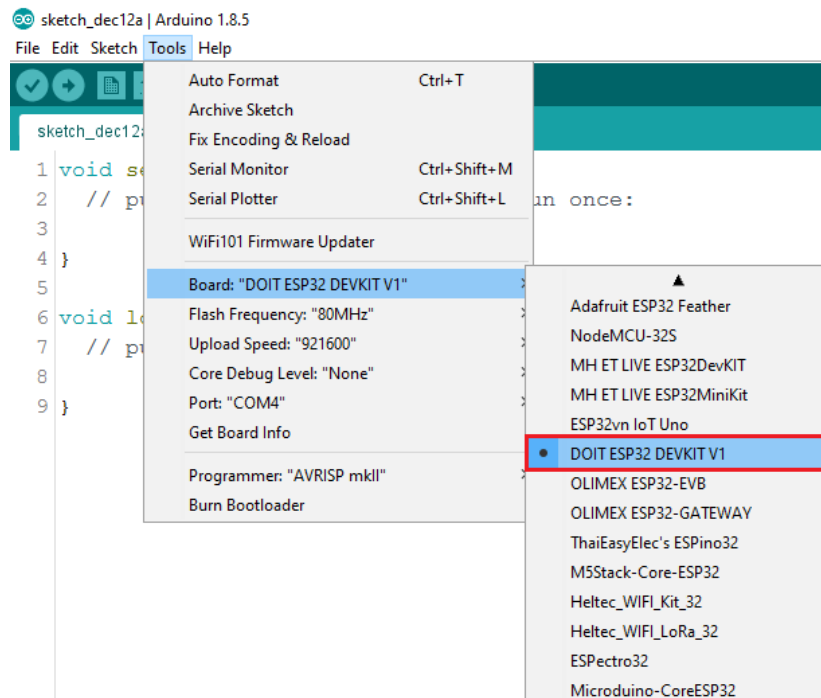


5) É isso aí. Deve ser instalado após alguns segundos.

Testando a instalação:

Conecte a placa ESP32 ao seu computador. Em seguida, siga estas etapas:

- 1) Abra o Arduino IDE
- 2) Selecione sua placa em **Ferramentas** > menu **Board** (no meu caso é o **DOIT ESP32 DEVKIT V1**).



- 3) Selecione a porta (se você não vê a porta COM no seu Arduino IDE, você precisa instalar o USB ESP32 CP210x para UART Bridge VCP Drivers);
- 4) Cole o seguinte código abaixo para comutar o acionamento do led interno no pino 2.

```
pinMode(2, OUTPUT);
}

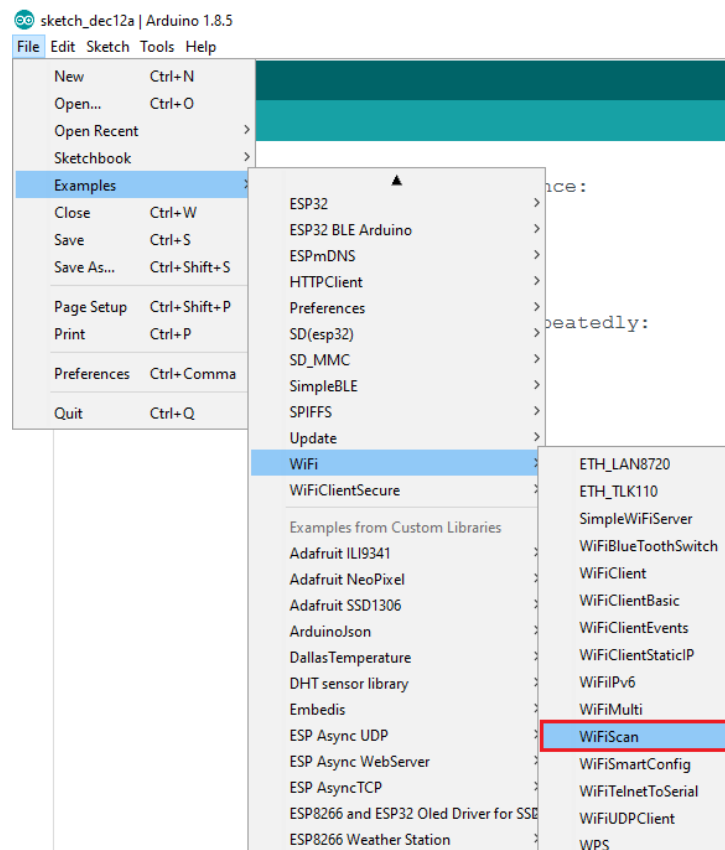
void loop()
{
  digitalWrite(2, HIGH); /* LED Liga */
  delay(1000);          /* Espera 1 segundo */
  digitalWrite(2, LOW); /* LED Desliga */
  delay(1000);          /* Espera 1 segundo */
}
```

O exemplo transferido terá como função piscar o LED D2 que encontra-se embutido na própria placa. A programação escreve-se similar ao padrão comum da linguagem C, declarando os componentes como entradas (INPUT) ou saídas (OUTPUT), controlando o nível lógico com HIGH-1 ou LOW-0 (Liga e desliga) com pausas com

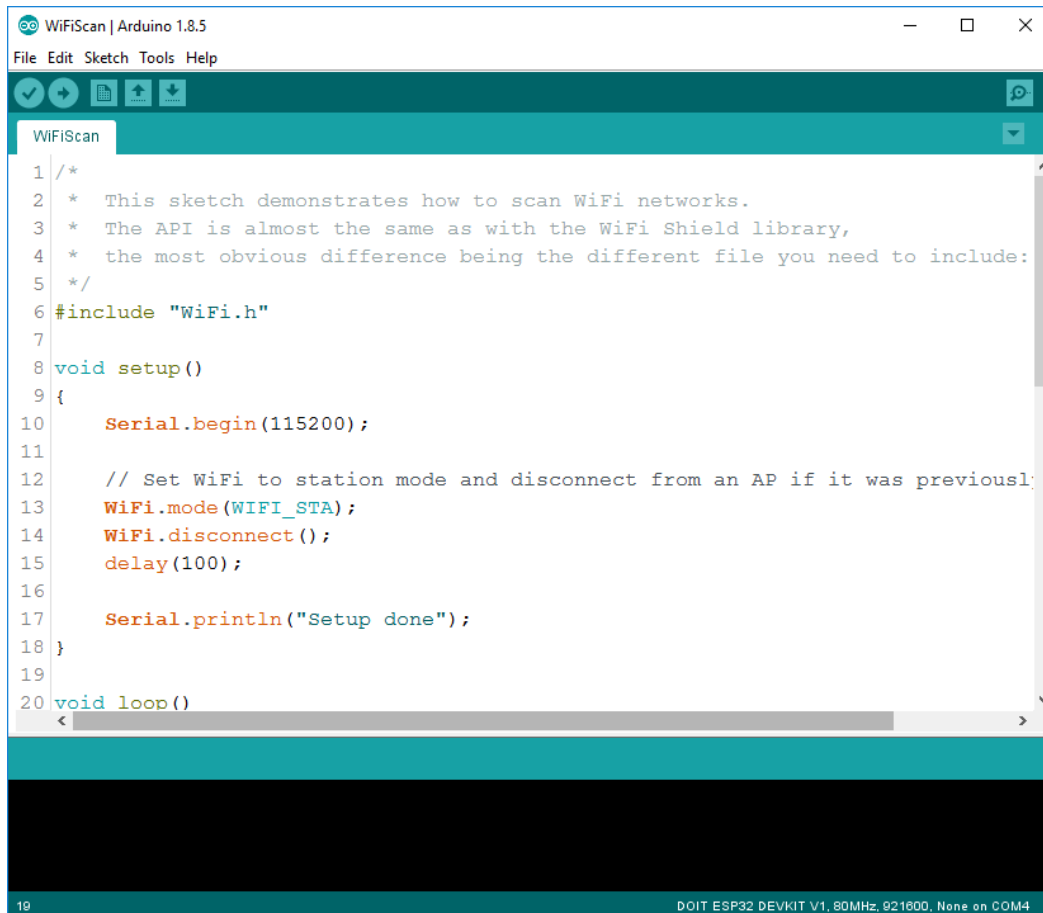
delay em milissegundos (1 segundo = 1000 milissegundos). Em algumas situações, para transferir um código para o microcontrolador, é necessário pressionar o botão BOOT quando a mensagem “Connecting” surgir, ou quando a barra de progresso estiver cheia. Mais detalhes em <https://www.crescerengenharia.com/post/instalando-esp32-arduino>.



5) Para testar a conexão WiFi, abra o seguinte exemplo em **Arquivo > Exemplos > WiFi (ESP32) > WiFi Scan**.



6) Um novo esboço é aberto:

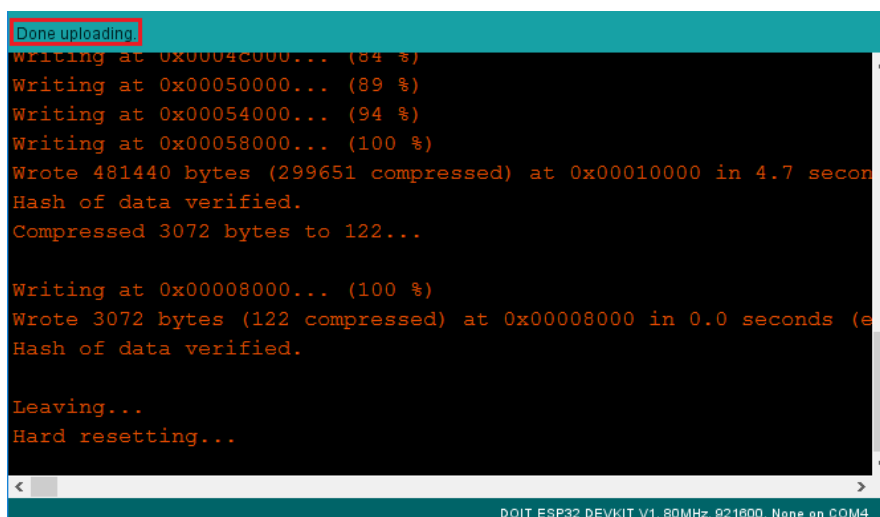


```

1  /*
2  *   This sketch demonstrates how to scan WiFi networks.
3  *   The API is almost the same as with the WiFi Shield library,
4  *   the most obvious difference being the different file you need to include:
5  */
6  #include "WiFi.h"
7
8  void setup()
9  {
10     Serial.begin(115200);
11
12     // Set WiFi to station mode and disconnect from an AP if it was previously
13     // connected
14     WiFi.mode(WIFI_STA);
15     WiFi.disconnect();
16     delay(100);
17
18     Serial.println("Setup done");
19 }
20 void loop()
  
```

7) Pressione o botão **Upload** no Arduino IDE. Aguarde alguns segundos enquanto o código é compilado e enviado para a sua placa.

8) Se tudo correu como esperado, você deve ver uma mensagem " **Done uploading**".



```

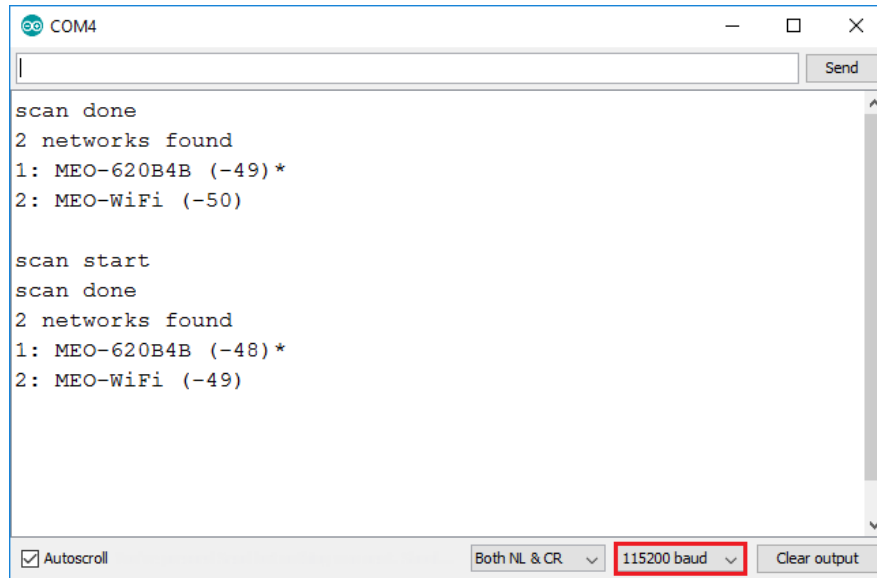
Done uploading.
Writing at 0x00040000... (84 %)
Writing at 0x00050000... (89 %)
Writing at 0x00054000... (94 %)
Writing at 0x00058000... (100 %)
Wrote 481440 bytes (299651 compressed) at 0x00010000 in 4.7 seconds
Hash of data verified.
Compressed 3072 bytes to 122...

Writing at 0x00008000... (100 %)
Wrote 3072 bytes (122 compressed) at 0x00008000 in 0.0 seconds (effective 115200 bps)
Hash of data verified.

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

8) Abra o Monitor Serial do Arduino IDE com uma taxa de transmissão de 115200;

9) Pressione o botão de **ativação da** placa ESP32 e você verá as redes disponíveis perto do seu ESP32:



OS MODELOS ESP8266 NODEMCU

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: $\approx 70\text{mA}$ (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 dupla ponte 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 a 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.

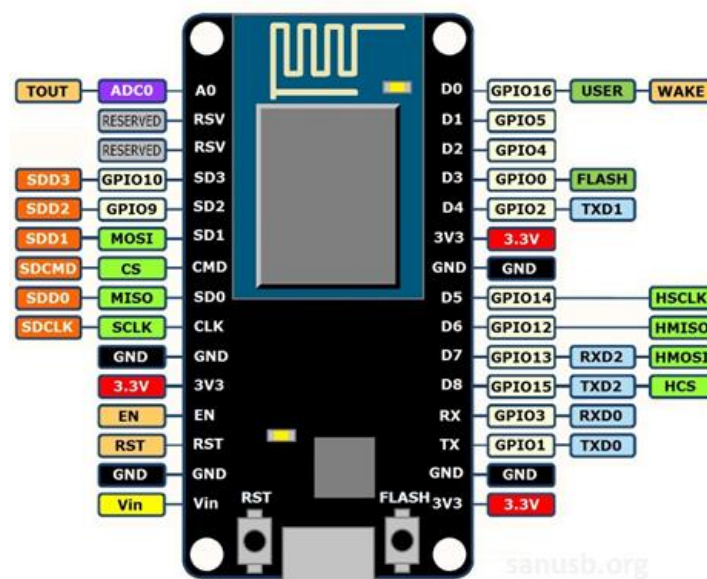


Figura 1: Placa ESP8266 NodeMCU

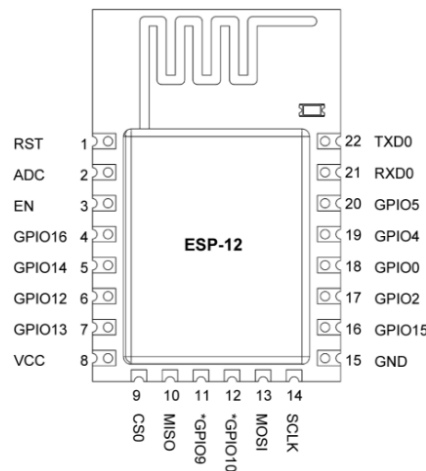


Figura 2: Chip ESP8266

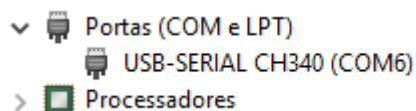
Essa placa possui suporta funções como PWM, I2C e 1-wire. Tem antena embutida, conversor USB-TTL integrado e o seu formato é ideal para ambientes de prototipação, encaixando facilmente em um *protoboard*.

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:



Integrando o ESP à IDE Arduino

Para instalar as placas ESP32 e ESP8266 no Arduino IDE, siga as próximas instruções: Na IDE do Arduino, vá em Arquivo > Preferências. Insira nos "URLs adicionais do gerenciador de placa":

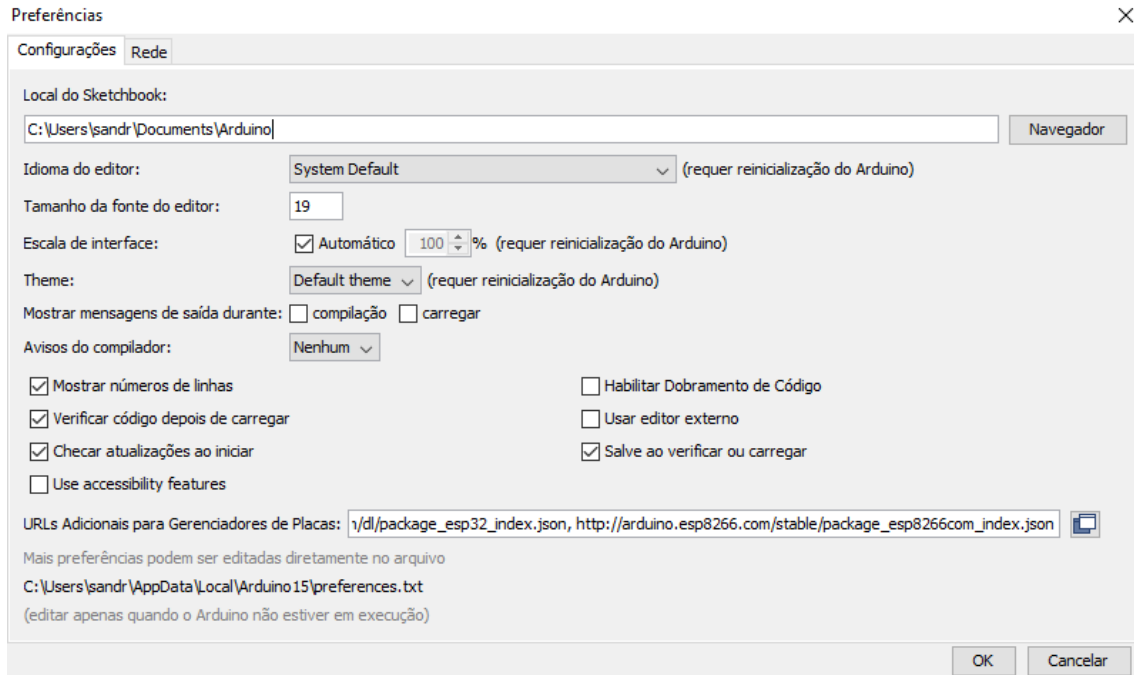
https://dl.espressif.com/dl/package_esp32_index.json,
http://arduino.esp8266.com/stable/package_esp8266com_index.json

Abra o Boards Manager: Vá em Ferramentas > Placas > Gerenciador de Placas: Procure por ESP32 e pressione o botão de instalação para o "*ESP32 by Espressif Systems*". Procure também por ESP8266 e pressione o botão de instalação para o "*ESP8266 by ESP8266 Community*".

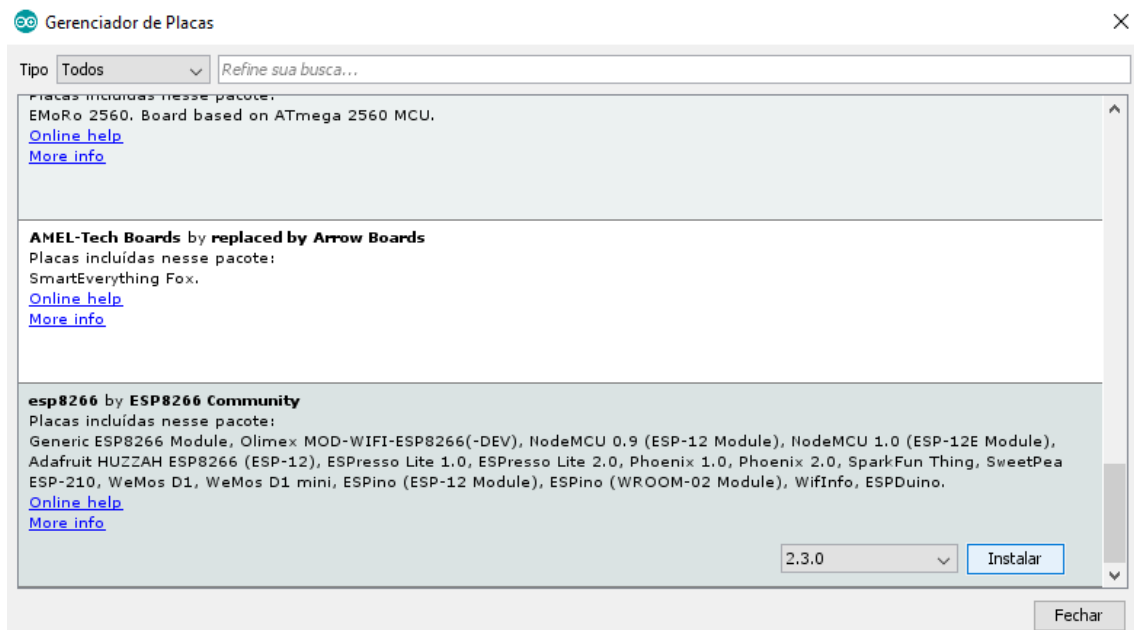
Para programar e usar o NodeMCU como se fosse um Arduino, a boa notícia é que é possível escrever-se firmwares personalizados e carregá-los no chip ("flash-it"). É importante lembrar que qualquer novo "firmware personalizado" irá substituir qualquer coisa previamente armazenada na memória flash do chip, 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



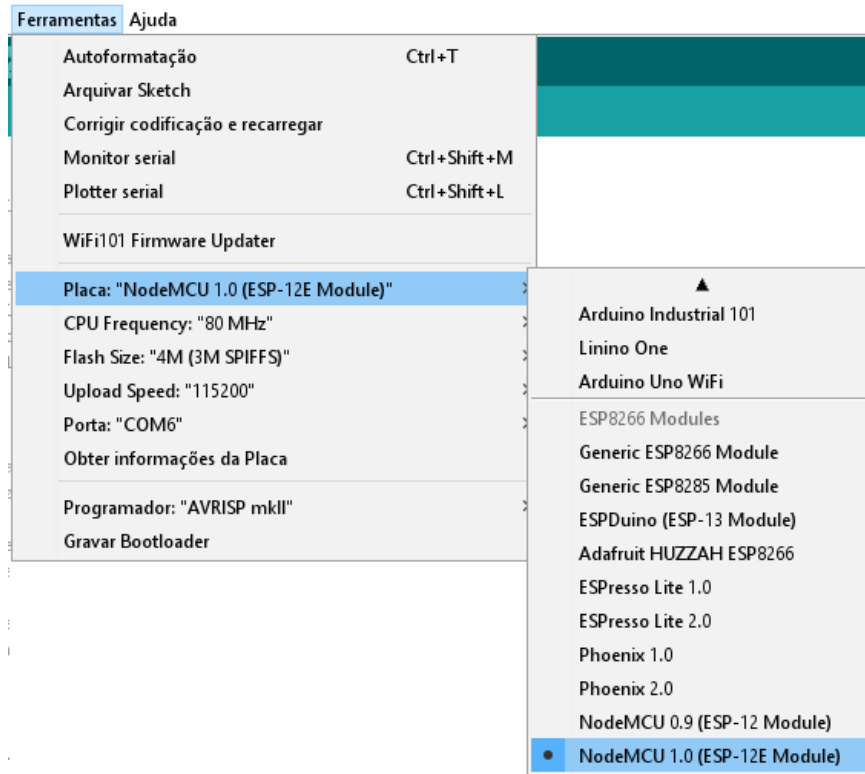
É importante salientar que todas as bibliotecas instaladas ficam disponíveis em um diretório convencional como em `~\Documents\Arduino\Libraries`.



Em relação ao ESP8266, 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 ou DOIT ESP32 Devkit1**. Em seguida, especificar a frequência de operação da CPU: (**Ferramentas** → **CPU Frequency**: “” → **80MHz**) e velocidade

de comunicação (**Ferramentas** → **Upload Speed:** “” → **115,200**). Finalmente, selecionar a porta serial virtual emulada no 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*.

1.2. FUNÇÕES BÁSICAS DOS PINOS DO ESP

Funções como *PinMode()*, *digitalRead()*, *digitalWrite()*, *analogWrite()* operam normalmente. Os números dos pinos correspondem diretamente aos números do GPIO. 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.

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

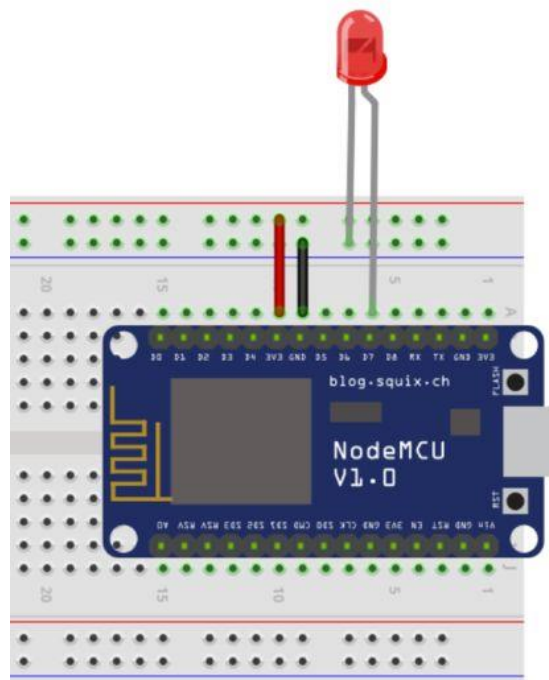
O comando `AnalogWrite (pino, valor)` permite implementar o software PWM no pino especificado. O PWM pode ser usado nos pinos 0 a 15. Chame `analogWrite (pino, 0)` para desabilitar o PWM no pino. O valor pode estar no intervalo de 0 a 1023.

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, lientecamente identificados pelo IDE, tal como descrito abaixo:

Placa ESP8266 → 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 = "*****";

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 cliente = server.available();
  if (!cliente) {
    return;
  }

  Serial.println("new cliente");
  while(!cliente.available()){
    delay(1);
  }

  String req = cliente.readStringUntil('\r');
  Serial.println(req);
  cliente.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>\r\n";

  cliente.print(buf);
  cliente.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);

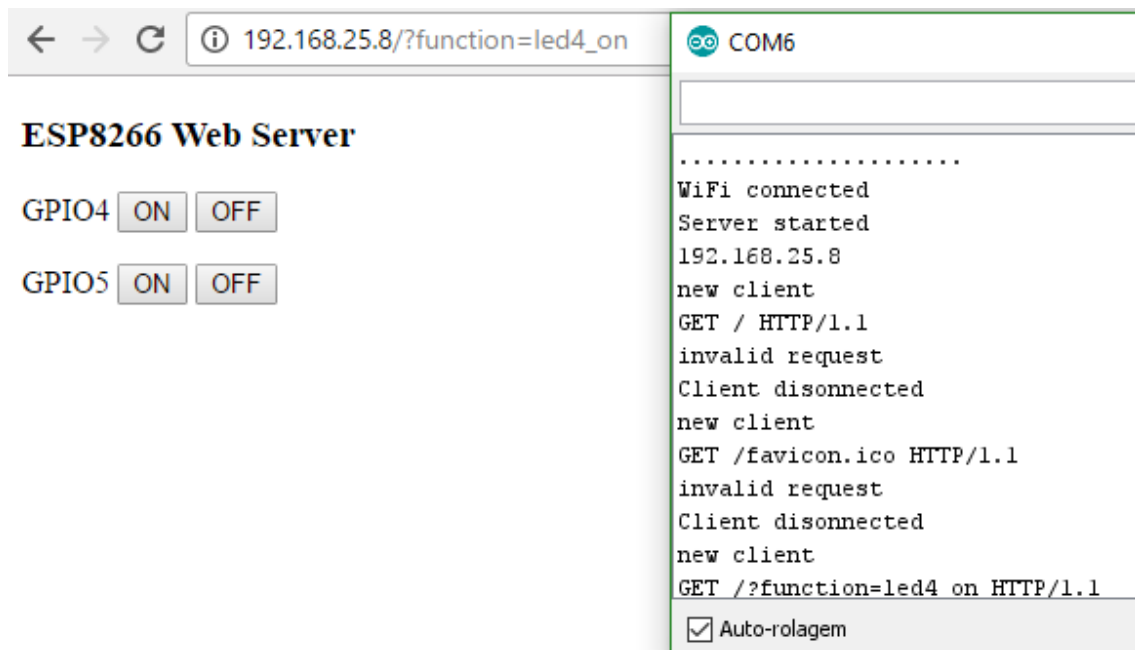
```

```

else if (req.indexOf("led6 off") != -1)
    digitalWrite(12, 0);
else {
    Serial.println("invalid request");
    cliente.stop();
}
Serial.println("Client disonnected");
}

```

O *printscreen* abaixo ilustra a operação do código do ESP operando como servidor. Mais detalhes desse exemplo no vídeo em: https://youtu.be/jt_elaBf6YE



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.

Servidor Async HTTP e WebSocket

Para o ESP8266, é necessária incluir as bibliotecas ESPAsyncWebServer e o ESPAsyncTCP acessando o item "adicionar bibliotecas .zip" em Sketch.

Para o ESP32, as bibliotecas necessárias são ESPAsyncWebServer e AsyncTCP. Para isso, basta baixar as bibliotecas em <https://github.com/me-no-dev/ESPAsyncWebServer> e extrair na pasta

`C:\Users\NomeDoUsuario\Documents\Arduino\hardware\espressif\esp32\libraries`

CONVERSOR ANALÓGICO DIGITAL (AD) COM O ESP32

Portas analógicas são comumente aplicadas ao controle e análise de sensores. Os 18 pinos analógicos disponíveis em placas ESP32, operam com resolução em 12 bits, tornando a faixa de leitura de 0 a 4095. Trabalhar com qualquer pino ADC será exatamente igual ao procedimento utilizado em placas Arduino, realizando leituras e controle por meio de funções como *analogRead* e *analogWrite*. Porém, ao conectar-se ao Wi-Fi, por exemplo, os pinos ADC2 não irão fornecer leituras, pois, até o presente momento, tais pinos serão utilizados como alimentação para o módulo, portanto, se desejar atribuir uma leitura analógica a qualquer programação conectada a internet, utilize os pinos ADC1. Um exemplo de aplicação do conversor AD no ESP32 pode ser visto em <https://www.youtube.com/watch?v=xp2em6C9Leg>. Um exemplo com AD do ESP8266 pode ser visto em <https://www.youtube.com/watch?v=WCZhH9DwQFY>.

//Video: <https://www.youtube.com/watch?v=xp2em6C9Leg>

```
int sensorPin = 15; // select the input pin for the potentiometer
int ledPin = 2; // select the pin for the LED
int sensorValue = 0; // variable to store the value coming from the sensor

void setup() {
  // declare the ledPin as an OUTPUT:
  pinMode(ledPin, OUTPUT);
}

void loop() {
  // read the value from the sensor:
  sensorValue = analogRead(sensorPin);
  // turn the ledPin on
  digitalWrite(ledPin, HIGH);
  // stop the program for <sensorValue> milliseconds:
```



```
delay(sensorValue);
// turn the ledPin off:
digitalWrite(ledPin, LOW);
// stop the program for for <sensorValue> milliseconds:
delay(sensorValue);
}
```

REPASSANDO PARÂMETROS PARA UM SERVIDOR COM GET

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>
#include <WiFi.h>
const char* rede = "*****"; //Cria o vetor de char rede
const char* senha = "*****";

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 conexao 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 usuario entrar com o cód da cidade
  while (!Serial.available()); //Aguarda os bytes de entrada serial
  while (Serial.available()) { //Guarda todos os caracteres car 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 caracteres na String resposta
    {
        char c = cliente.read();
        Serial.print(c);
        resposta += c;
    }
}
delay(1000);
cod = ""; //Limpa a String
resposta = "";
}

```



```
//-----//
void setup() {
  pinMode(ledPin, OUTPUT);

  Serial.begin(115200); // inicializa a porta serial com 115200 baud
  Serial.println(rede);
  WiFi.begin(rede, senha); //conecta a Rede
  Serial.println("Conectando ao WIFI...");
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print("-");
  }
  Serial.print("\nIP: ");
  Serial.println(WiFi.localIP()); //Imprime o IP

  configTime(timezone * 3600, dst * 0, "pool.ntp.org", "time.nist.gov");
  while (!time(nullptr)) {
    Serial.print(".");
    delay(1000);
  }
}
//-----//
void loop() {

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

  if (cliente.connect(host, 80))//
  {

    time_t now;
    struct tm * timeinfo;
    time(&now);
    //timeinfo = gmtime(&now);
    timeinfo = localtime(&now);

    sprintf(url, "GET /ftpmonitor/getESP_cond.php?action=send1&condut=%d&date=%02d-%02d-%02d-%02d:%02d:%02d", analogRead(sensorPin), timeinfo->tm_year + 1900, timeinfo->tm_mon + 1, timeinfo->tm_mday, timeinfo->tm_hour, timeinfo->tm_min, timeinfo->tm_sec);

    Serial.println(String(url)); // depurar a url ela serial
    cliente.print(String(url));
```

```
cliente.print (" HTTP/1.1\r\n");
cliente.print ("Host: ");
cliente.print (host);
cliente.print ("\r\nConnection: close\r\n\r\n");

}

digitalWrite(ledPin, LOW);
delay(3000);
digitalWrite(ledPin, HIGH);
delay(3000);
}
```

INERRUPÇÃO EXTERNA

Os pinos de entrada e saída podem ser configurados como OUTPUT, INPUT, INPUT_PULLDOWN ou INPUT_PULLUP. No caso do INPUT_PULLUP, é habilitado um resistor interno, ou seja, não sendo necessário ligar um resistor externo no Vcc para definir o nível lógico alto no pino.

No caso do INPUT_PULLUP, quando um botão, conectado ao Gnd (0), é pressionado será detectado a queda de tensão (falling).

```
pinMode(GPIO_BOTAO, INPUT_PULLUP);
```

```
attachInterrupt(GPIO_BOTAO, funcao_ISR, FALLING);
```

No caso do INPUT_PULLDOWN, quando um botão, conectado ao Vcc, é pressionado é detectado a queda de tensão (rising).

```
pinMode(GPIO_BOTAO, INPUT_PULLDOWN);
```

```
attachInterrupt(GPIO_BOTAO, funcao_ISR, RISING);
```

Mais detalhes em:

<https://youtu.be/HAuJ4zn4S84>

<https://www.filipeflop.com/blog/uso-de-interruptoes-externas-com-esp32/>

<https://www.arduino.cc/reference/pt/language/functions/external-interrupts/attachinterrupt/>

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 firmam 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 consequência 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 para 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 default 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 main window displays a C++ sketch for a task named `Rtos1ESP32Mutex`. The code includes semaphore and task management functions from the `FreeRTOS` library. It sets up a serial port at 115200 baud, configures a pin mode, and creates a semaphore. Three tasks are created: `teste`, `teste2`, and `teste3`, each with a period of 10000. The `loop` function toggles a digital pin (pin 2) and prints "SanUSB" to the serial monitor. The right-hand pane shows the serial monitor output, which displays the sequence of task execution: `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`, and `SanUSB`. The `Auto-rolagem` (Auto-scroll) checkbox is checked.

```

Arquivo Editar Sketch Ferramentas Ajuda
Rtos1ESP32Mutex
20 xSemaphoreTake(SanMutex, portMAX_DELAY); //instancia o mutex que tambem é um semáforo
21 Serial.print("Task Teste3: ");
22 Serial.println(my_shared_var);
23 if (my_shared_var > 1) {--my_shared_var;} else {++my_shared_var;}
24 Serial.println((char*)pvParameters);
25 delay(2000); //Trava um segundo segurando o token do Mutex
26 xSemaphoreGive(SanMutex);
27
28 }
29 }
30
31 void setup(){
32   Serial.begin(115200);
33   pinMode(2, OUTPUT); //pino com LED default na placa ESP32 DEVKIT
34   SanMutex = xSemaphoreCreateMutex();
35   if(SanMutex != NULL){
36     xTaskCreatePinnedToCore(teste, "Print1", 10000, (void *) "pvParametros da task1", 3, NULL, 0);
37     xTaskCreatePinnedToCore(teste2, "Print2", 10000, (void *) "pvParametros da task2", 3, NULL, 0); //u
38     xTaskCreatePinnedToCore(teste3, "Print3", 10000, NULL, 3, NULL, 0); // Para criar com mutex: xTas
39   }
40 }
41
42 void loop(){
43   digitalWrite(2, HIGH);
44   delay(1000);
45   digitalWrite(2, LOW);
46   delay(1000);
47   Serial.println("SanUSB"); //Executa sem o delay global do mutex
48 }

```

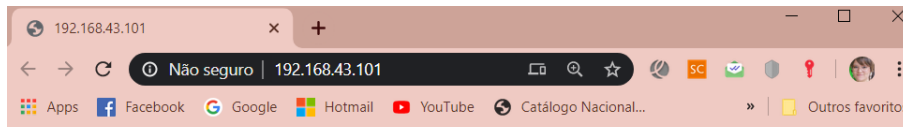
SanUSB
Task Teste: 1
SanUSB
Task Teste: 2
Task Teste3: 1
SanUSB
Task Teste: 2
SanUSB
Task Teste: 1
Task Teste3: 2
SanUSB
Task Teste: 1
SanUSB

☒ 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/>

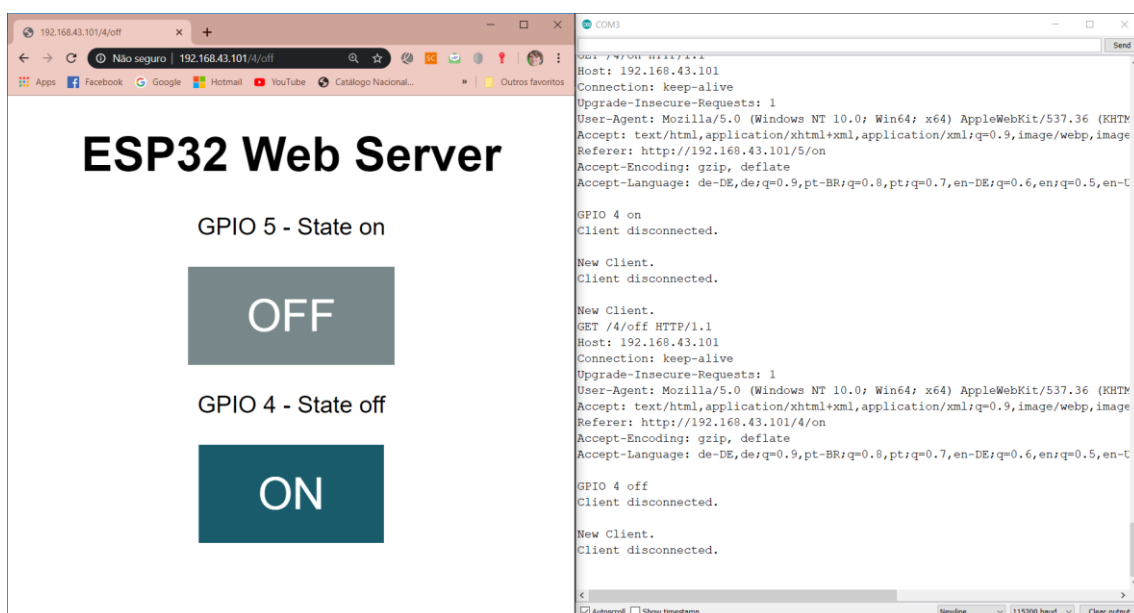


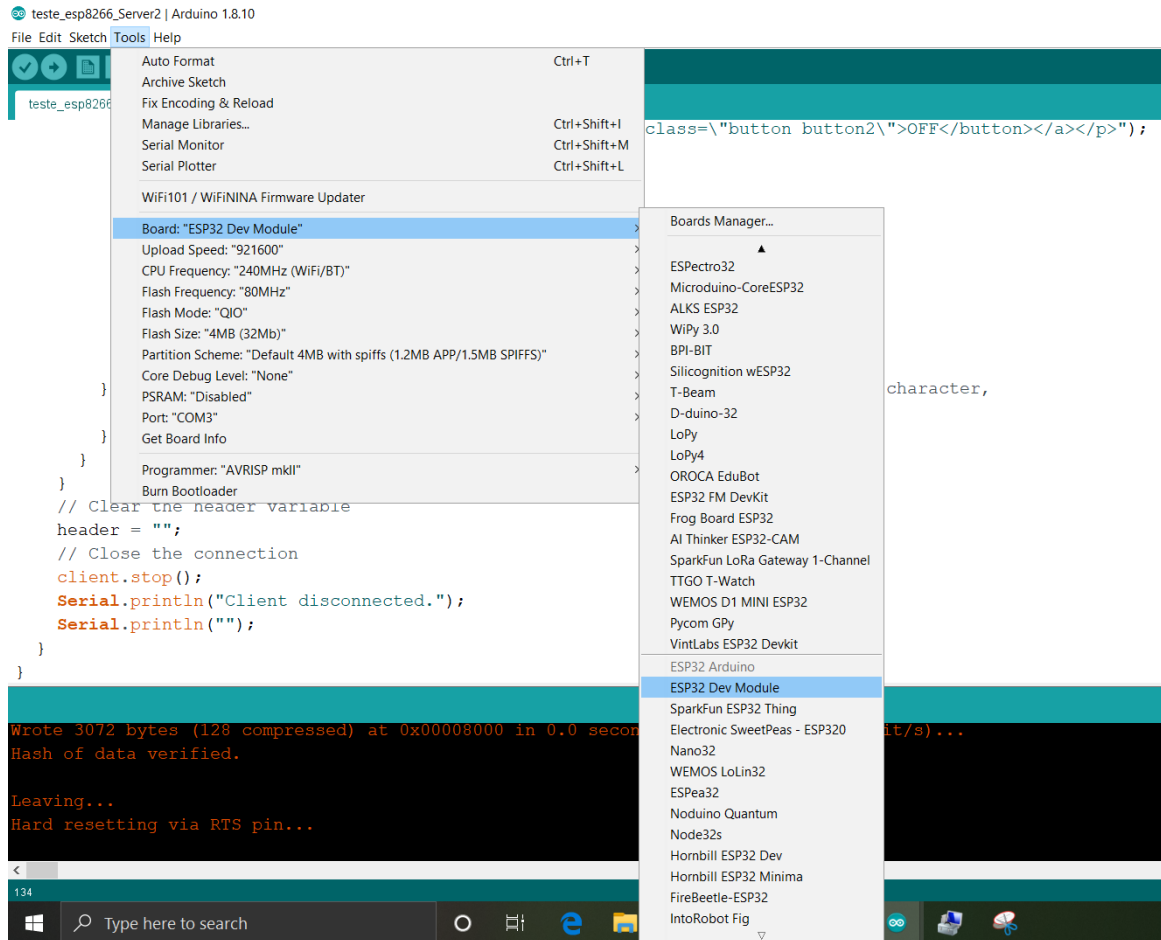
ESP32 Web Server

GPIO 5 - State off



GPIO 4 - State off





```
// Load Wi-Fi library
#include <WiFi.h>

// Replace with your network credentials
const char* ssid = "RenataESP";
const char* password = "risp2018";

// Set web server port number to 80
WiFiServer server(80);

// Variable to store the HTTP request
String header;

// Auxiliar variables to store the current output state
String output5State = "off";
String output4State = "off";

// Assign output variables to GPIO pins
const int output5 = 5;
const int output4 = 4;

// Current time
unsigned long currentTime = millis();
// Previous time
unsigned long previousTime = 0;
// Define timeout time in milliseconds (example: 2000ms = 2s)
const long timeoutTime = 2000;

void setup() {
  Serial.begin(115200);
  // Initialize the output variables as outputs
```

```
pinMode(output5, OUTPUT);
pinMode(output4, OUTPUT);
// Set outputs to LOW
digitalWrite(output5, LOW);
digitalWrite(output4, LOW);

// Connect to Wi-Fi network with SSID and password
Serial.print("Connecting to ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    delay(500);
    Serial.print(".");
}
// Print local IP address and start web server
Serial.println("");
Serial.println("WiFi connected.");
Serial.println("IP address: ");
Serial.println(WiFi.localIP());
server.begin();
}

void loop(){
    WiFiClient client = server.available(); // Listen for incoming clients

    if (client) { // If a new client connects,
        Serial.println("New Client."); // print a message out in the serial port
        String currentLine = ""; // make a String to hold incoming data from the client
        currentTime = millis();
        previousTime = currentTime;
        while (client.connected() && currentTime - previousTime <= timeoutTime) { // loop while the client's connected
            currentTime = millis();
            if (client.available()) { // if there's bytes to read from the client,
                char c = client.read(); // read a byte, then
                Serial.write(c); // print it out the serial monitor
                header += c;
                if (c == '\n') { // if the byte is a newline character
                    // if the current line is blank, you got two newline characters in a row.
                    // that's the end of the client HTTP request, so send a response:
                    if (currentLine.length() == 0) {
                        // HTTP headers always start with a response code (e.g. HTTP/1.1 200 OK)
                        // and a content-type so the client knows what's coming, then a blank line:
                        client.println("HTTP/1.1 200 OK");
                        client.println("Content-type:text/html");
                        client.println("Connection: close");
                        client.println();

                        // turns the GPIOs on and off
                        if (header.indexOf("GET /5/on") >= 0) {
                            Serial.println("GPIO 5 on");
                            output5State = "on";
                            digitalWrite(output5, HIGH);
                        } else if (header.indexOf("GET /5/off") >= 0) {
                            Serial.println("GPIO 5 off");
                            output5State = "off";
                            digitalWrite(output5, LOW);
                        } else if (header.indexOf("GET /4/on") >= 0) {
                            Serial.println("GPIO 4 on");
                            output4State = "on";
                            digitalWrite(output4, HIGH);
                        } else if (header.indexOf("GET /4/off") >= 0) {
                            Serial.println("GPIO 4 off");
                            output4State = "off";
                            digitalWrite(output4, LOW);
                        }
                    }
                }
            }
        }

        // Display the HTML web page
    }
}
```

```

client.println("<!DOCTYPE html><html>");
client.println("<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1\">");
client.println("<link rel=\"icon\" href=\"data:;\">");
// CSS to style the on/off buttons
// Feel free to change the background-color and font-size attributes to fit your preferences
client.println("<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center; }");
client.println(".button { background-color: #195B6A; border: none; color: white; padding: 16px 40px;");
client.println("text-decoration: none; font-size: 30px; margin: 2px; cursor: pointer; }");
client.println(".button2 { background-color: #77878A; } </style></head>");

// Web Page Heading
client.println("<body><h1>ESP32 Web Server</h1>");

// Display current state, and ON/OFF buttons for GPIO 5
client.println("<p>GPIO 5 - State " + output5State + "</p>");
// If the output5State is off, it displays the ON button
if (output5State=="off") {
  client.println("<p><a href=\"/5/on\"><button class=\"button\">ON</button></a></p>");
} else {
  client.println("<p><a href=\"/5/off\"><button class=\"button button2\">OFF</button></a></p>");
}

// Display current state, and ON/OFF buttons for GPIO 4
client.println("<p>GPIO 4 - State " + output4State + "</p>");
// If the output4State is off, it displays the ON button
if (output4State=="off") {
  client.println("<p><a href=\"/4/on\"><button class=\"button\">ON</button></a></p>");
} else {
  client.println("<p><a href=\"/4/off\"><button class=\"button button2\">OFF</button></a></p>");
}
client.println("</body></html>");

// The HTTP response ends with another blank line
client.println();
// Break out of the while loop
break;
} else { // if you got a newline, then clear currentLine
  currentLine = "";
}
} else if (c != '\r') { // if you got anything else but a carriage return character,
  currentLine += c; // add it to the end of the currentLine
}
}
}
// Clear the header variable
header = "";
// Close the connection
client.stop();
Serial.println("Client disconnected.");
Serial.println("");
}
}

```


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);

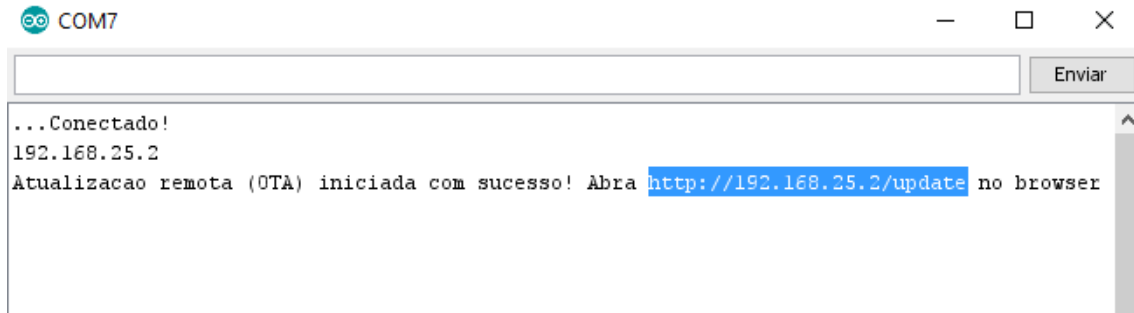
  httpUpdater.setup(&httpServer, update_path, update_username, update_password);
  httpServer.begin();

  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);
}

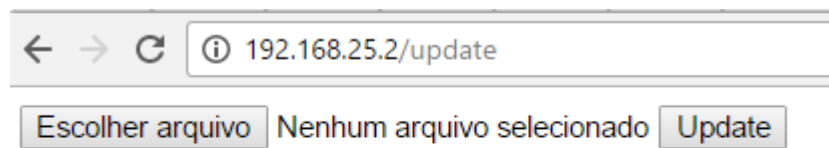
void loop(void){
httpServer.handleClient();

  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 o compilador PaltformIO, 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 reposta 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 invés 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; //Velocidade 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;

```



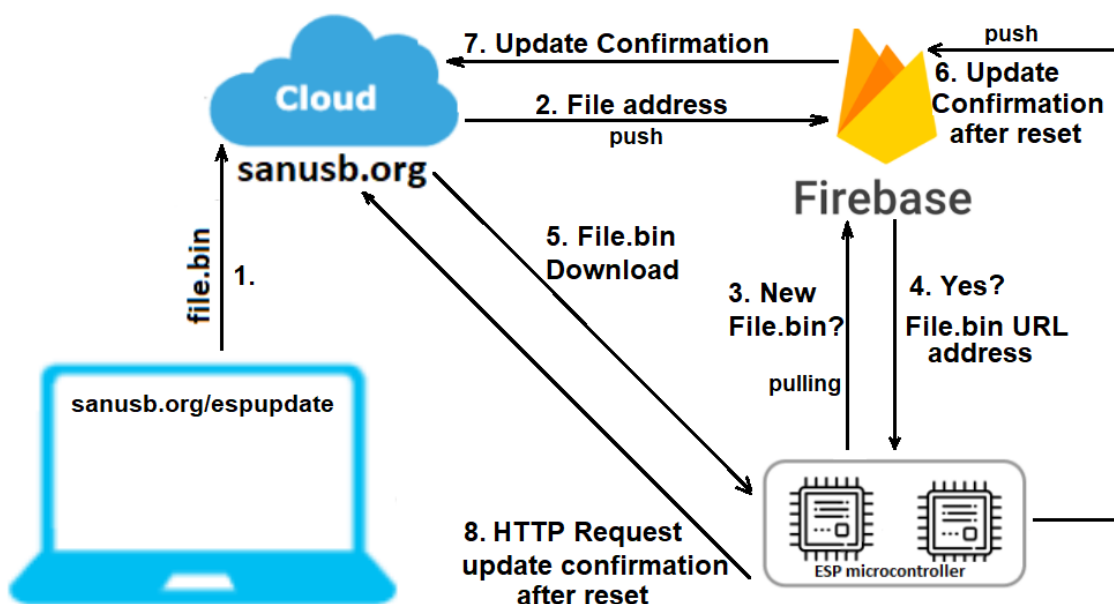
```

    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 WIFI OTA (OVER-THE-AIR) NO ESP

Este projeto mostra um ambiente de atualização de códigos de microcontroladores ESP32 e ESP266 com o mesmo sketch genérico usando o site <http://sanusb.org/espupdate>. A versão gratuita do Firebase (*Google JSON objects Database*) foi implementada para gerar o gatilho de atualização na nuvem para arquivos .bin conformr a ilustração abaixo.



A atualização (OTA) transmite os arquivos binários .bin compilados via Internet. Para gerar um arquivo .bin a partir do sketch, acesse a aba *Sketch* da IDE > *Export binário compilado*. Para realizar essa atualização de firmware na nuvem, os

usuários necessitam escrever no sketch apenas o ssid, a senha e o mesmo nome do perfil inserido no site <http://sanusb.org/espupdate> e ao fazer o upload do .bin para o site, acontece a atualização remota de firmware dos microcontroladores ESP via Internet. O nome do perfil de usuário inserido no site sanusb.org/espupdate pode ser alfanumérico (por exemplo: *perfil190575* ou *kekietevaldo*).

É possível testar esse aplicativo de transmissão pela Internet em redes diferentes, acessando o site <http://sanusb.org/espupdate/> através da rede do laboratório ou de casa e o microcontrolador ESP32/ESP8266 ancorado ao smartphone conectado à rede móvel 4G, ou vice vice-versa.

Para instalar as placas ESP32 e ESP8266 no Arduino IDE, siga as próximas instruções: No seu Arduino IDE, vá em File > Preferences. Insira nos "URLs adicionais do gerenciador de placa":

https://dl.espressif.com/dl/package_esp32_index.json,

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

Abra o Boards Manager: Vá em Tools > Board > Boards Manager: Procure por ESP32 e pressione o botão de instalação para o "ESP32 by Espressif Systems". Procure também por ESP8266 e pressione o botão de instalação para o "ESP8266 by ESP8266 Community".

As dependências da ferramenta EspCloudUpdate são as bibliotecas Firebase. Para ESP8266:

<https://github.com/mobizt/Firebase-ESP8266>

Para ESP32:

<https://github.com/mobizt/Firebase-ESP32>

Para instalar todas as bibliotecas, incluindo <https://github.com/SanUSB/EspCloudUpdate> , siga os passos:

Arduino IDE -> Sketch -> Include Library -> Add .zip Libraries.

Normalmente, você pode encontrar as bibliotecas ESP32 e ESP8266 instaladas em:

No Windows: C:\Users\UserName\Documents\Arduino\libraries (testado)

No Linux: /home/UserName/Arduino/Libraries (testado).

No OSX: ~/Documents/Arduino/libraries.

Na pasta da biblioteca EspCloudUpdate instalada há um exemplo para testar esta ferramenta proposta chamada EspUpdateStart.ino.

Vale considerar que, por meio de testes realizados, arquivos compilados .bin com o mesmo nome e com downloads sequenciais para atualização na nuvem, pode acontecer que o arquivo .bin enviado por último para a atualização não seja baixado pelo microcontrolador ESP, mas sim um arquivo .bin enviado anteriormente, pois eles têm o mesmo nome e a mesma URL de download. Por esta razão, neste projeto, o nome dos arquivos .bin enviados no site possuem nomes versionados (modificados), consequentemente o endereço URL de download também, isso impede que um arquivo

enviado anteriormente ou outro arquivo com o mesmo nome seja baixado, gerando uma URL única e exclusiva. Neste caso, para a versão do nome e URL de download do arquivo .bin, foram utilizados: ano, mês, dia, hora e a ordem cíclica de *upload*.

Para a condição de atualização {if (newVersionInt> InitialVersionInt)}, considera-se entre as variáveis maiores que (>) e não diferentes (!=), para garantir que a versão do arquivo .bin mais recente seja sempre maior que a versão .bin anterior e para evitar que a leitura de variáveis possa gerar acionamento de atualização indesejada durante a operação.

Em resumo, baixe e instale a biblioteca de amostra .zip disponível em <https://github.com/SanUSB/EspCloudUpdate> e dependências para ESP8266 (<https://github.com/mobizt/Firebase-ESP8266>) ou ESP32 (<https://github.com/mobizt/Firebase-ESP32>).

Para instalar as bibliotecas, siga os passos: Arduino IDE -> Sketch -> Include Library -> Add .zip Libraries. Descompacte a pasta .zip e abra o exemplo EspUpdateStart.ino. Escreva o nome do seu perfil e rede WiFi no esboço. Atualize o esboço apenas na primeira vez usando a porta USB. Então agora é possível transferir pela internet gerando o arquivo .bin. Para isso, acesse o menu Sketch da IDE do Arduino > Exportar binário compilado.

Após a conclusão da atualização do arquivo .bin e o reset automático do microcontrolador, o novo código de verificação é enviado para http://sanusb.org/espupdate/*YourProfile*/ para confirmar a atualização no site.

Neste projeto, a função loop sketch é praticamente gratuita para que você possa implementar seus projetos e poder atualizá-los de forma simples e remota pela Internet. Tutorial:

https://youtu.be/En_hFO5f4U8 e <https://youtu.be/GjbDdPD5cWM>.

Vale salientar que para utilizar o EspCloudUpdate não é necessário abrir portas no roteador ou habilitar permissões de firewall.

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 seguimentos.

```
/* 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, 1, 0, // Plota 0
    0, 1, 1, 0, 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 commando (Mac/Linux/Windows), após ter instalado o python 2.7 (<https://www.python.org/downloads/>) ou superior adicionando o *Python.exe* to *Path*:

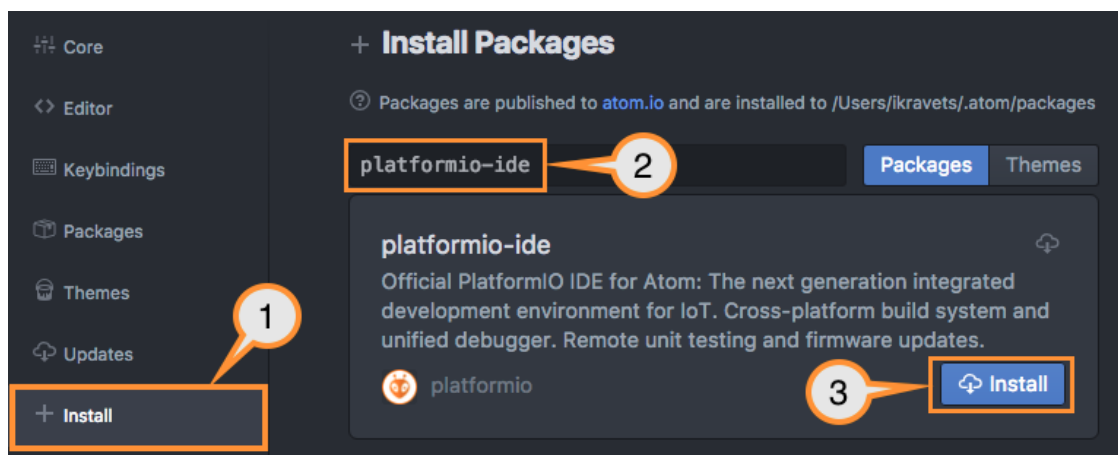
```
pip install -U platformio
```



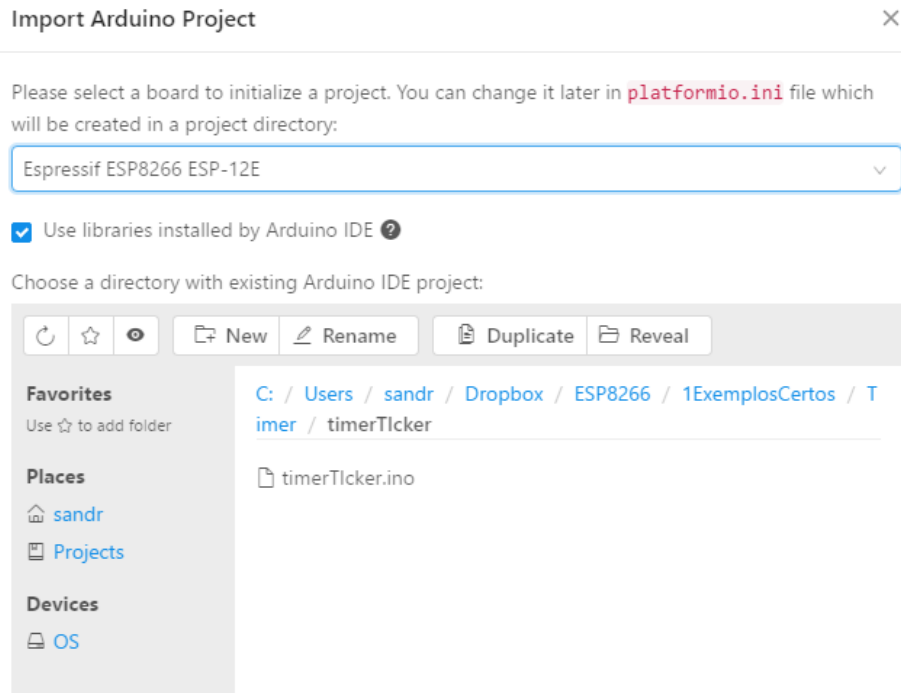

Outras formas de instalação em: <http://docs.platformio.org/en/latest/installation.html>

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

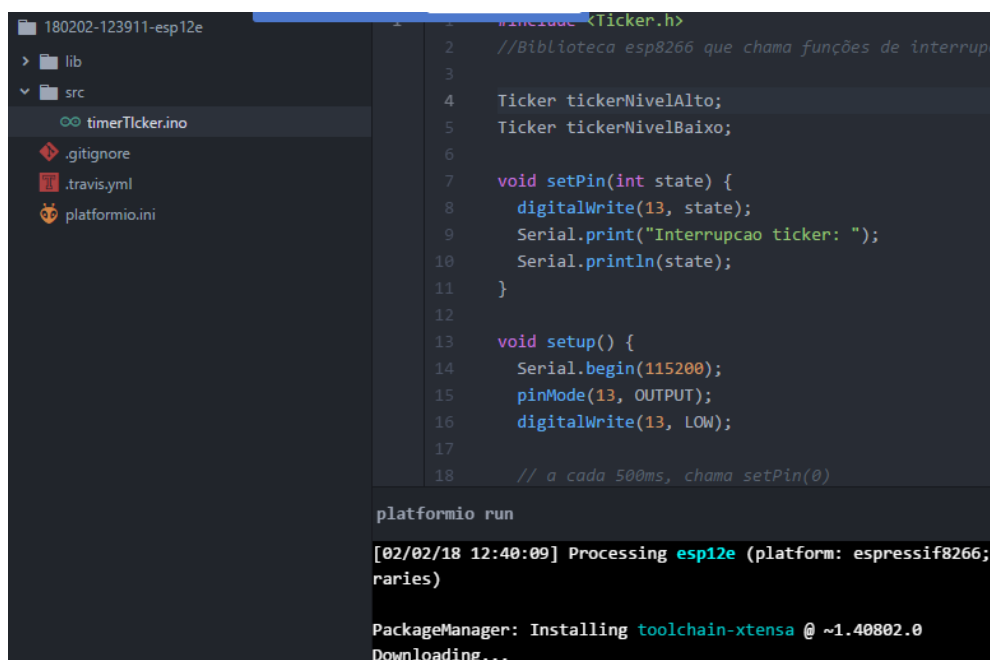


A IDE Atom foi criada pela comunidade do GITHUB para edição de códigos. É importante salientar que não é necessário instalar o PlatformIO Core se for instalado a IDE PlatformIO Atom, pois o Core já é integrado ao IDE e é possível acessá-lo no Terminal do PlatformIO. Para realizar a migração de um firmware 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*.



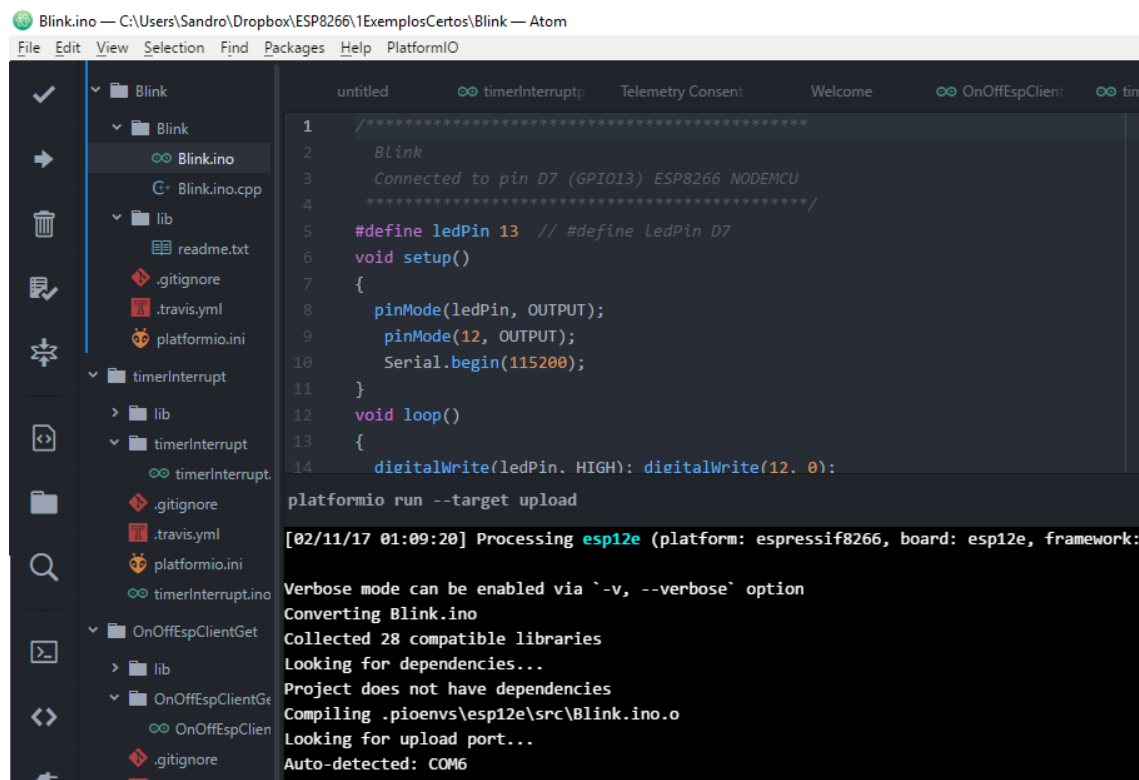
Essa plataforma compila também a linguagem *Wiring* de programação do Arduino, mas a linguagem de compilação padrão do platformIO e similar a wiring e mais rápida é a c++ (cpp). Ao importar um projeto baseado na IDE do Arduino, o Platformio cria uma subpasta, por exemplo em *C:\Users\[NOME]\Documents\PlatformIO\Projects* 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.



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 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 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*).



The screenshot shows the PlatformIO IDE interface. On the left, the file explorer displays the project structure, including the 'Blink' folder with 'Blink.ino' and 'Blink.ino.cpp'. The main editor shows the code for 'Blink.ino', which includes a comment about connecting to pin D7 (GPIO13) on an ESP8266 NodeMCU, and defines a LED pin 13. The code defines a setup function to initialize pin 12 as an output and a loop function to toggle the LED. The terminal at the bottom shows the command 'platformio run --target upload' and the output, which includes the platform and board details (espressif8266, esp12e), the verbose mode option, and the compilation process of 'Blink.ino' to 'Blink.ino.cpp'.

```

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);

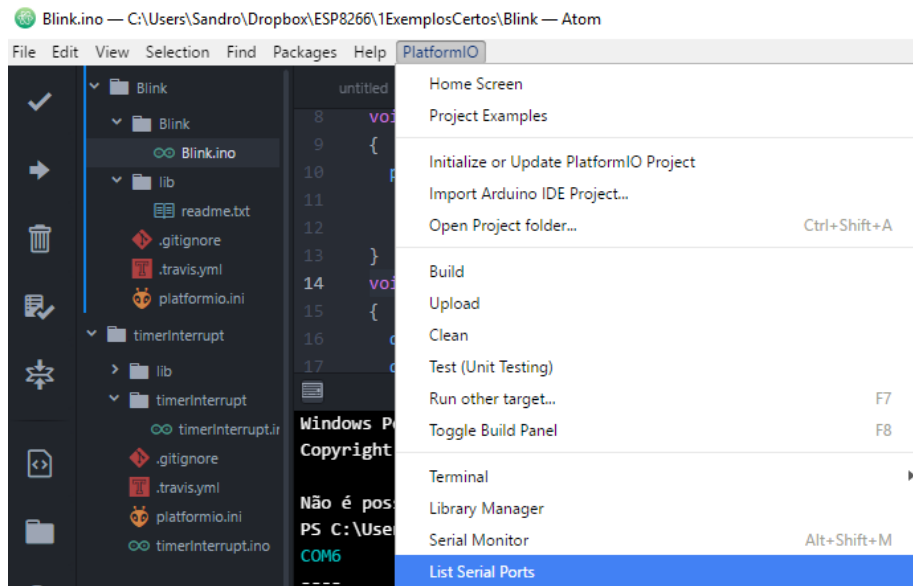
```

```

platformio run --target upload
[02/11/17 01:09:20] Processing esp12e (platform: espressif8266, board: esp12e, framework:
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

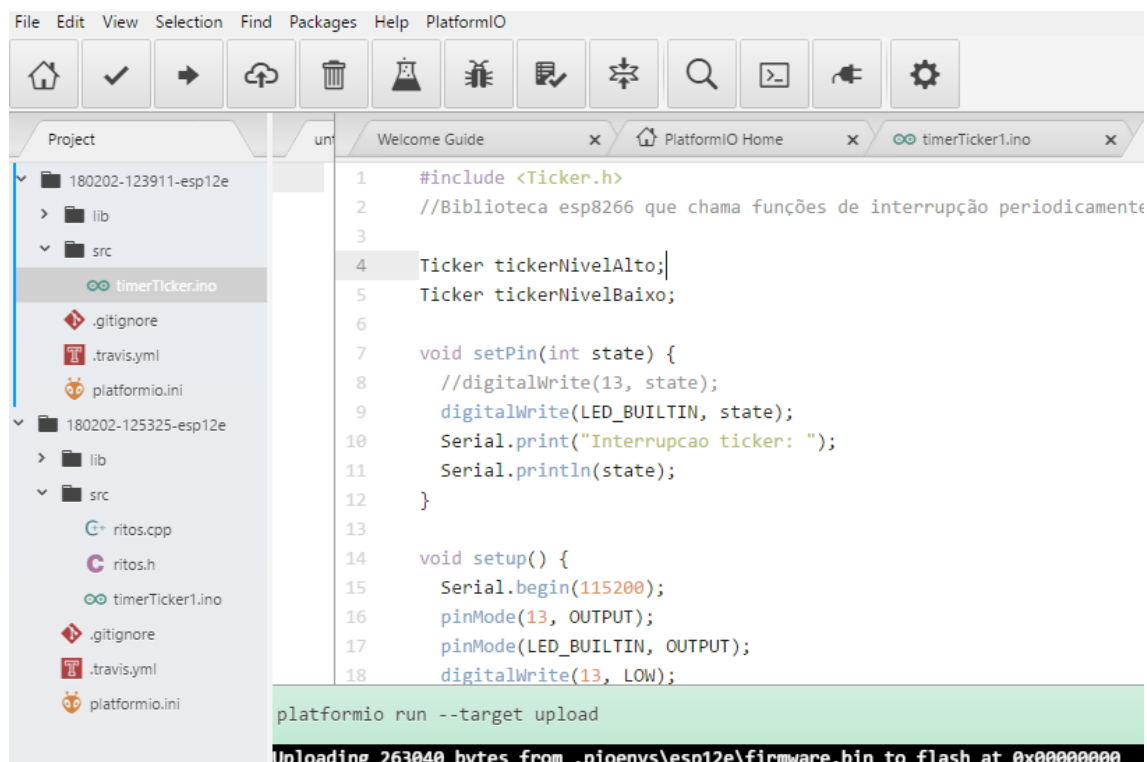
```

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

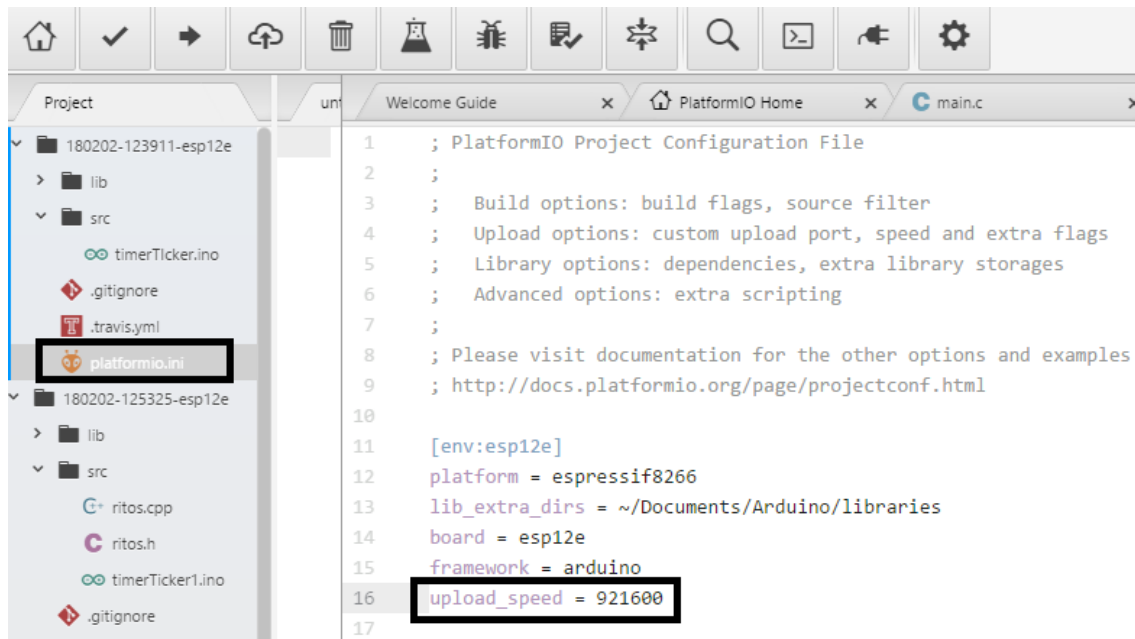


Se quiser salvar automaticamente todos os arquivos após clicar em build, basta ir em *build: PlatformIO > Settings > Build* e selecione: *Automatically save on build*. Esse ambiente gera o binário em *.pioenvs\esp12e\firmware.bin* que pode ser utilizado para gravação *Over the Air* através da biblioteca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

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

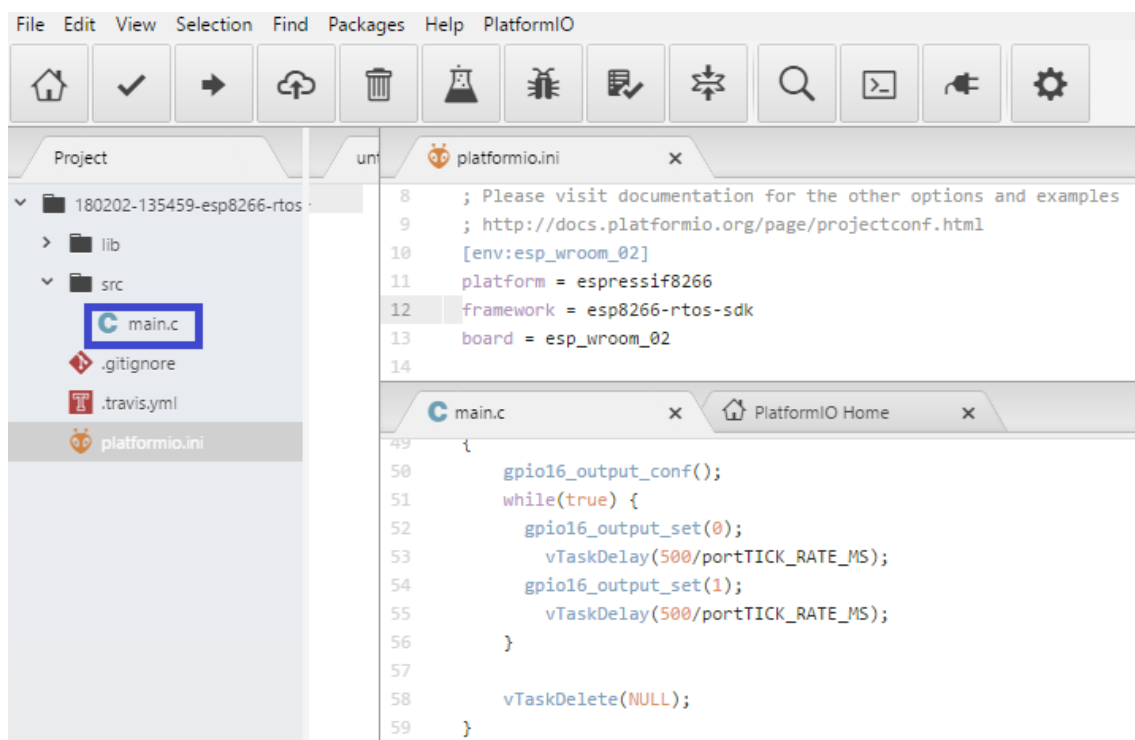


É possível alterar a taxa de transferência de gravação do ESP (pulo), acessando o arquivo `platformio.ini` e digitando `upload_speed = 921600`, como na Figura abaixo:



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

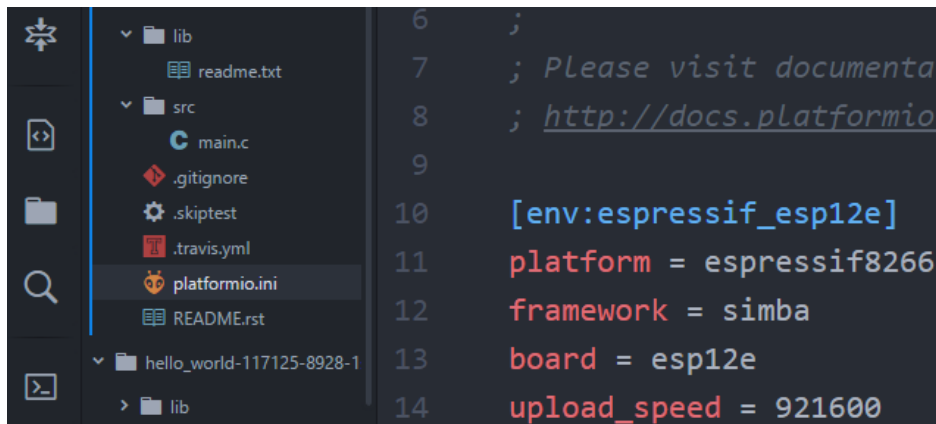
É possível também compilar projetos RTOS FreeRTOS, baixando o exemplo em Platformio Home -> *Project Examples* e selecionando *esp-rtos-sdk-blink*.



COMPILADOR 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 deseje 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

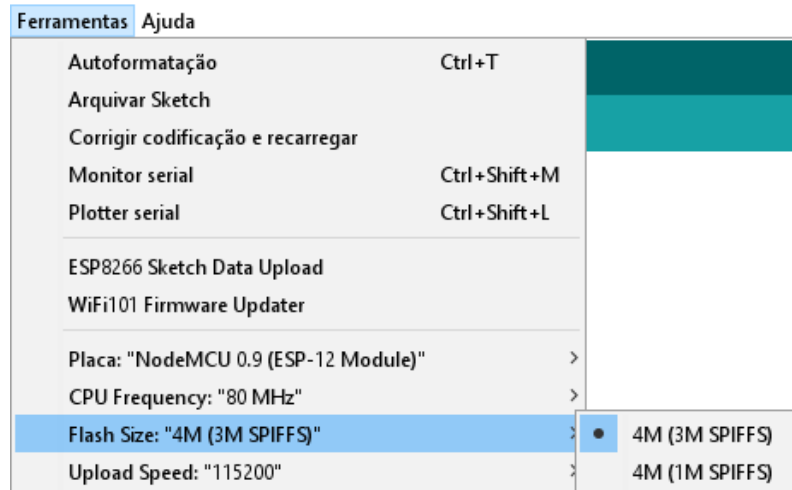
O ESP8266, não possui memória não volátil interna, sendo necessário incluí-la externamente no módulo. No caso dos módulos do ESP8266, essa memória é uma memória flash que pode variar de 512kB até 4MB, que é o caso dos módulos NodeMCU e utiliza um barramento QSPI (*queued serial peripheral interface*).

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

Dá para utilizar a memória 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 memória 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 memória 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 nao 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-os 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());

  //Inicia o webserver
  server.begin();
  writeFile("Web Server started");
}

void loop(void){

  //Tratamento das requisições http
  WiFiClient client = server.available();
  if (!client) {
    return;
  }

  Serial.println("new client");
```

```
while(!client.available()){
  delay(1);
}

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

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 style=\"\"text-align: center;\"\">ESP8266 Web Log</h3>";
buf += "<p>";
readFile();
buf += "</p>";
buf += "<h4>Exemplo de Registro de Logs em SPIFFS</h4>";
buf += "</html>\r\n";

client.print(buf);
client.flush();

Serial.println("Client disconnected");
}
```

Os exemplos em <https://github.com/SanUSB/AsyncIOT> têm o objetivo de mostrar como embarcar páginas html e javascript na memória SPIFFS no ESP32 e no ESP8266 e os passos para servir um app do React.js e não de explicar o desenvolvimento do código, pois será utilizado um exemplo de aplicativo padrão do React já existente, que pode ser visto nesse [link](#).

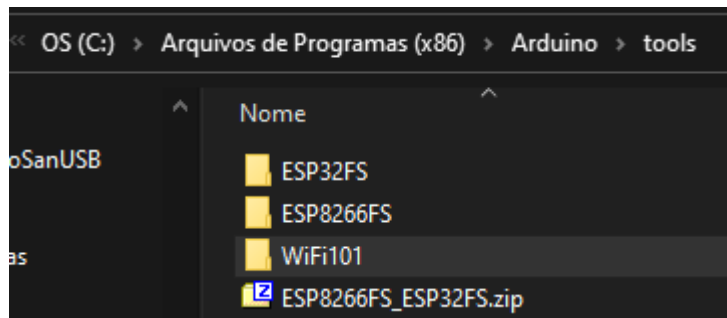
1. Exemplo 1Form_LED.ino (Comutar pino do ESP32 e ESP8266 através de página html e Javascript na memória SPIFFS): <https://youtu.be/5d9Z8eptu8A>
2. Exemplo 2Form_LED_PWM.ino (Comutar a intensidade luminosa do pino por PWM no ESP32 e ESP8266 através de página html e Javascript na memória SPIFFS).
3. Exemplo 3Forms_Reactjs.ino (Servindo um exemplo de App React.js com ESP8266 e ESP32 na memória SPIFFS) faz parte de mais uma aplicação Async com código disponível nos exemplos do repositório <https://github.com/SanUSB/AsyncIOT> onde todas as bibliotecas Async descritas no README do repositório devem ser previamente instaladas.

Depois de baixar e extrair o arquivo .zip do exemplo do aplicativo React, são necessários somente o arquivos demo.js e index.html, que devem ser colocados na pasta data para serem gravados na memória SPIFFS de acordo com os tutoriais

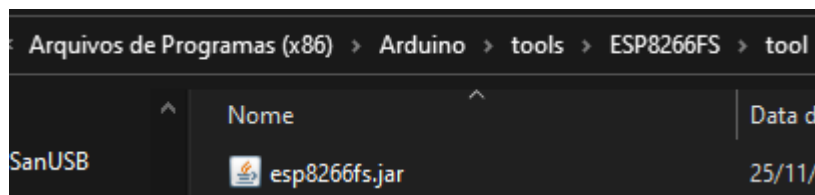
<https://youtu.be/XiBGUGb7Fdo> e <https://youtu.be/5d9Z8eptu8A> , senso possível excluir todos os outros arquivos do exemplo .zip.

Memória SPIFFS:

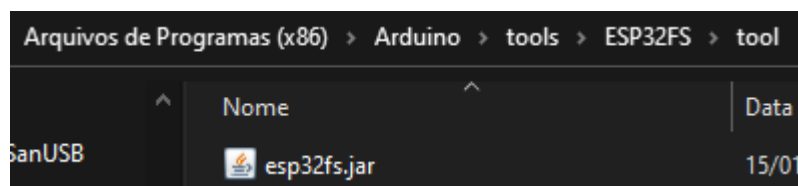
Baixe e extraia as pastas ESP8266FS e ESP32FS disponíveis em http://sanusb.org/tools/ESP8266FS_ESP32FS.zip dentro de C:\Program Files (x86)\Arduino\tools.



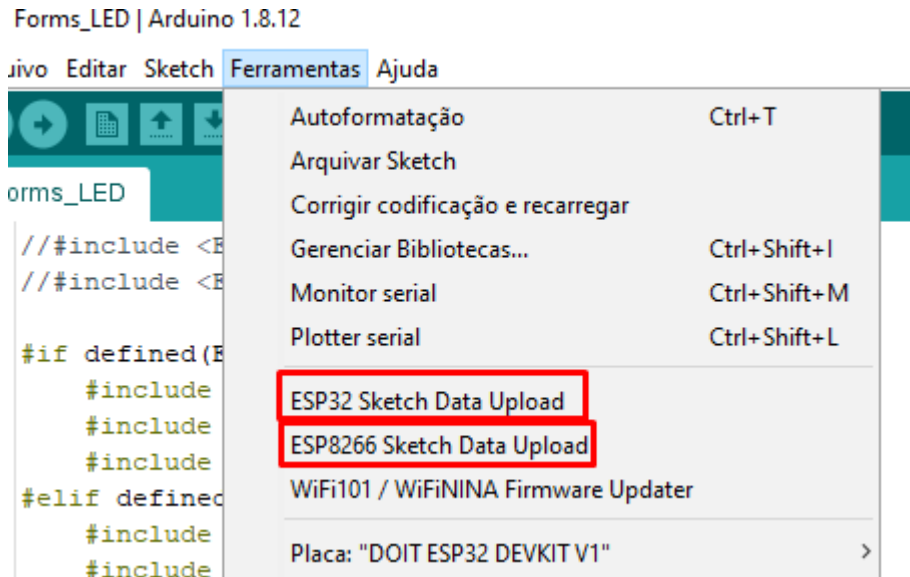
Verifique se o arquivo esp8266fs.jar está contido dentro da pasta tools, no endereço abaixo C:\Program Files (x86)\Arduino\tools\ESP8266FS\tool.



Verifique se o arquivo esp32fs.jar está contido dentro da pasta tools, no endereço abaixo C:\Program Files (x86)\Arduino\tools\ESP32FS\tool.



Depois disso, reinicie a IDE do Arduino e transfira pela serial os arquivos contidos na pasta Data pela Arduino IDE clicando em Sketch Data Upload.



Clicando em ESP8266 Sketch Data Upload, os arquivos contidos na pasta DATA sobem para a memória SPIFFS do ESP.

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, addr); //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 memória.

```
#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 memória 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.

INTRODUÇÃO A IoT COM MQTT

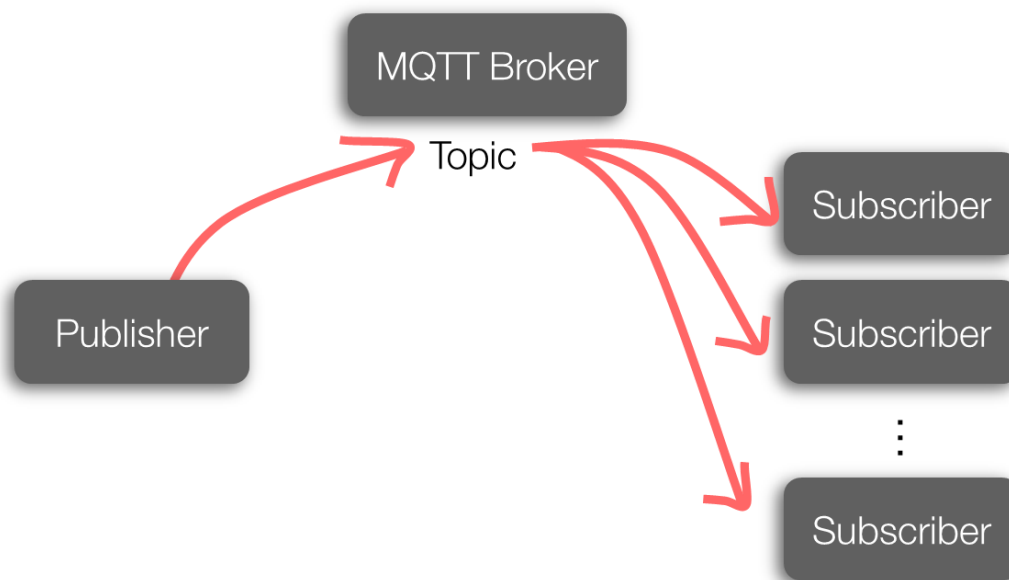
No modelo convencional de comunicação WEB, toda nova notificação somente será realizada se houver uma requisição do cliente. Isto torna o modelo ineficiente, pois em diversas situações é necessário atualizar o cliente no momento em que ocorreram modificações nos dados do servidor, sob pena da informação 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.

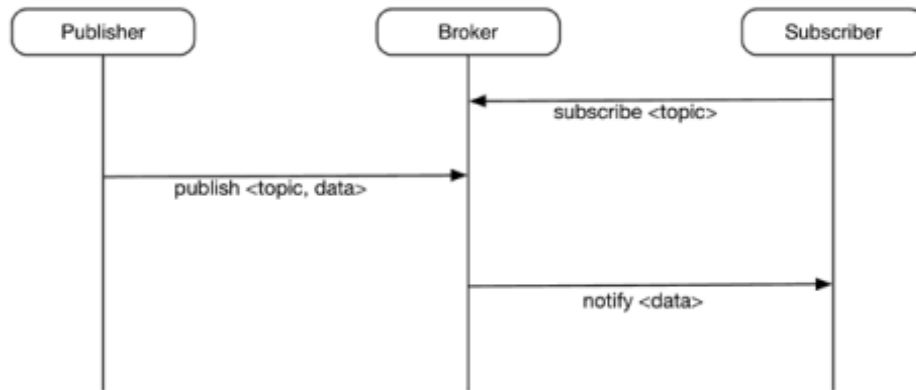
O MQTT permite o envio de mensagens para **filas**, e estas podem ser escutadas por outros dispositivos inscritos. O protocolo utiliza uma arquitetura *publish/subscribe*, em contraste ao *request/response* do protocolo web HTTP. A insustentável leveza desse ser consagrou-lhe como o protocolo oficial 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 IFCE (*ifce.sansubs.org*). Um exemplo de

firmware testado para enviar dados para o broket mqtt e acompanhá-lo por terminais inscritos como subscriber no smartphone e no google chrome é mostrado abaixo.

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

const char* ssid = "LAESE";
const char* password = "*****";
const char* mqtt_server = "ifce.sanusb.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. Um vídeo demonstra outro exemplo em que um LED é comandado por uma interface da plataforma sanusb.org/gpio e pode enviar valores (comentado no código) para um determinado tópico via MQTT.

To install the ESP8266 board, (using Arduino 1.6.4+):

- Add the following 3rd party board manager under "File -> Preferences -> Additional Boards Manager URLs":
http://arduino.esp8266.com/stable/package_esp8266com_index.json
- Open the "Tools -> Board -> Board Manager" and click install for the ESP8266"
- Select your ESP8266 in "Tools -> Board"

```

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

//const char* ssid = "LAESE";
//const char* password = "*****i";
//const char* mqtt_server = "ifce.sanusb.org";

const char* ssid = "HOTNET_SanUSB";
const char* password = "*****";
const char* mqtt_server = "ifce.sanusb.org";

const char* topico = "san5";

```

```

WiFiClient espClient;
PubSubClient client(espClient);
long lastMsg = 0;
char msg[50];
int value = 0;

void setup_wifi() {

  delay(10);
  // We start by connecting to a WiFi network
  Serial.println();
  Serial.print("Connecting to ");
  Serial.println(ssid);

  WiFi.begin(ssid, password);

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

  randomSeed(micros());

  Serial.println("");
  Serial.println("WiFi connected");
  Serial.println("IP address: ");
  Serial.println(WiFi.localIP());
}

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++) {
    Serial.print((char)payload[i]);
  }
  Serial.println();

  // Switch on the LED if an 1 was received as first character
  if ((char)payload[0] == '0') {
    digitalWrite(2, HIGH); // o Builtin Led da placa NodeMCU Lolin apaga (on) quando o valor do pino 2 é HIGH
  } else {

```



```

    digitalWrite(2, LOW); // o Bultin Led 2 da placa NodeMCU Lolin acende (on) quando o valor do pino 2 é LOW
(0)
    Serial.println("On"); //Vcc ---->|-----pino 2
}

}

void reconnect() {
    // Loop until we're reconnected
    while (!client.connected()) {
        // Serial.print("Attempting MQTT connection...");
        // Create a random client ID
        String clientId = "esp-test";
        // Attempt to connect
        if (client.connect(clientId.c_str())) {
            // Serial.println("connected");
            // ... and resubscribe
            client.subscribe(topico); //*****Topico*****
        } else {
            Serial.print("failed, rc=");
            Serial.print(client.state());
            Serial.println(" try again in 5 seconds");
            // Wait 5 seconds before retrying
            delay(1000);
        }
    }
}

void setup() {
    pinMode(2, OUTPUT); // D5 Initialize the BUILTIN_LED pin as an output  pinMode(2, OUTPUT); //bultin Led
Lolin
    Serial.begin(115200);
    setup_wifi();
    client.setServer(mqtt_server, 1883);
    client.setCallback(callback);
}

void sendData(){
    //329.104477612
    float temp = (analogRead(A0)*3.3/(1023*0.01));

    float light = analogRead(A0)*5/(1023);
    float solar = (analogRead(A0)/329.104477612);
    String solar2 = String(temp);

```

```
String message = String("Valor: "+ solar2);
client.publish(topico, &message[0]); //sanusmqtt/monitor
delay(5000);
}

void loop() {

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

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

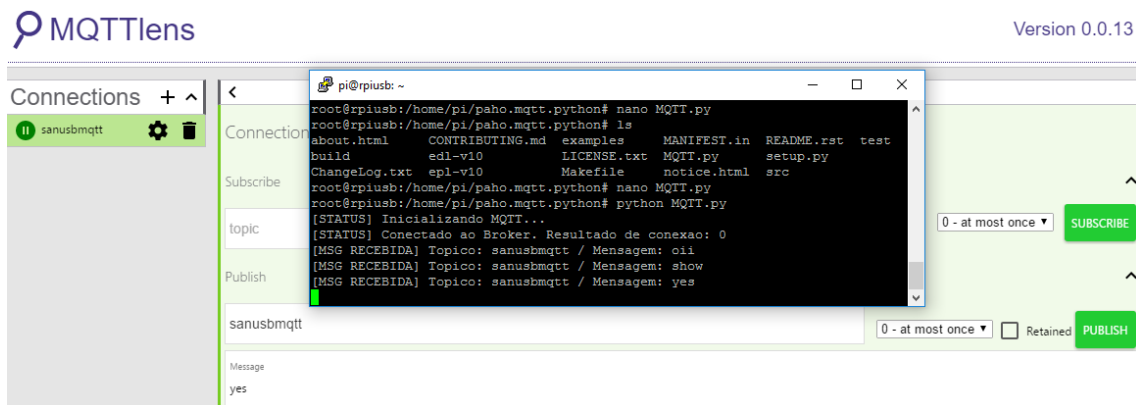
<https://chrome.google.com/webstore/detail/mqttlens/hemojaaeigabkbcookmlgmdigohjobjm> ou o **MyMQTT** disponível para Android em https://play.google.com/store/apps/details?id=at.tripwire.mqtt.client&hl=pt_BR. Depois de instalar o cliente MQTT, conecte-se ao servidor broker iot.eclipse.org e publique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

Connection Details

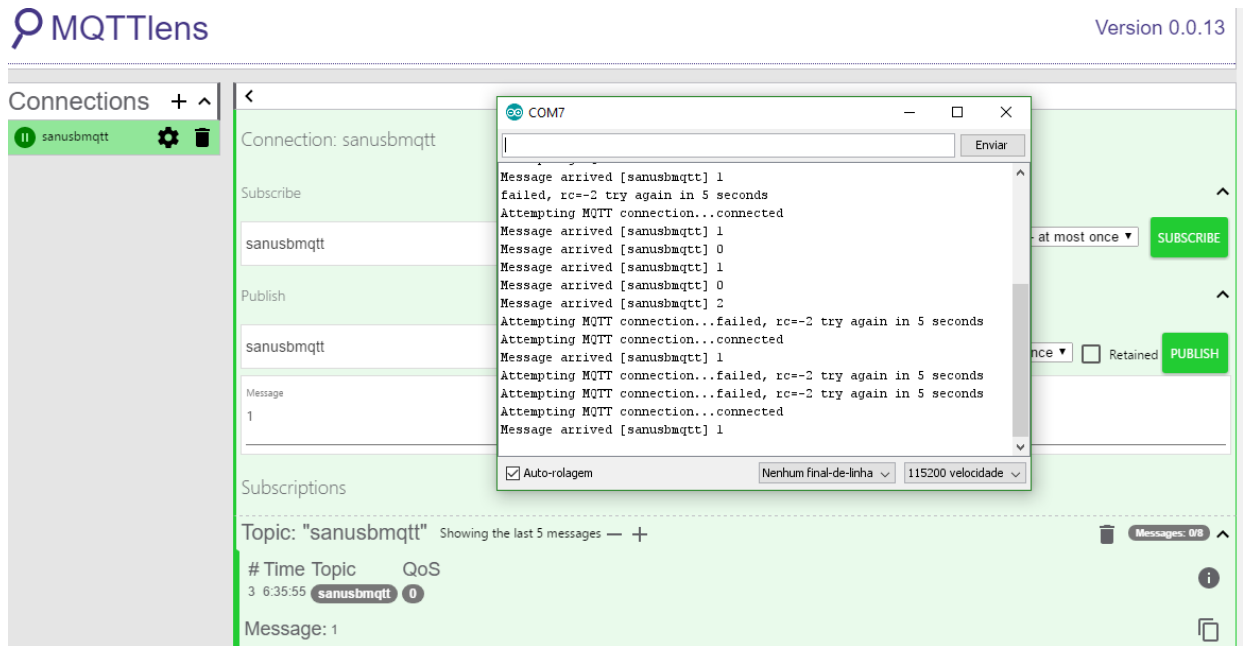
Connection name <input type="text" value="sanusbmqtt"/>		Connection color scheme <div style="background-color: #76b82a; width: 100px; height: 15px;"></div>
Hostname <input type="text" value="tcp://"/> <input type="text" value="iot.eclipse.org"/>	Port <input type="text" value="1883"/>	
Client ID <input type="text" value="lens_VoxFWCTXJ1iV6yoX60xzUu37AC"/>		<input type="button" value="Generate a random ID"/>
Session <input checked="" type="checkbox"/> Clean Session	Automatic Connection <input checked="" type="checkbox"/> Automatic Connection	Keep Alive <input type="text" value="120"/> <input type="text" value="seconds"/>

Credentials

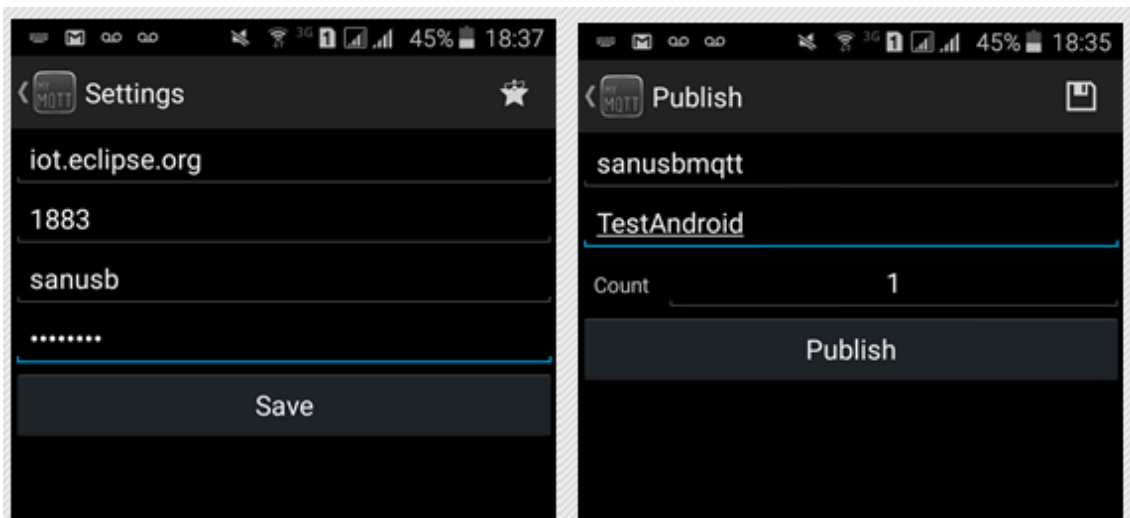
Username <input type="text" value="sanusb"/>
Password <input type="password" value="*****"/>



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 acesse 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:

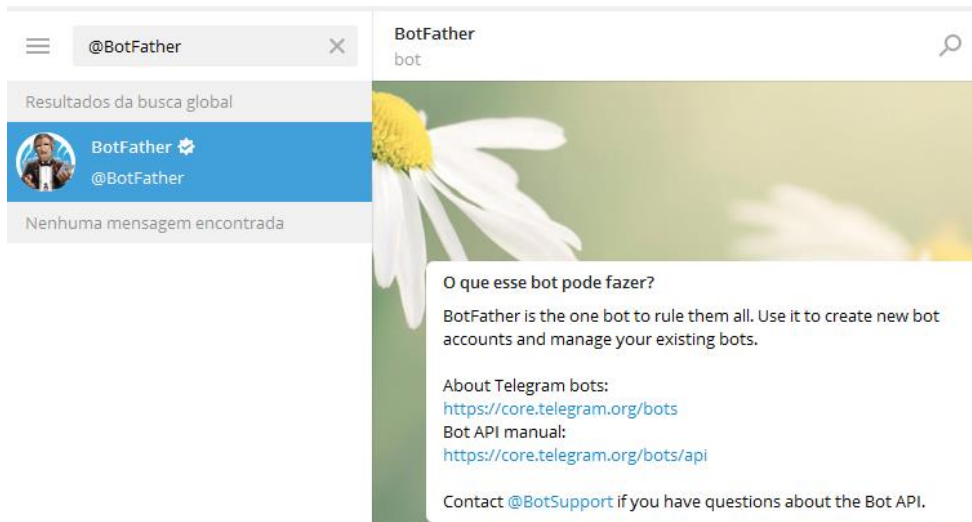
Robô ou bot	Descrição	Serviço	Parâmetros
gif	Busca e envia GIFs animados	giphy.com	
vid	Compartilha vídeos	youtube.com	
bing	Busca e envia imagens	bing.com	
pic	Busca e envia imagens	yandex.com	
wiki	Compartilha artigos da Wikipedia	en.wikipedia.org	Use wiki pt para ver em português
imdb	Compartilha 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/>

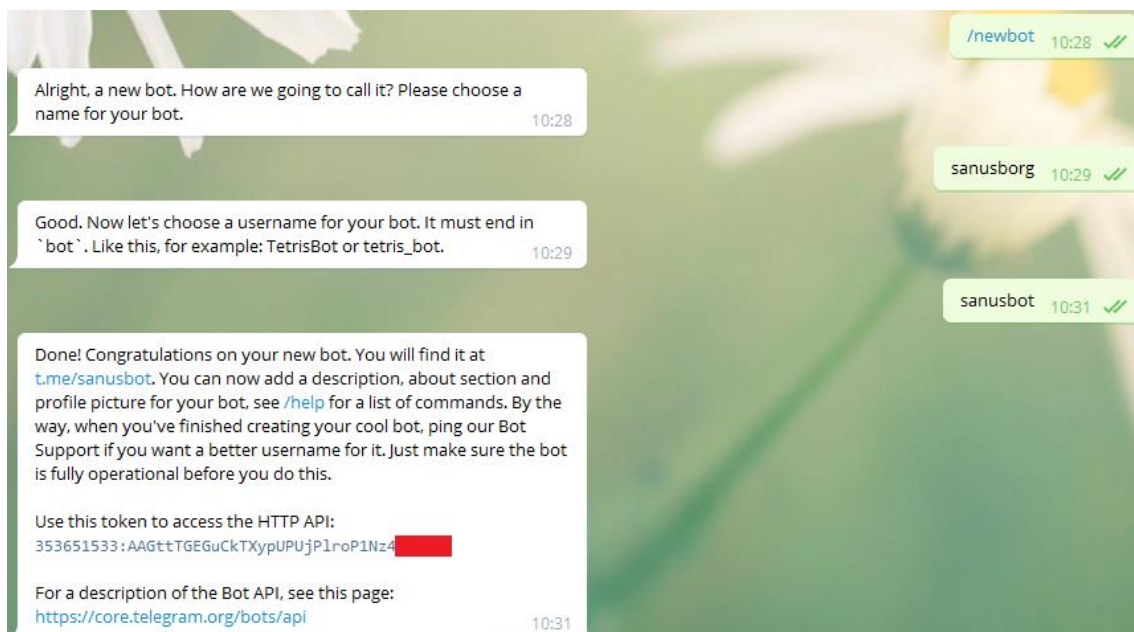
Criando um Bot

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[] = "xxxxxxxxxxxx";
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", "");
    }
}

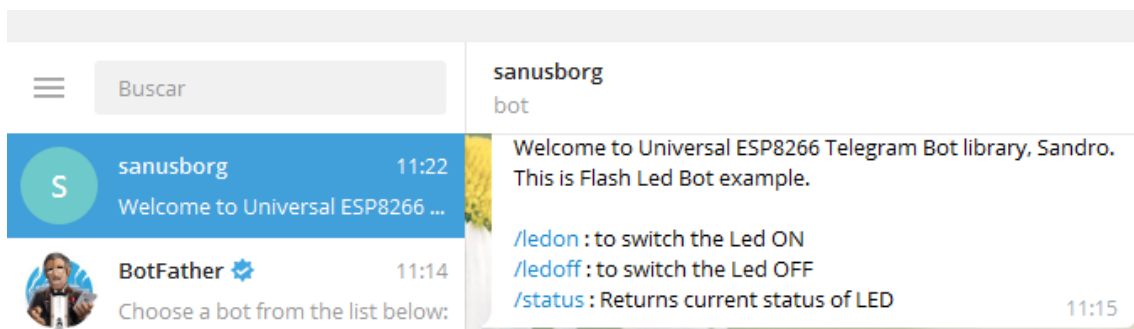
if (text == "/start") {
    String welcome = "Welcome (/start) to SanUSB ESP8266 Telegram Bot library, " + first_name + ". Click /ledon
: to switch the Led ON. /ledoff : to switch the Led OFF. /status : Returns current status of LED.";
    bot.sendMessage(chat_id, welcome, "");
}
}
bot.message[0][0] = ""; // 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.



A Bot API 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, *utilizado por aplicações javascript*, e **Webhooks** do outro, que é um simples evento de notificação assíncrona via HTTP POST *para aplicações em PHP*, por exemplo. 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 javascript JSON.

Exemplos de outros comandos da API:

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

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

Exemplos:

<https://api.telegram.org/bot353651533:AAGttTGEguCkTXypUPUjPlroP1Nz4vKjNYc/getUpdates>

sendMessage: Enviando Mensagens.

Através do token obtido, basta construir a URL com **sendMessage** abaixo:

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

https://api.telegram.org/bot353651533:AAGttTGEguCkTXypUPUjPlroP1Nz4vKjNYc/sendMessage?chat_id=309616823&text=Hello

Para enviar via terminal com CURL:

curl -X POST "https://api.telegram.org/botXXX:YYYY/sendMessage" -d "chat_id= yyyyyyy &text=my sample text"

[curl -X POST
"https://api.telegram.org/bot353651533:AAGttTGEguCkTXypUPUjPlroP1Nz4vKjNYc/sendMessage?chat_id=309616823&text=my sample text"](https://api.telegram.org/bot353651533:AAGttTGEguCkTXypUPUjPlroP1Nz4vKjNYc/sendMessage?chat_id=309616823&text=my sample text)

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

Exemplo:

<https://api.telegram.org/bot353651533:AAGttTGEguCkTXypUPUjPlroP1Nz4vKjNYc/getMe>

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](#).

Utilizando o Webhook

Os bots em PHP que interagem com os usuários através do Webhook do telegram, ou seja, um gancho da Web, que serve para integrar o bot com um código PHP que por sua vez define o comportamento do bot. 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 PHP configurada junto ao Webhook.

O Webhook, ou seja, um gancho da Web serve para integrar o bot com um código php que define o seu comportamento. Dessa forma, quando um comando é enviado ao telegram, a API do Telegram dispara um evento no Webhook, 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 necessário 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 URL. O método setWebhook é configurado pela URL do browser com o seguinte comando:

[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/bot1193063525:AAHdl7KfZ4fefaRZU0cQTbHp85paJJ0c9e4/setwebhook?url=https://xx.sanusb.org/sanusb.php>

<https://api.telegram.org/bot1193063525:AAHdl7KfZ4fefaRZU0cQTbHp85paJJ0c9e4/setwebhook?url=https://xx.sanusb.org/sanusb1.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"}
```

Vale salientar, que por motivo de segurança, normalmente 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/bot399661601:AAEnqDHU1M3LPCiPwN7ZyQfVq6M1CplUlfU/setwebhook?url=>

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

Exemplo de código php para um endereço https:

```
<?php

//set up the Bot API token
$botToken = "xxxxxxxxxx:yyyyyyyyyyyyyyyyyyyyyyyyyyyyyyyy";
$website = "https://api.telegram.org/bot".$botToken;

//Grab the info from the webhook, parse it and put it into $message
$content = file_get_contents("php://input");
$update = json_decode($content, TRUE);
$message = $update["message"];

//Make some helpful variables
$chatId = $message["chat"]["id"];
$text = $message["text"];

$date = date('d/m/y H:i:s');

//send the random Bob wisdom back
if($text === "/usb" || $text === "/start") {

//file_get_contents($website."/sendMessage?chat_id=".$chatId."&text=chat_id= ".$chatId);

file_get_contents($website."/sendMessage?chat_id=".$chatId."&text=data= ".$date);
}
```

TEMPORIZADORES E RELÓGIOS EM TEMPO REAL

Nesse material didático é proposta uma breve revisão sobre os **temporizadores e relógios/calendários em tempo real (RTC)** disponíveis para famílias de microcontroladores ESP32 e ESP8266. Com excessão do exemplo do RTC interno, que

é um periférico disponível apenas no ESP32, todos os outros exemplos didáticos do repositório foram testados e funcionam tanto no ESP32 quanto no ESP8266, sendo uma característica relevante para diversas aplicações reais. Mais detalhes em: https://youtu.be/vyq2sJE_DwM. Para instalar as bibliotecas desse material didático, siga as etapas:

Faça o download do repositório <https://github.com/SanUSB/NTPClient>,

Arduino IDE -> Sketch -> Incluir biblioteca -> Adicionar a biblioteca .zip NTPClient-Master. Para usar, descompacte a pasta .zip e acesse a pasta de exemplos.

Este tópico é muito relevante em projetos práticos reais. Geralmente, em processos automáticas de IoT, os temporizadores e/ou relógios em tempo real (RTC) são usados para as tarefas agendadas dos processos.

Nesse sentido, segundo o próprio github em <https://help.github.com/pt/github/getting-started-with-github/fork-a-repo>, foi bifurcado (*fork*) um projeto e modificado para dar suporte aos exemplos práticos sobre temporizadores e RTCs disponíveis para os dois modelos em <https://github.com/SanUSB/NTPClient>. Todos os exemplos podem ser repetidos por qualquer interessado, bastando somente um computador e um microcontrolador ESP e mais nenhum outro periférico, pois até o LED utilizado é o BULTIN contido na placa do microcontrolador.

Quanto aos **temporizadores** nos microcontroladores, podemos classificá-los como:

- a. **do Hardware:** são periféricos reais contidos no hardware interno dos microcontroladores;
- b. **Emulados por software:** são temporizadores emulados pelo processador dos microcontroladores. A biblioteca Ticker é baseada em temporizadores emulados por software.

Quanto aos **relógios/calendários em tempo real (RTC)** aplicados em sistemas IoT, podemos classificá-los como:

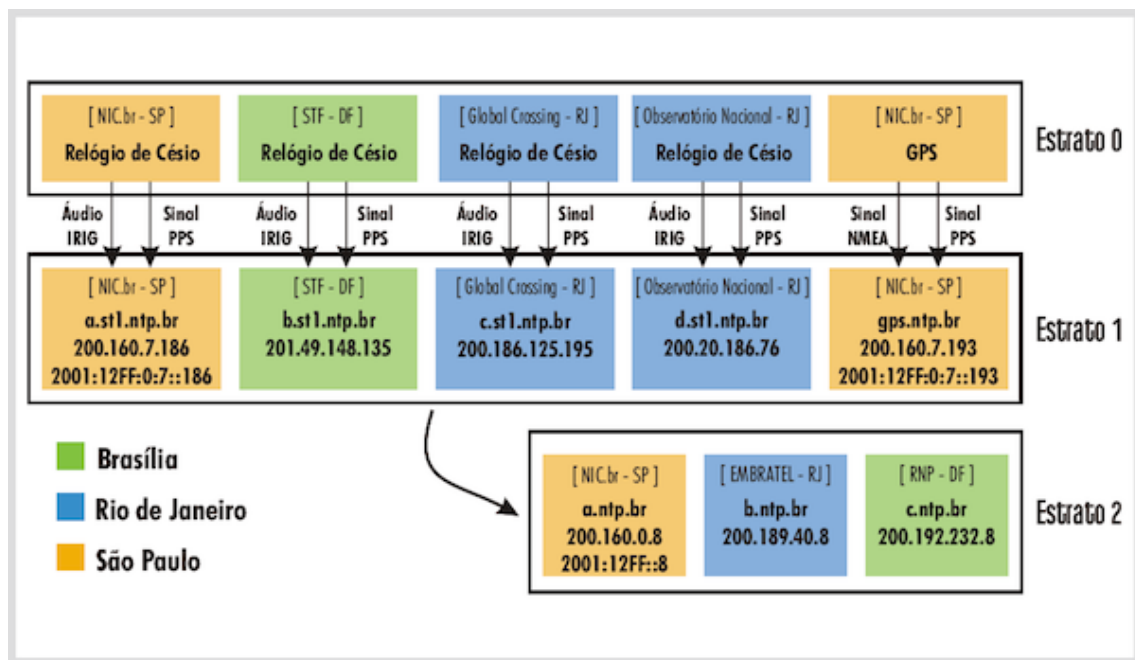
- c. **RTC Online:** Baseado em NTP (*Network Time Protocol*).

O Brasil tem o projeto NTP.br com objetivo de oferecer condições para que os servidores de Internet estejam sincronizados com a Horal Legal Brasileira. Os servidores públicos NTP no Brasil do estrato 2 do NTP.br são **a.ntp.br**, **b.ntp.br** e **c.ntp.br**.

Eles são alimentados por servidores primários (estrato 1), também acessíveis publicamente, entre eles, **a.st1.ntp.br**, **b.st1.ntp.br**, **c.st1.ntp.br** e **d.st1.ntp.br**.

Esses, por sua vez, são sincronizados com relógios atômicos, que são de responsabilidade do Observatório Nacional.

Existe também o servidor **gps.ntp.br**, ilustrado na Figura abaixo, que é utilizado para monitoramento do sistema. É um servidor ntp estrato 1, sincronizado com o Sistema de Posicionamento Global (GPS).



Fonte: <https://ntp.br/estrutura.php>

d. RTC Offline:

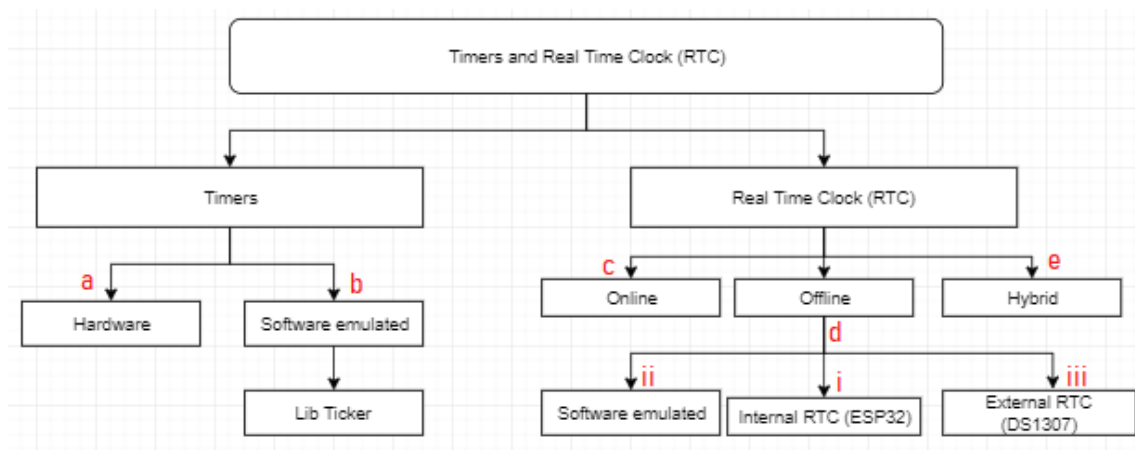
d.i. Utilizando RTC real interno, no caso do ESP32;

d.ii. Emulados via software, como por exemplo, utilizando a biblioteca *ticker* (funcional). Vale salientar que como o ESP8266 não tem um RTC interno, emular um RTC via software pode ser uma possibilidade viável, principalmente para sistemas que necessitam de muita exatidão e nível de segundos, tendo em vista que esse procedimento foi utilizado

em um projeto de controle de acesso RFID de laboratório e funcionou como esperado. Um teste sobre a comparação desse sistema com o NTP é mostrado no vídeo.

d.iii. Utilizando RTC real externo, como por exemplo, o DS1307;

e. RTC híbrido: Que utiliza, por exemplo, o NTP para referenciar e/ou atualizar os relógios em tempo real (RTC) offline. Abaixo, um gráfico com resumo dos tipos sugeridos nessa pesquisa:



Quanto à preservação dos dados do RTC, em caso de queda de energia, se utiliza normalmente:

- **Bateria** em paralelo com a fonte de alimentação do microcontrolador;
- **Memória persistente** para restaurar o relógio com o último horário armazenado até a correção manual ou automática via NTP quando voltar a conexão com a Internet.
- Aplicar **redundância** na atualização de relógio, como por exemplo, o RTC interno do ESP32 em paralelo com NTP. É considerado o maior valor Unix time, ou seja, o sistema mais atual;

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 TEMPORIZADORES DO HARDWARE

Nesse caso, é utilizado um temporizador real interno 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 microssegundo possibilita o processamento de 80 ciclos de instrução.

INTERRUPÇÃO DE TEMPORIZADORES EMULADOS POR SOFTWARE

Esse temporizador é baseado em software e pode suportar a geração de vários timers concorrentes. Por ser emulado por 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 interrupcao, 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 microssegundo pra respirar  
}
```

A BIBLIOTECA DE INTERRUPÇÃO TICKER

Essa Biblioteca para o ESP8266 e ESP32 é baseada em interrupção de temporizadores emulados 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.

A biblioteca Ticker funciona tanto para ESP8266, como para ESP32 e a estrutura da função de configuração pode conter dois ou três parametros:

Ticker ticker; //Declaração do objeto

ticker.attach(tempo, função de interrupção);

```
#include <Ticker.h>  
Ticker ticker;  
  
int LED = 2;  
  
void tick()  
{  
  //toggle state  
  digitalWrite(LED, !digitalRead(LED)); // set pin to the opposite state  
  Serial.print("Interrupcao timer ticker\n");  
}  
  
void setup() {  
  Serial.begin(115200);  
  pinMode(LED, OUTPUT);  
  ticker.attach(1, tick); //Função tick  
}  
  
void loop() {
```

```
}
```

ticker.attach(tempo, função de interrupção, parâmetro da função);

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(2, state);
  Serial.print("Interrupcao ticker: ");
  Serial.println(state);
}

void setup() {
  Serial.begin(115200);
  pinMode(2, OUTPUT);
  digitalWrite(2, LOW);

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

  // a cada 1500 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 pelo watchdog
  delay(1000);
}
```

RTC INTERNO NO ESP32

O ESP32 contém um RTC (relógio em tempo real interno), que pode ser setado com Unix time, ou seja, a contagem de segundos desde 01/01/1970. Esse relógio também necessita de uma bateria ou uma memória persistente dedicada em caso de queda da alimentação, pois, caso contrário, a hora irá se perder ou congelar. Exemplo:

```
#include <time.h>
#include <sys/time.h>
struct tm data; //Cria a estrutura que contem as informacoes da data.
/*
Unix time é a contagem de segundos desde 01/01/1970.
*/
void setup()
{
  timeval tv; //Cria a estrutura temporaria para funcao abaixo.
  tv.tv_sec = 1591037364; //Atribui minha data atual. Voce pode usar o NTP
  settimeofday(&tv, NULL); //Configura o RTC para manter a data atribuida atualizada.
}

void loop()
{
```

```
delay(1000);
time_t tt = time(NULL); //Obtem o tempo atual em segundos. Utilize isso sempre que precisar obter o tempo atual
data = *gmtime(&tt); //Converte o tempo atual e atribui na estrutura

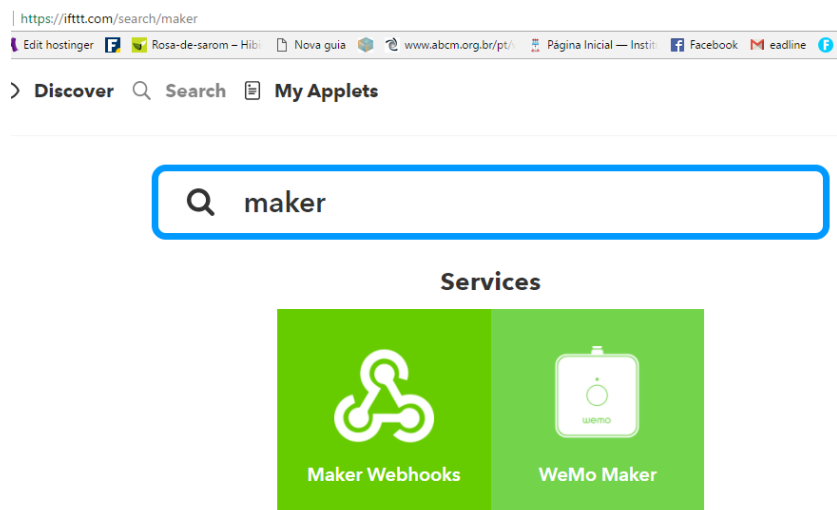
char data_formatada[64];
strftime(data_formatada, 64, "%d/%m/%Y %H:%M:%S", &data); //Cria uma String formatada da estrutura "data"
printf("\nUnix Time: %d\n", int32_t(tt)); //Mostra na Serial o Unix time
printf("Data formatada: %s\n", data_formatada); //Mostra na Serial a data formatada

if (tt >= 1591037374 && tt < 1591037379) //Use sua data atual, em segundos, para testar o acionamento por datas e
horarios
{
    printf("Alarm!!!\n");
}
}
```

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 listado (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>

Edit hostinger Rosa-de-sarom – Hibi Nova guia www.abcm.org.br/pt/ Página Inicial — Instit. Facebook eadlin

Complete trigger fields

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:

Complete action fields

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 um Kit 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.

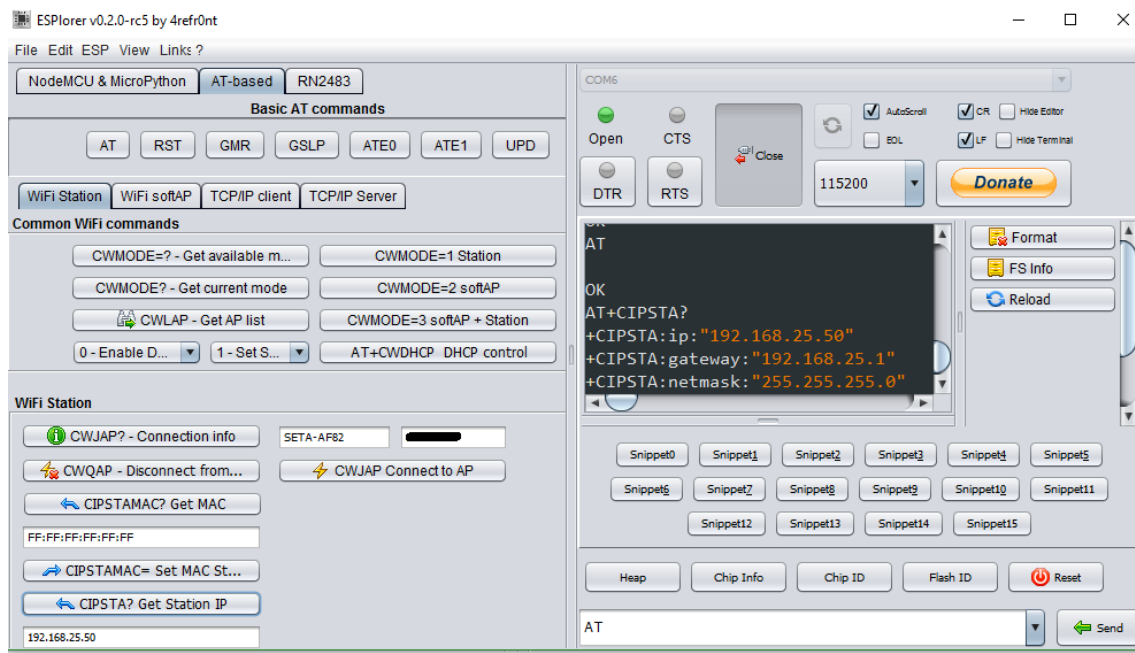
ESPLORER

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 printscreen 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

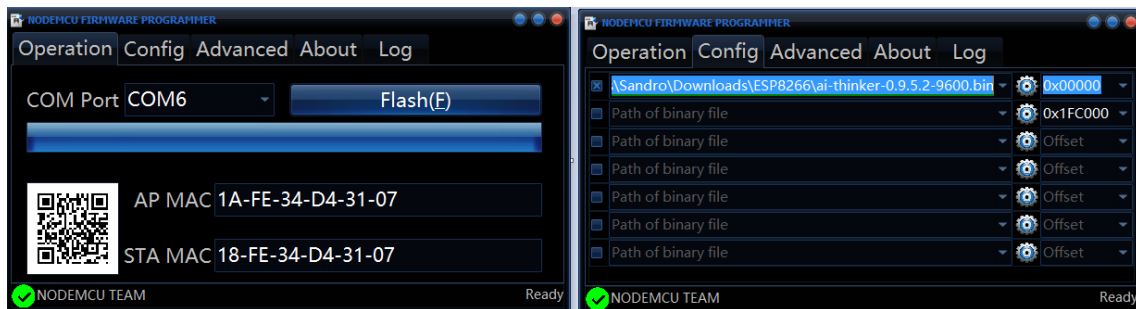
Se você precisar restaurar o firmware do ESP8266, siga as instruções abaixo:

4) Faça o download do software flasher <https://github.com/nodemcu/nodemcu-flasher>

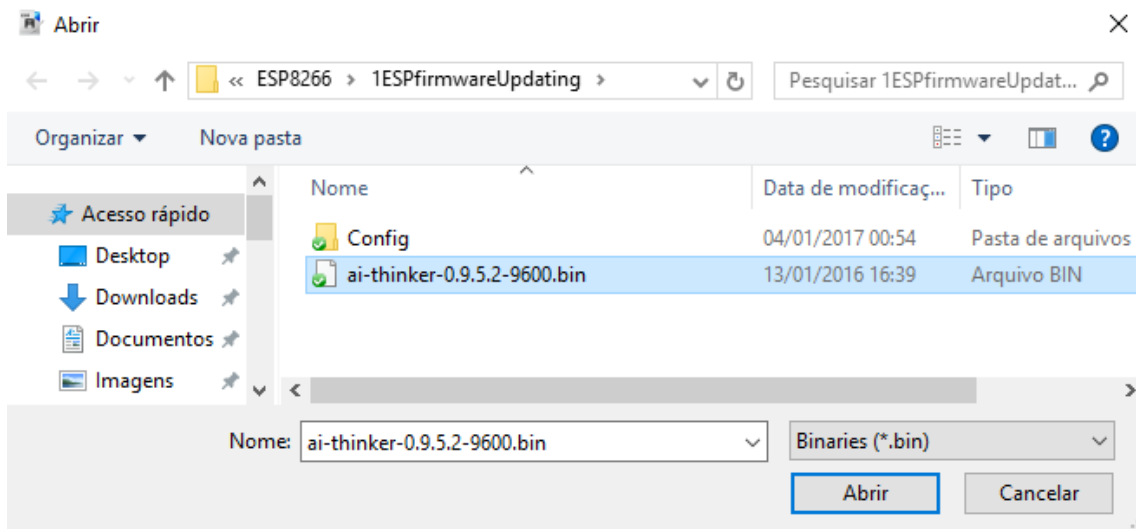
baixe a pasta com o executável e o binário de comandos AT em sanusb.org/esp/flasher/FirmwareAT.zip.

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".

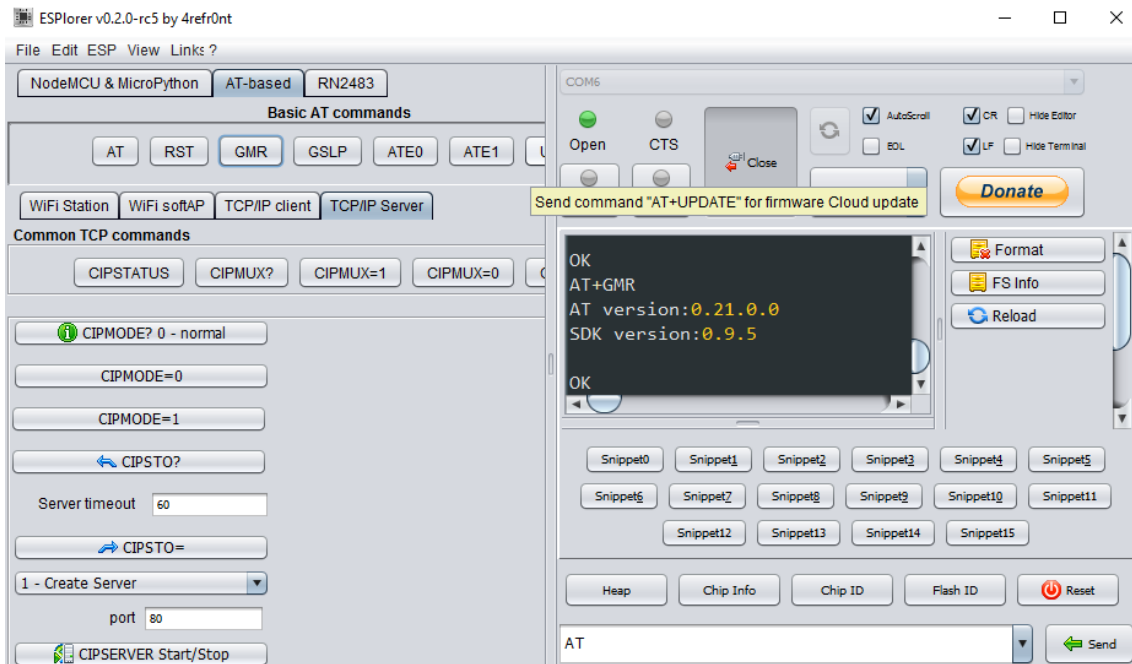


9) Clique na Aba Operation, Selecione a porta COM e clique no botão "Flash":

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 THINKSPEAK 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.scv> (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","sanusblaese"

AT+CIPMUX=1

AT+CIPSTART=4,"TCP","184.106.153.149",80 //api.thingspeak.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

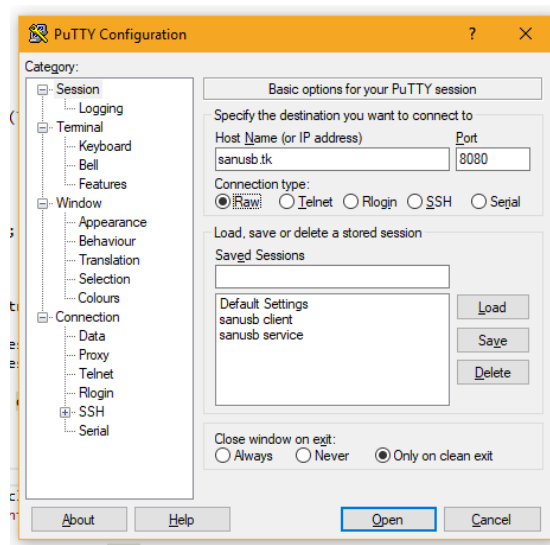
O IoT push se trata de um software instalado em um servidor. Esse software escuta conexões de clientes denominados dispositivo e serviço, e permite que serviços enviem mensagens para dispositivos que são identificados por códigos únicos (*strings*). Via TCP/IP, os serviços se conecta via na porta 8081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

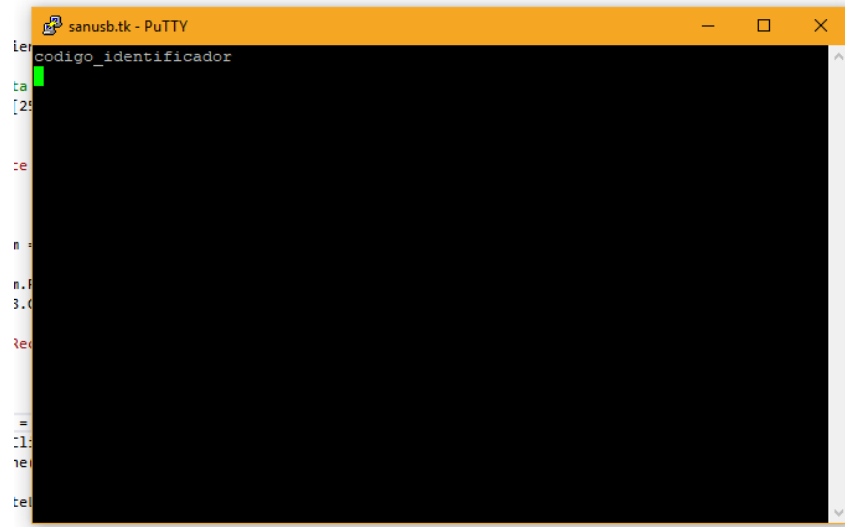
Para conexão é necessário seguir os passos a seguir:

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

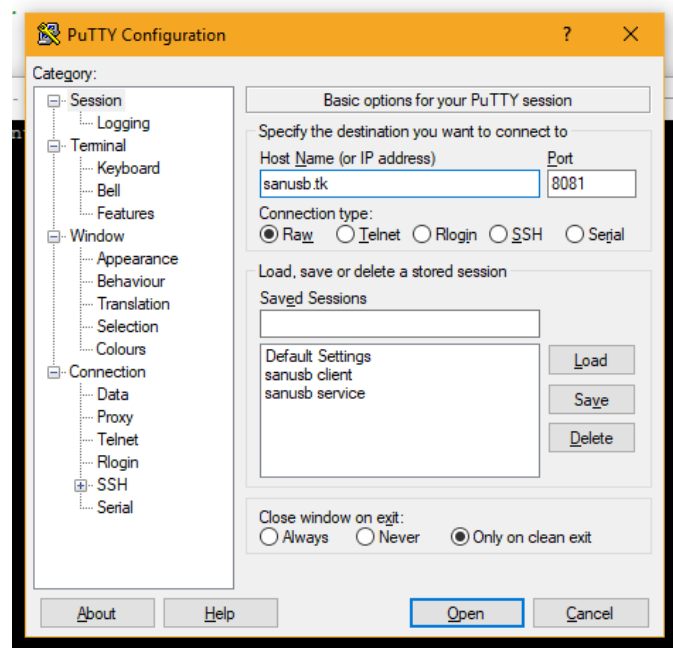


Logo ao se conectar na porta 8080, o dispositivo deve se identificar (*device_id*) enviando somente uma string única (no caso do Putty, digite a string e pressione

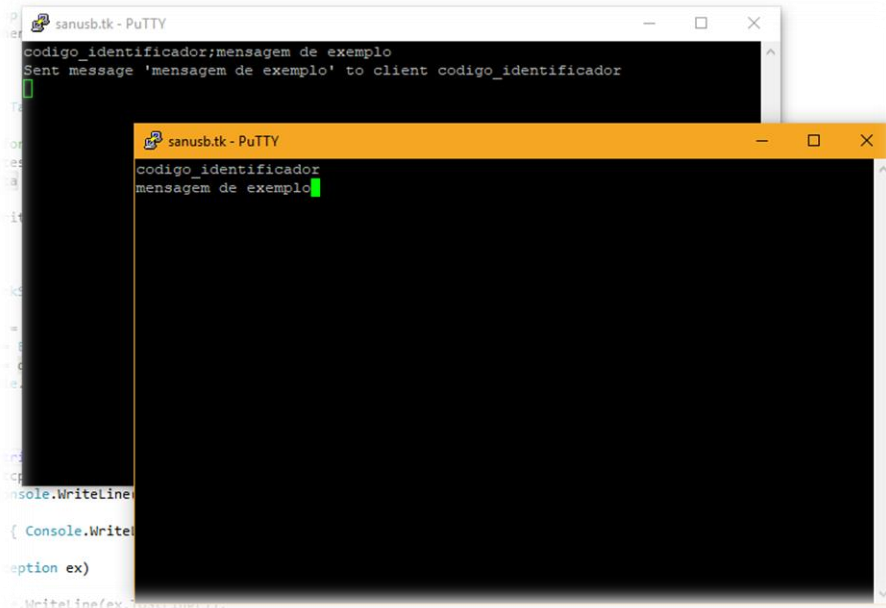
Enter). Obs.: Caso a string não seja única, a operação vai resultar na desconexão de um dispositivo previamente identificado por essa mesma string.



2. 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.



3. 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:

- plataformas Windows, Ubuntu e Mac 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;

```

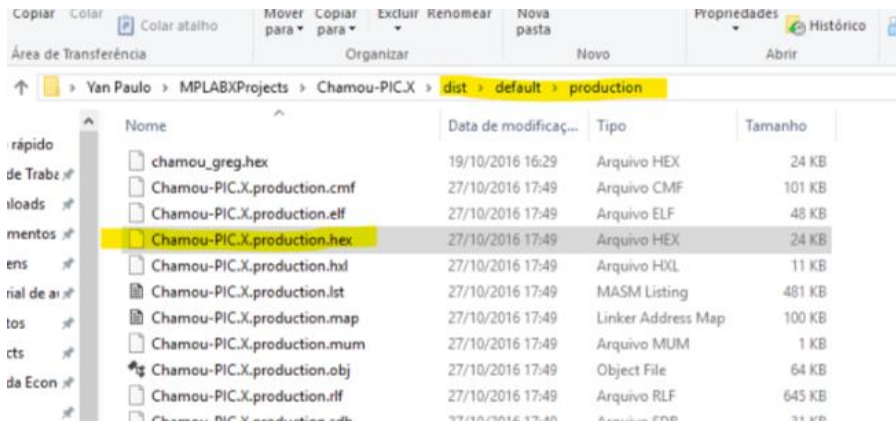
5). (Opcional) O comportamento padrão do projeto é receber o conteúdo e exibir em um display LCD conectado ao microcontrolador. Caso deseje alterar este comportamento, pule para a linha 124 e altere o código pelo que for conveniente. O trecho a ser alterado é exibido na figura a seguir:

```

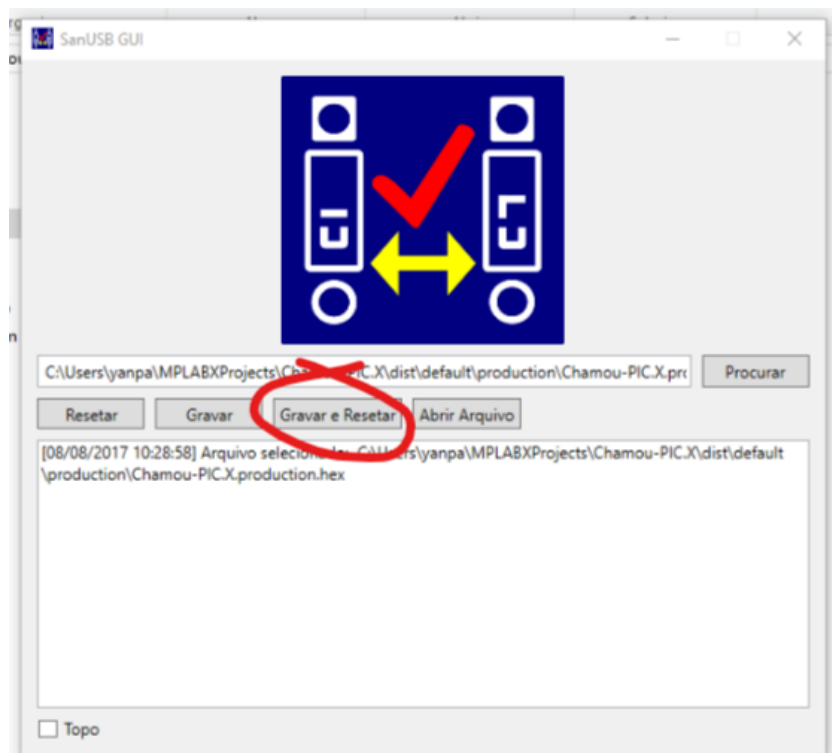
120     blink = 5;
121
122     str = strtok(NULL, ";");
123     /*-----'str' contem o texto recebido do servidor-----*/
124     /*-----Manipule 'str' aqui-----*/
125     Lcd_Cmd(LCD_CLEAR);
126
127     lcd_pos = size <= 18 ? (19 - size) / 2 : 1;
128     lcd_escreve(1, lcd_pos, str);
129     /*-----Pare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

```

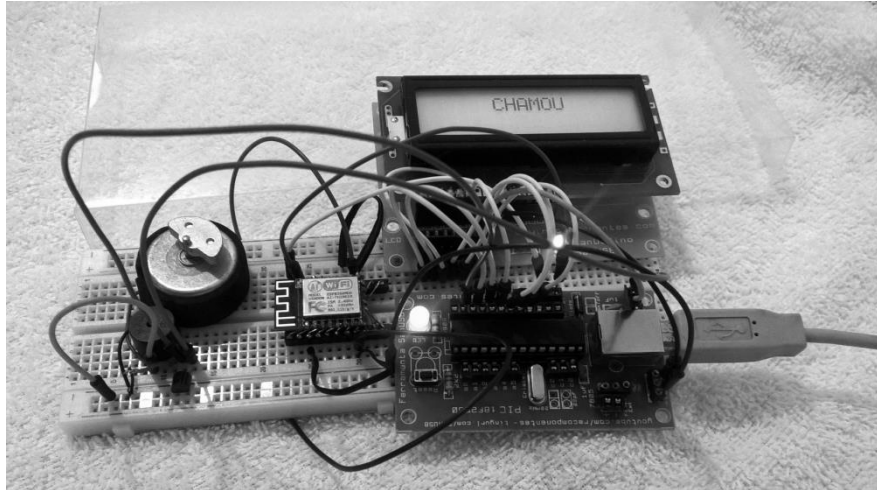
6). Compile o projeto e localize o arquivo “Chamou-PIC.X.production.hex” no diretório “dist/default/Production” do projeto (veja figura a seguir).



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 reinicializar, 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 portátil 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

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

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

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

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

ESP32 – Passos com IDE Arduino:

Passos com IDE Arduino:

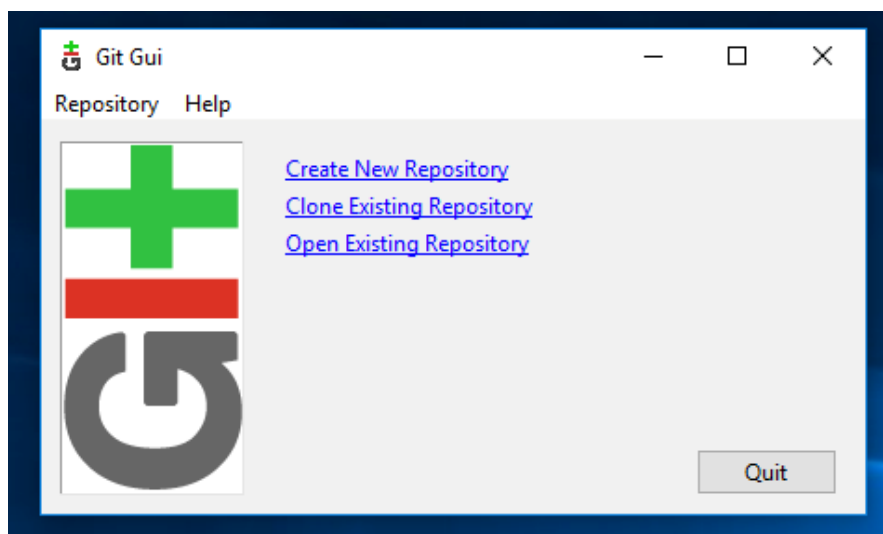
1. Abra a IDE Arduino;

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

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

No caso do sistema operacional Windows:

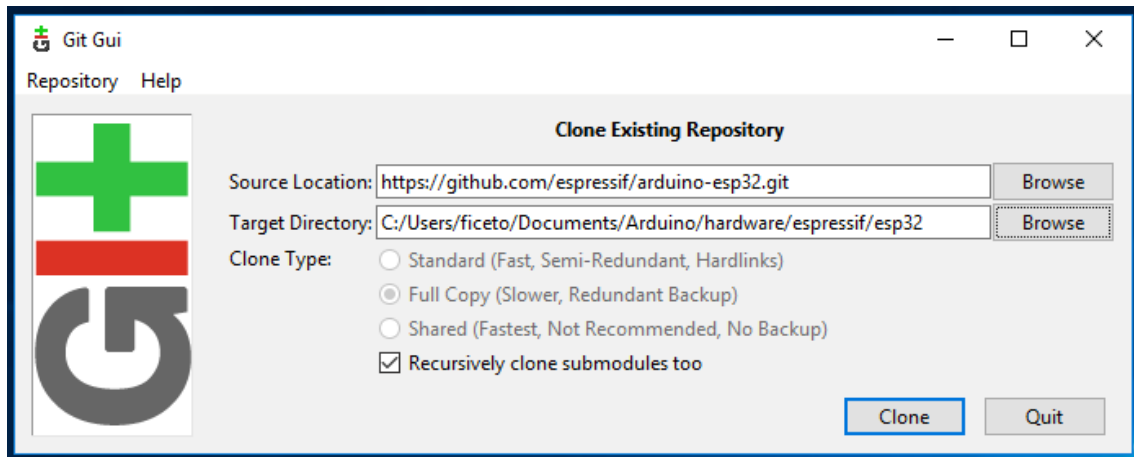
- Baixe e instale o Git de <https://git-scm.com/download/win>
- Inicie o Git GUI ou digite git gui no prompt e execute as seguintes etapas:
- Selecione o item de clone repositório existente (*Clone Existing Repository*):



- Selecione a origem e o destino:

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

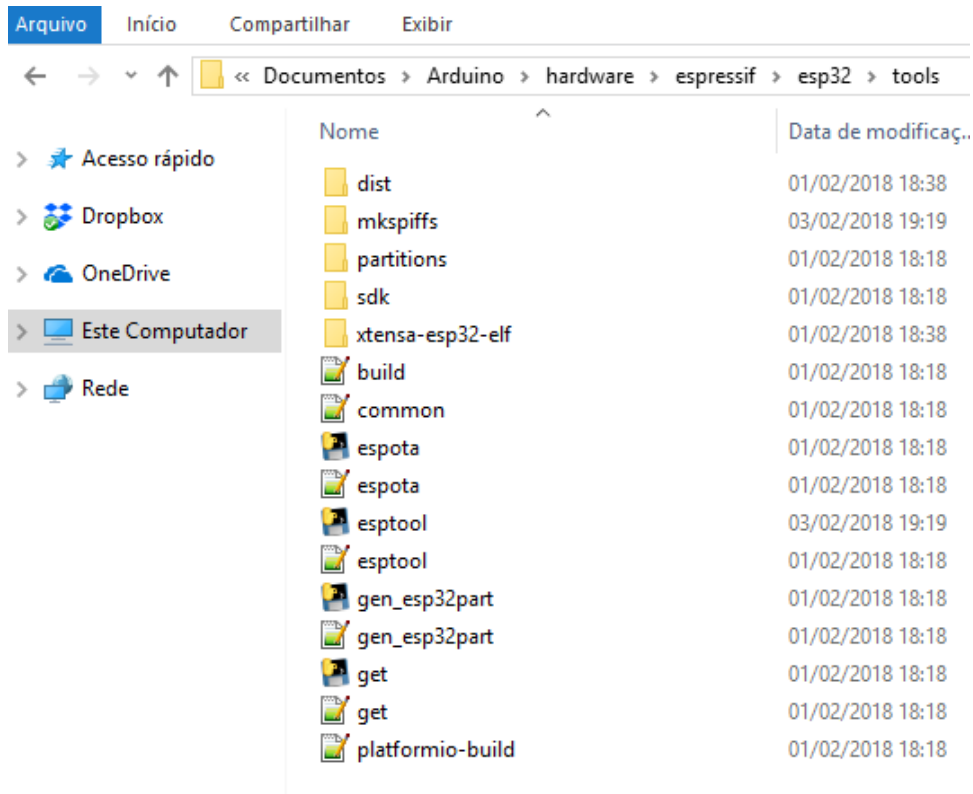
Destino: *C:\Users\[usuário]\Documents\Arduino\hardware\espressif\esp32*



- 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\sand\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`:



2. 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>

3. 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 default na placa ESP32 DEVKIT  
}  
void loop() {  
    digitalWrite(2, HIGH);  
    delay(500);  
    digitalWrite(2, LOW);  
    delay(500);  
}
```

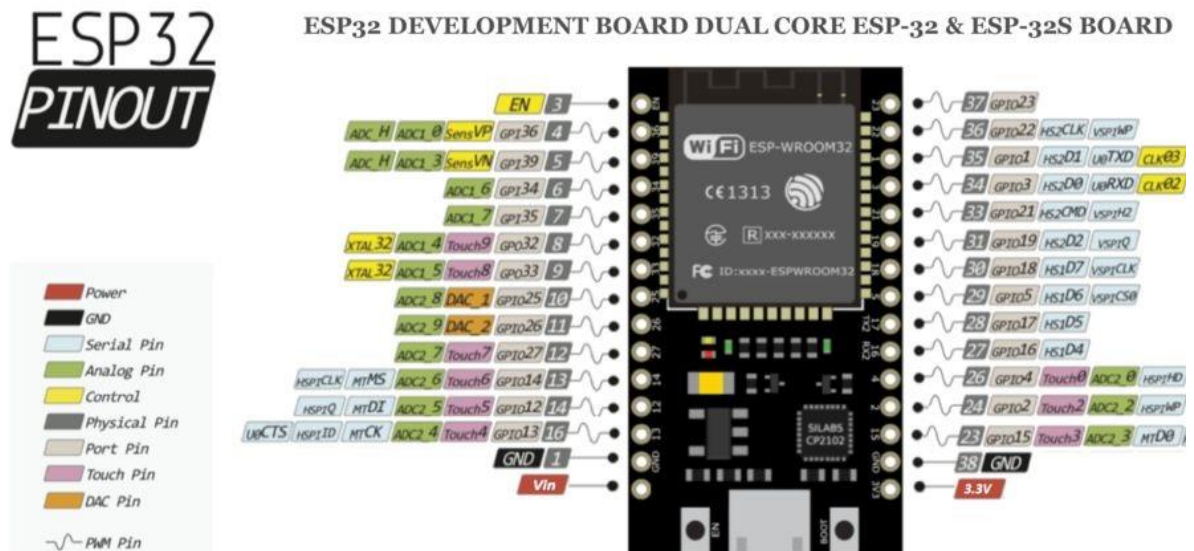

- Selecione Ferramentas -> Placa DOIT ESP32 DEVKIT1

- 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 ser 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()**.



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
10  delay(500);
11  digitalWrite(2, LOW);  // turn
12  delay(500);
13  Serial.write("Oi\n");
14 }

```

de suporte para implementação de sistema operacional em tempo real FreeRTOS.

Instalação do Toolchain esp32 para Windows

Outra forma de compilar para o esp32 é 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

```
~ /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
$ |
```

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

```
~ /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`

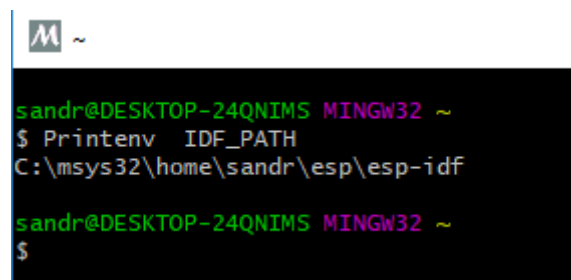
Usando qualquer editor (no meu caso, usei NotePad++), digite o comando **export** abaixo *no script* do arquivo, por exemplo:

```
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 ~
$
```

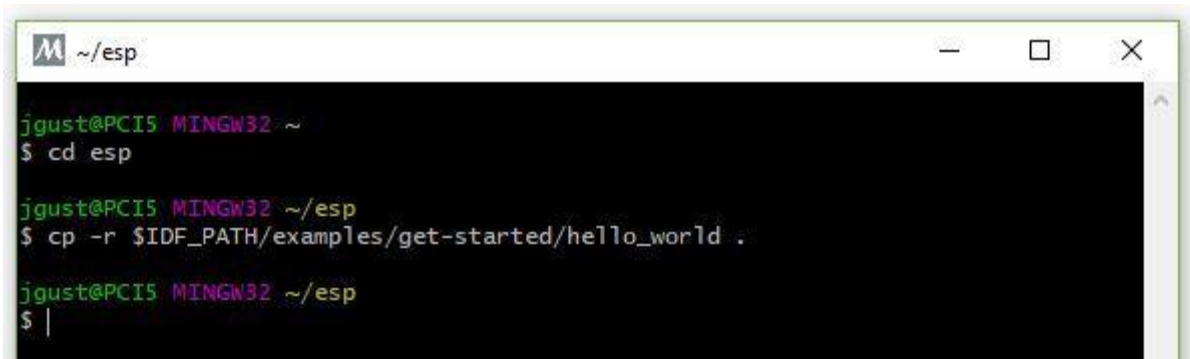
Testando um Projeto exemplo

O ambiente no seu PC, esta 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 .
```



```
~ /esp
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

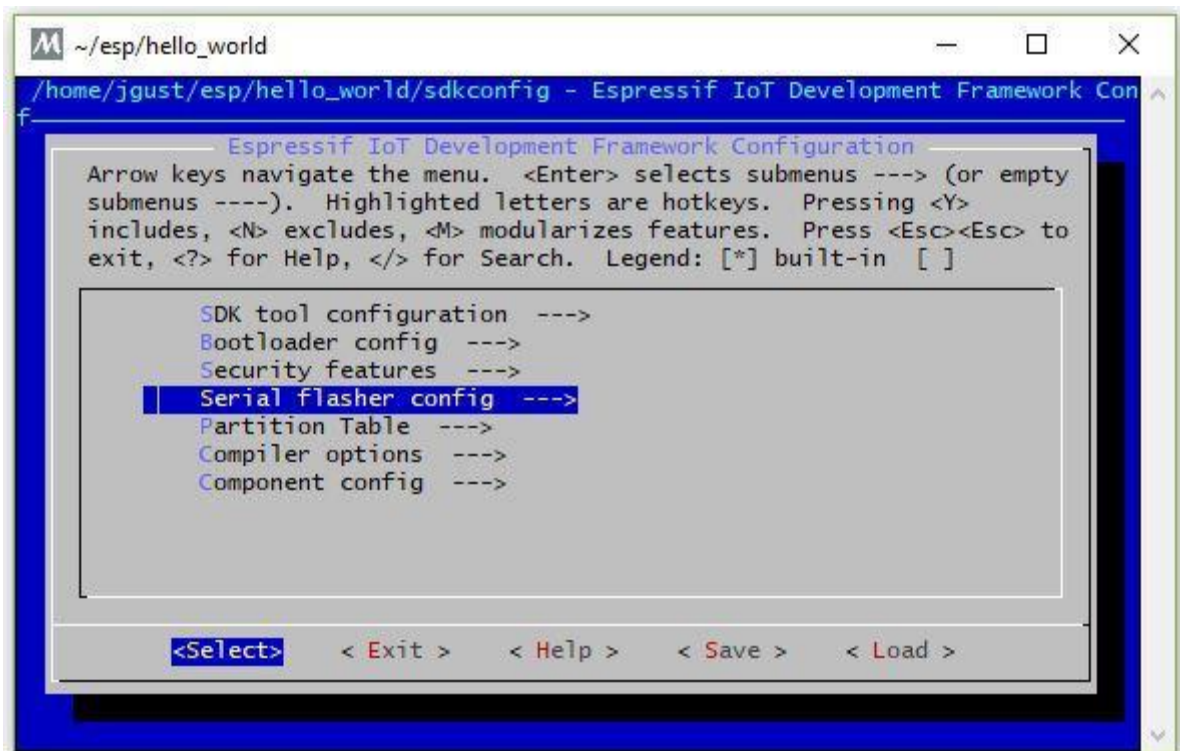
make menuconfig



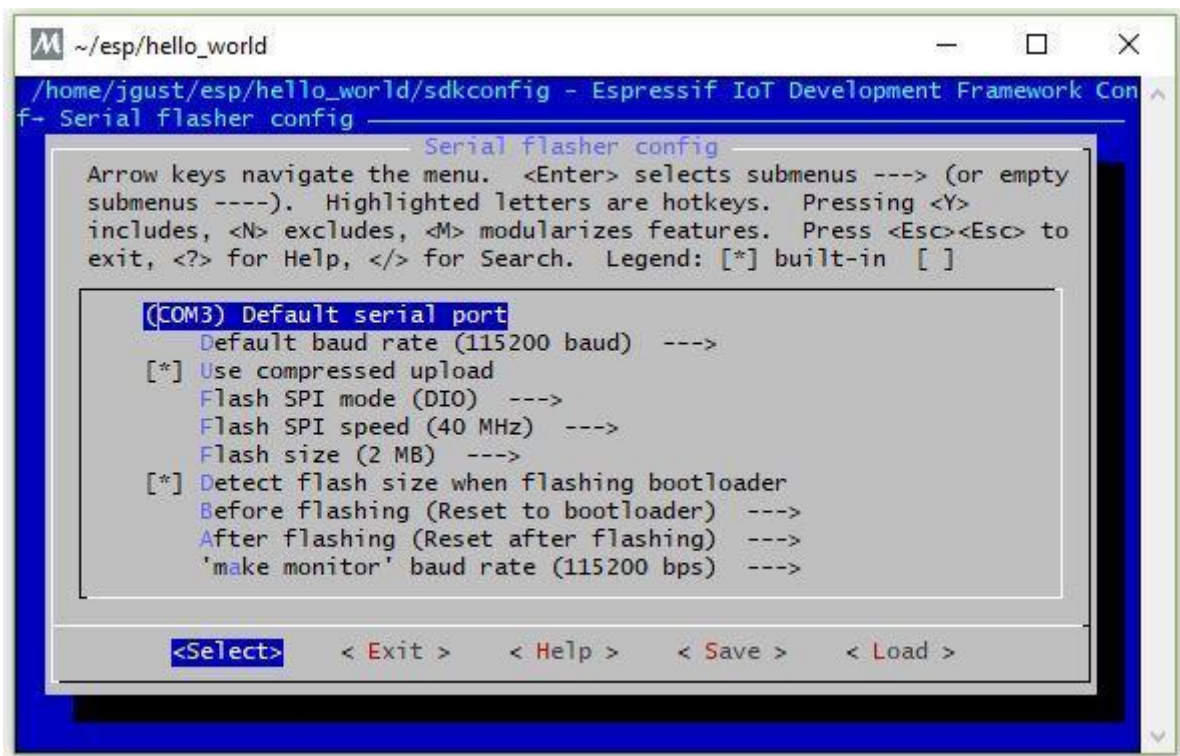
```
~ /esp/hello_world
jgust@PCI5 MINGW32 ~
$ cd ~/esp/hello_world

jgust@PCI5 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)



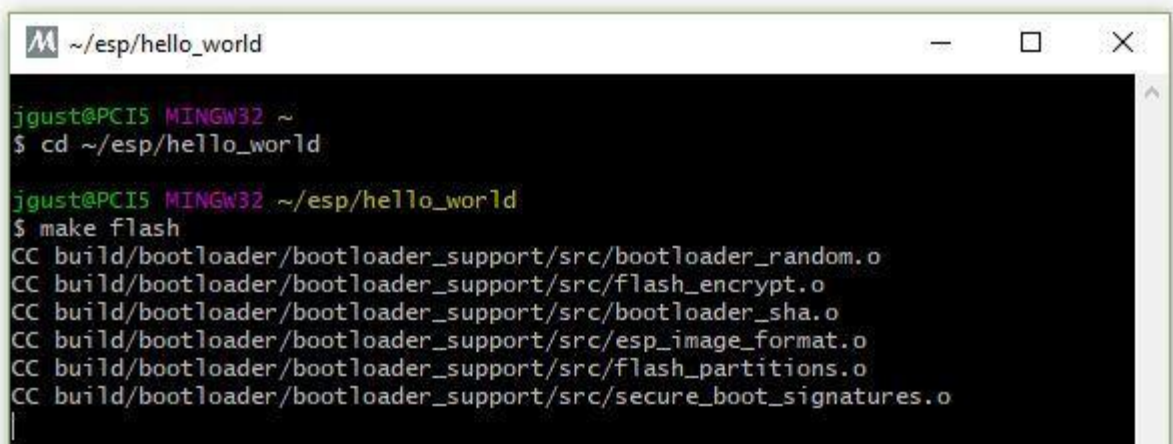
Depois selecione **Exit** duas vezes e **Salve** a configuração .

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.

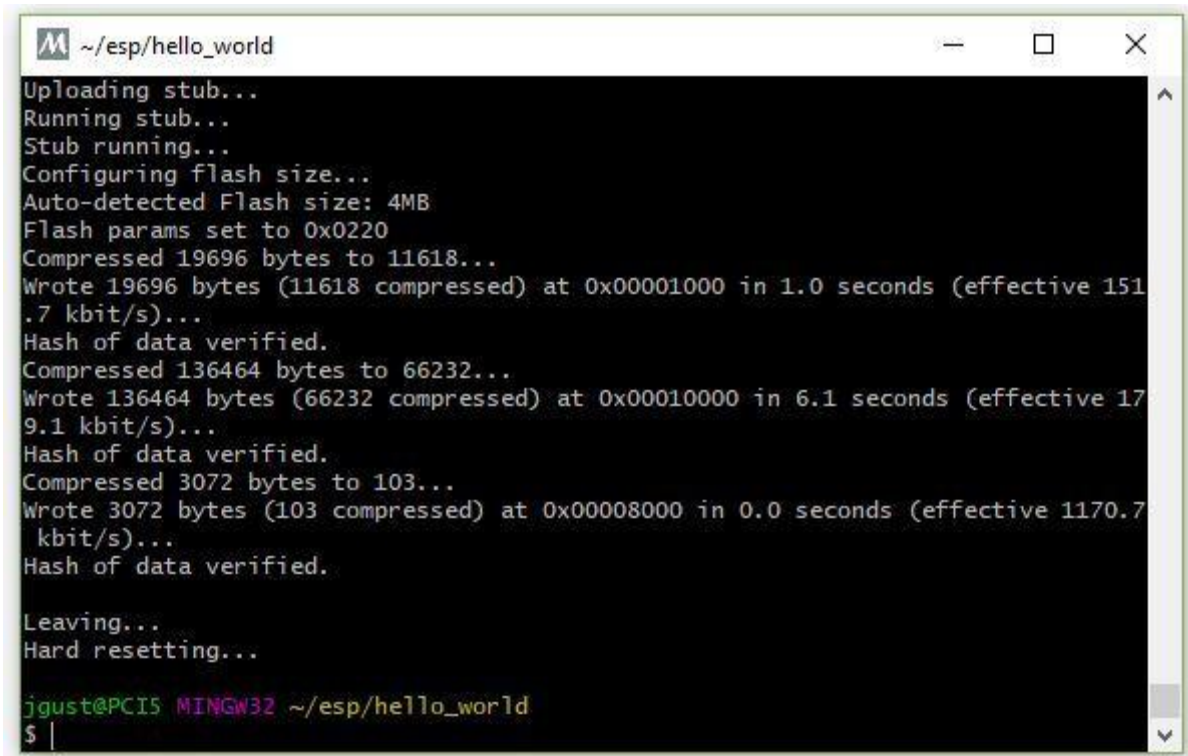
```
cd ~/esp/hello_world
```

```
make flash
```

A screenshot of a terminal window titled '~ /esp/hello_world'. The prompt is 'jgust@PCI5 MINGW32 ~'. The user enters '\$ cd ~/esp/hello_world'. The prompt changes to 'jgust@PCI5 MINGW32 ~/esp/hello_world'. The user enters '\$ make flash'. The terminal shows the compilation of several files: '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', and 'CC build/bootloader/bootloader_support/src/secure_boot_signatures.o'. The list is truncated with a vertical bar at the end.

```
~ /esp/hello_world
jgust@PCI5 MINGW32 ~
$ cd ~/esp/hello_world
jgust@PCI5 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.



```
~/esp/hello_world
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...

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

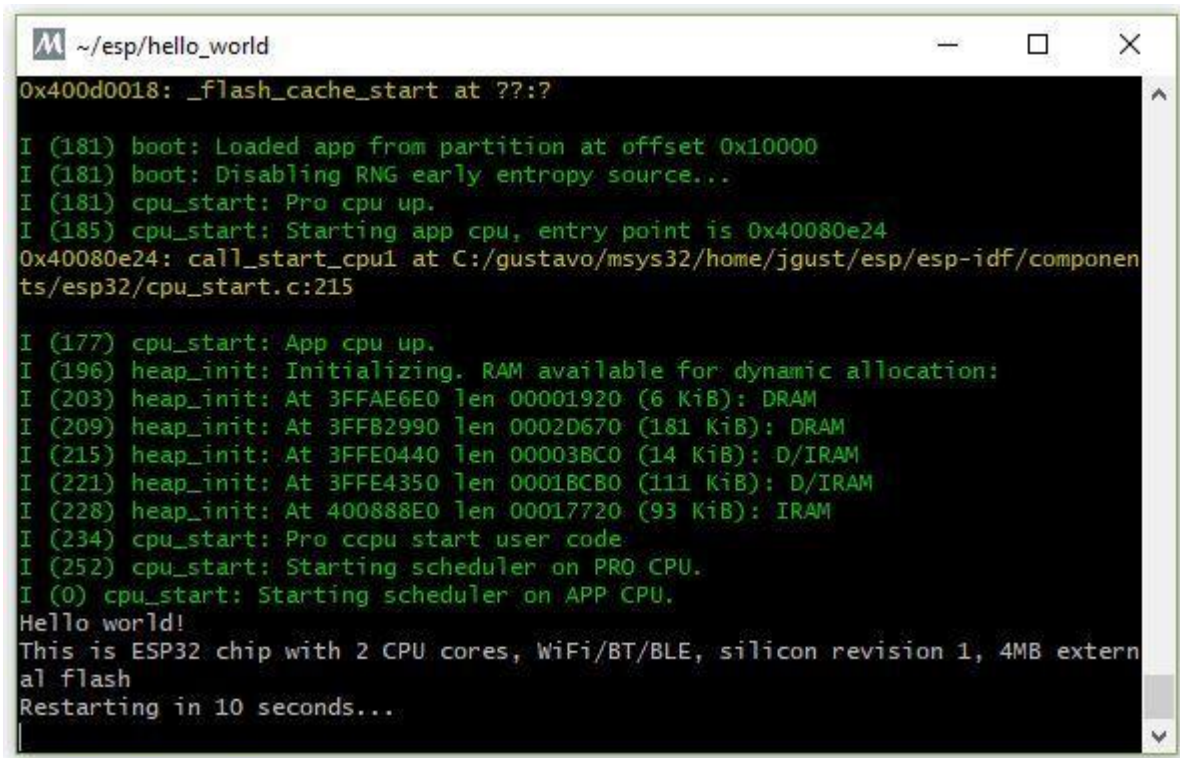
Pronto, a aplicação já está gravada na Flash do ESP32. Agora vamos acessar a porta serial-USB , através de uma console (pode usar um outro programa – velocidade 115200 Bps). Na janela do **mingw32**, dê o comando para abrir a console da IDF:

make monitor



```
~/esp/hello_world
jgust@PCI5 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 .



```
~/esp/hello_world
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:

Blog SanUSB. <https://sanusb.blogspot.com/>

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

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

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

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

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

Minatel, P. (2016) “Arquivos sobre ESsp8266”, <http://pedrominatel.com.br>

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