

ARDUINO

Internet das Coisas ESP8266 e ESP32

Professor: Austeclynio Pereira - 2023

Bibliografia



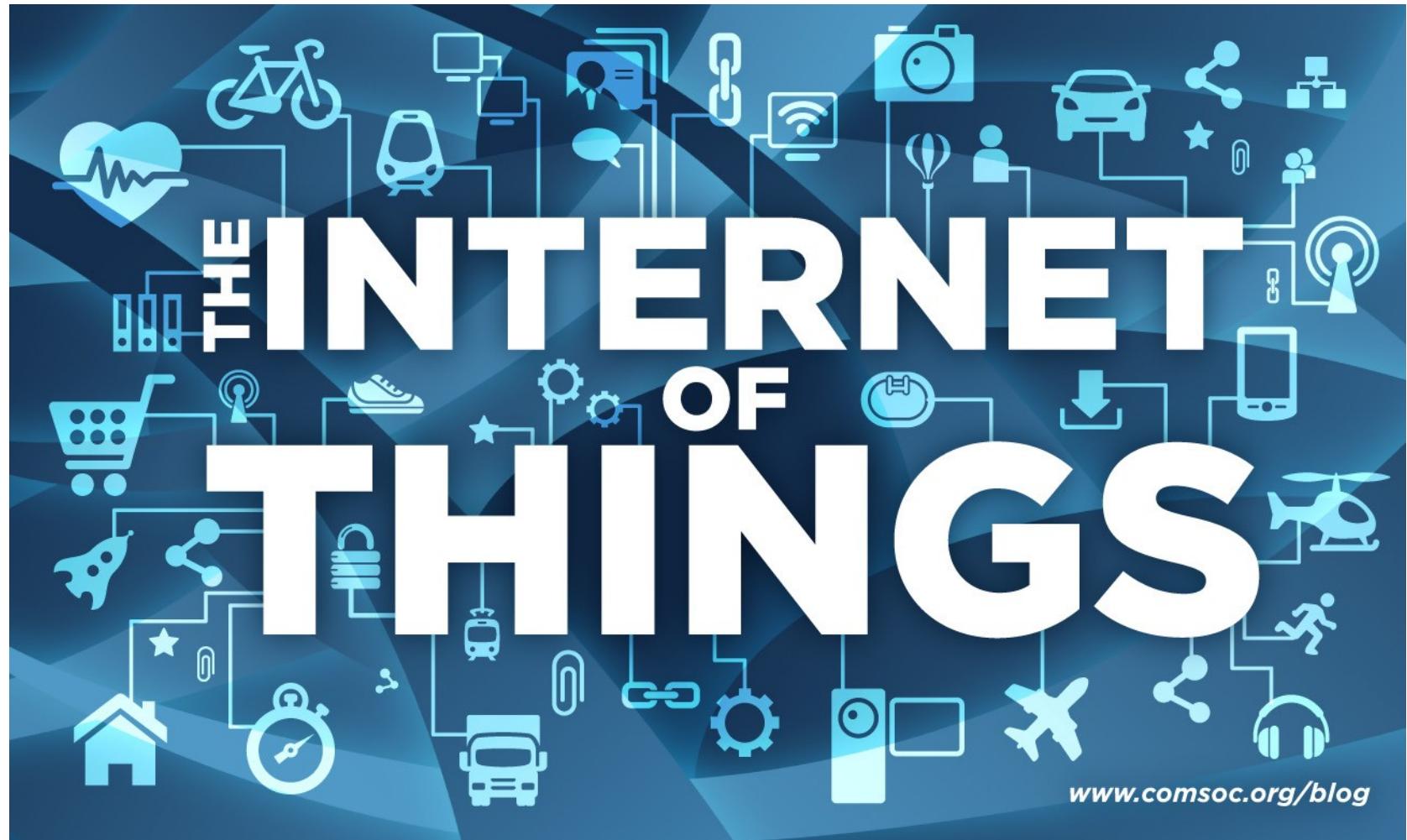
1. *Exploring Arduino* – Blum, Jeremy. Wiley, 2013
2. *Building Internet of Things With the Arduino* – Doukas, Charalampos. CreateSpace Publishing, 2012
3. *Internet of Things with Arduino* – Schwartz, Marco. CreateSpace Publishing, 2015
4. *The Internet of Things* – International Telecommunication Union, 2005
5. ESP8266 Arduino IDE Guide: Internet Of Things With ESP8266(NodeMCU) - Magesh Jayakumar. Bytestem, 2016
6. www.espressif.com/en/products/hardware/esp32/overview
7. www.espressif.com/sites/default/files/documentation/esp32_datasheet_en.pdf
8. www.espressif.com/sites/default/files/documentation/esp32-wroom-32d_esp32-wroom-32u_datasheet_en.pdf

Internet das Coisas ESP8266 e ESP32



- Início em 18/04/2023;
- Término em 30/05/2023;
- Horário: das 9h às 12h;
- Dias da semana: 3as e 5as;
- Conteúdo do curso:
Será enviado para o e-mail dos participantes
- Avaliação – Projetos ao longo e ao final do curso;

Internet das Coisas



www.comsoc.org/blog

IoT - Introdução



- É uma grande rede de coisas interconectadas com capacidade de coletar e trocar dados;
- Existem divergências sobre a autoria do termo;
- Uns atribuem ao norte-americano Peter T. Lewis, que teria cunhado o termo em 1985 em um artigo técnico;
- Outros ao britânico Kevin Ashton, que teria cunhado em 1999 em uma palestra na Procter and Gamble;
- Também chamada de Internet dos Objetos ;

IoT - Introdução



- Engloba a conexão entre:
 - Máquinas-Máquinas (M2M)
 - Pessoas-Pessoas (P2P)
 - Pessoas-Máquinas(P2M)
- A regra é: qualquer coisa que pode ser conectada, será conectada;
- Segundo a Cisco, em 2020 haveria 50 bilhões de dispositivos conectados;

IoT - Introdução



- Alguns estimavam este número em 100 bilhões de dispositivos;
- A internet é considerada uma das mais importantes e poderosas criações de toda a história da humanidade;
- Surge na década de 60 para interligar computadores de universidades americanas ;
- Nas décadas de 70 e 80 é dominado pelos serviços de *e-mail* e *file transfer*. Passa a ser usada por milhares;
- Na década de 90 surgem os navegadores. Passa a ser usada por milhões;
- Na década de 2000, com o advento dos telefones móveis, passa a ser usada por bilhões;

IoT - Introdução



- Impactou:
 - A educação
 - A comunicação
 - Os negócios
 - A ciência
 - O governo
 - A humanidade
- Os *smartphones* potencializaram o uso da internet pela ubiquidade;
- Redes estão disponíveis em qualquer lugar e em qualquer tempo;
- IoT representa a próxima evolução da internet;
- Integrante da 4a revolução industrial junto a IA, robótica etc.
- Habilidade para obter, analisar e distribuir dados que podem ser transformados em informação e conhecimento;

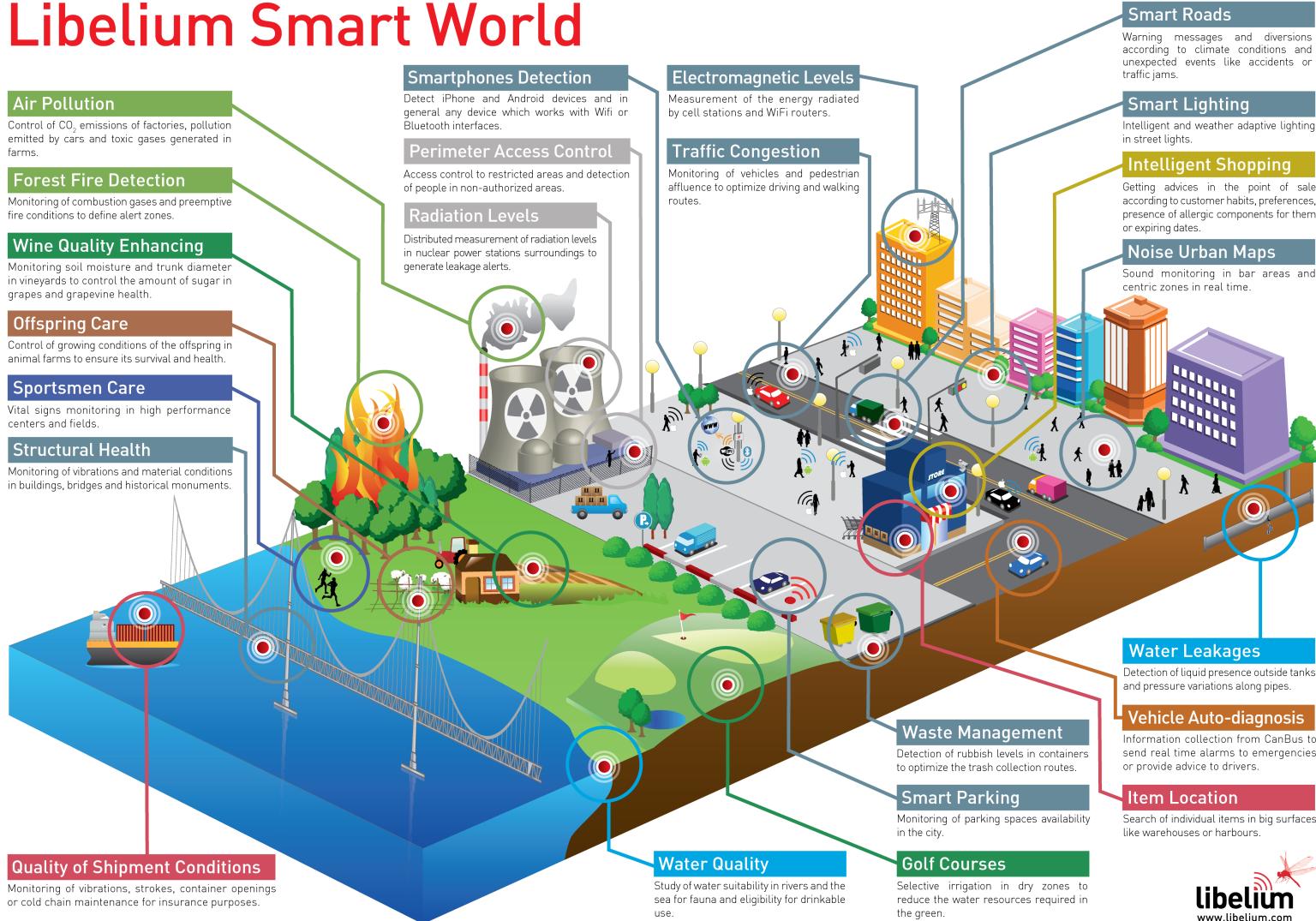
IoT - Introdução



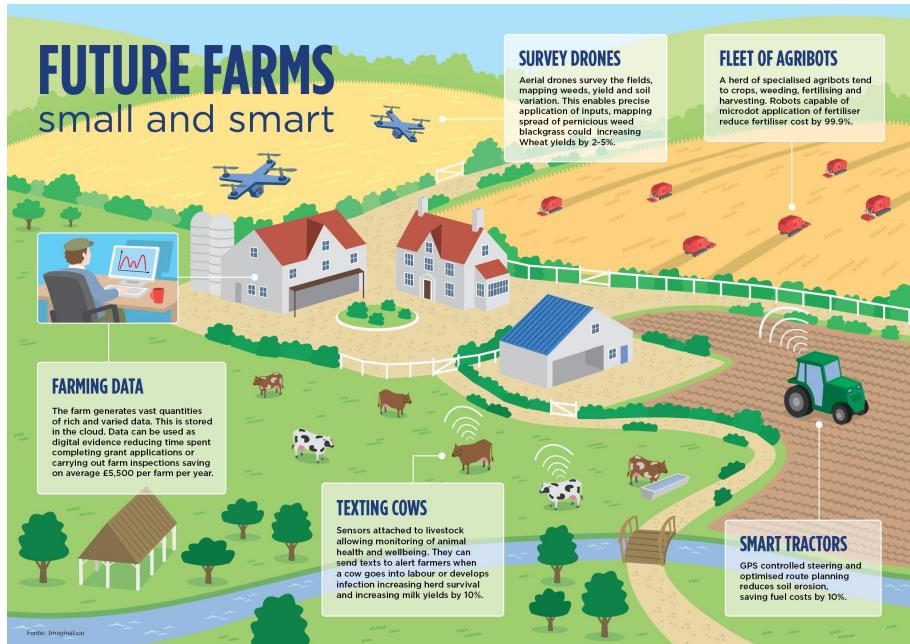
- Com a IoT, objetos e dispositivos estarão conectados todos os dias;
- E em todos os tipos de redes:
 - Intranets
 - Ponto-a-ponto
 - Internet global
- A IoT baseia-se no imenso sucesso das redes móveis e da internet;
- Com os efeitos da miniaturização e da nanotecnologia não é possível prever quais serão seus limites;
- Discussões sobre a padronização da IoT em:
 - Industrial Internet Consortium;
 - Industrial Internet Reference Architecture
 - OpenFog Consortium
 - Open Connectivity Foundation

IoT – Grandes Áreas

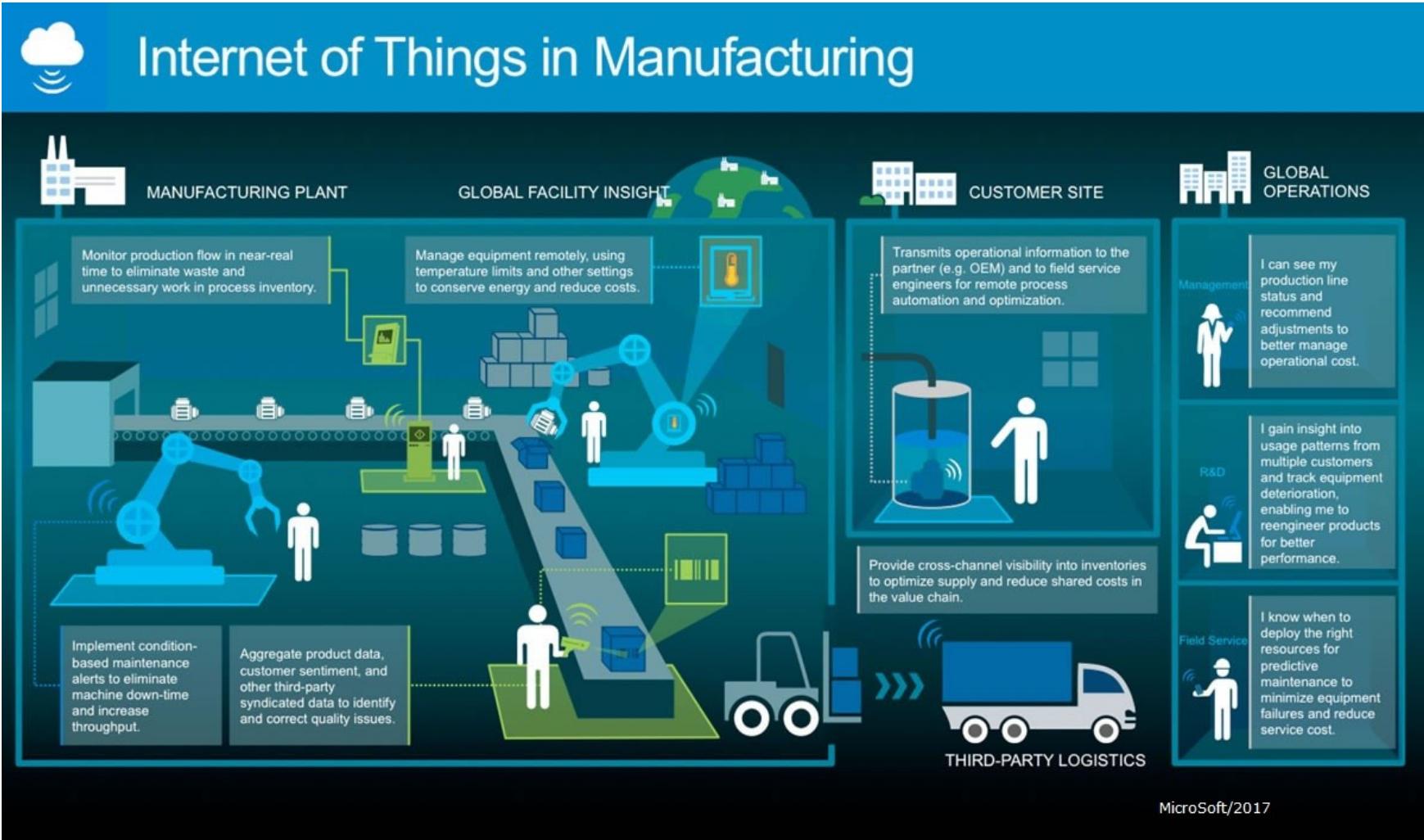
Libelium Smart World



IoT - Grandes Áreas



IoT – Grandes Áreas



Construindo a IoT



■ Construtores da IoT:

- Unidades de Controle;
 - É parte da inteligência da IoT.
- Sensores;
 - São dispositivos que medem uma quantidade física transformando-a em um sinal elétrico.
- Atuadores;
 - São dispositivos que transformam um sinal elétrico em um estímulo físico.
- Módulos de Comunicação;
 - São responsáveis pela comunicação, com ou sem fio, entre os dispositivos e com a internet.

ESP8266



- Produzido pela empresa chinesa Espressif Systems;
- Primeira versão em Agosto/2014;
- Possui plataforma de desenvolvimento própria;
- Roda a linguagem LUA;
- Pode ser utilizado pela IDE do Arduino;

ESP8266

Especificação



- Microcontrolador Tensilica L106 de 32 bits a 80 Mhz;
- RAM de instrução de 32 kB;
- RAM de dados do sistema ETS de 16 kB;
- Memória Flash 4Mb;
- Modelo Pro com memória flash de 16Mb;
- USB – porta micro USB para alimentação, programação e depuração;
- Até 17 pinos GPIO;
- 802.11 b/g/n, compatível com WPA/WPA2;

ESP8266

Especificação



- Suporte aos modos STA/AP;
 - Protocolo TCP/IP;
 - Protocolo TCP/UDP;
 - Compatível com pinos com Arduino UNO, Mega;
 - Corrente de operação WiFi: operação de transmissão contínua: \approx 70mA (200mA MAX), modo de hibernação profunda: <3mA;
 - Taxa de transmissão WiFi serial: 110-460800bps;
 - Temperatura: -40 °C ~ + 125 °C;

ESP8266

Especificação



- Umidade: 10%-90% sem condensação;
- Peso: cerca de 20g (0,7 onças);
- Modulação por largura de pulso (PWM);
- Capacidade de interrupção;
- Corrente máxima através dos pinos GPIO: 20mA;
- Firmware disponível para Arduino IDE;
- Pode operar em *light* e *deep sleep mode*(acorda por Tempo ou evento Externo)

ESP8266 e ESP32



- Há 3 tipos de interfaces com o ESP8266 e o ESP32;
 - Digital
 - Analógica
 - Protocolos seriais de comunicação:
 - TTL serial
 - I²C
 - I²S
 - 1-Wire
 - Serial Peripheral Interface(SPI)

ESP8266



■ Suporta:

- Redes Mesh
- Protocolo NOW
- Flash File System
- HTTP e HTTPS
- Email
- MQTT
- Wi-Fi (Como *station*, *access point* ou ambos)
- Web Server(contendo HTML, CSS e JavaScript)
- JSON
- OTA (over the air)

WeMos e NodeMCU



Baseado no ESP-8266-EX

11 digital I/O pins

4 PWMs

1 ADC – máximo 3.2V

1 UART

Voltagem de operação 3.3V

Voltagem de entrada recomendada 9-24V

Memória flash de até 4Mbytes

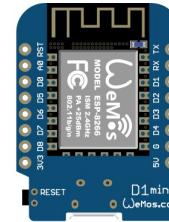
Processador de 80/160Mhz

Conexão micro USB

Suporte a Wi-Fi nativo, ideal para a IoT

OTA – Wireless Upload(program)

Driver CH340G



NodeMCU

Baseado no ESP-8266-12E

11 digital I/O pins

4 PWMs

1 ADC

1 UART

Voltagem de operação 3.3V

Memória flash de até 4Mbytes

Processador de 80/160Mhz

Conexão micro USB

Suporte a Wi-Fi nativo, ideal para a IoT

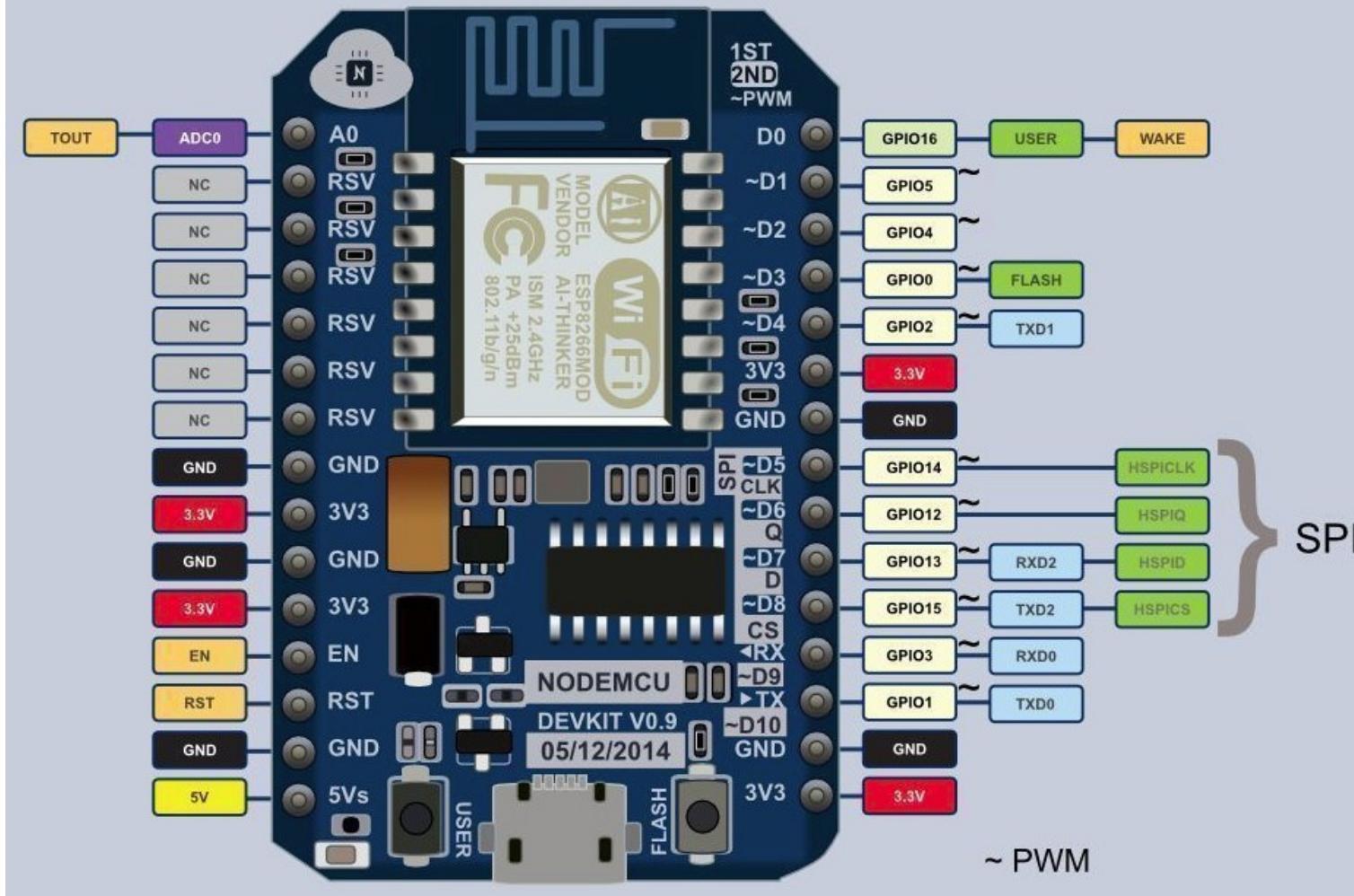
OTA – Wireless Upload(program)



NodeMCU



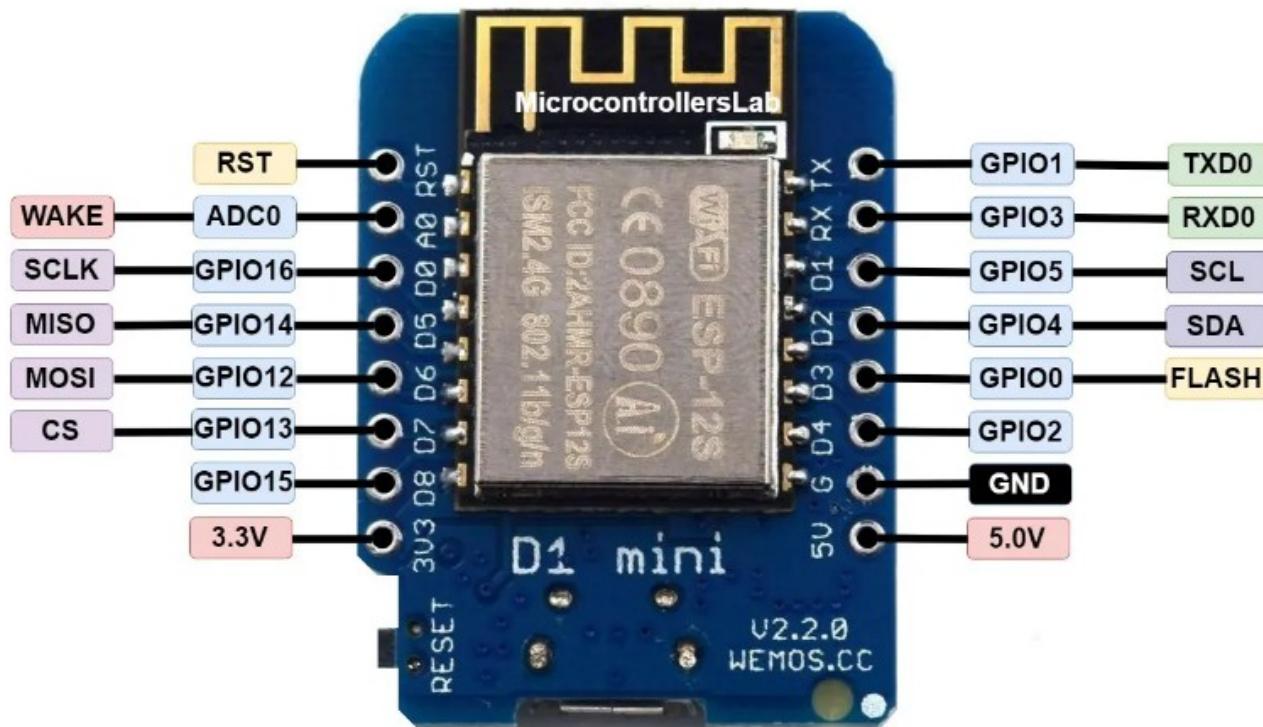
The NODEMCU Development kit



Wemos mini D1 pinagem



ESP8266 12E Wemos D1 Mini pinout



Wemos mini d1 pinagem



Pin		
Pin	Function	ESP-8266 Pin
TX	TXD	TXD
RX	RXD	RXD
A0	Analog input, max 3.3V input	A0
D0	IO	GPIO16
D1	IO, SCL	GPIO5
D2	IO, SDA	GPIO4
D3	IO, 10k Pull-up	GPIO0
D4	IO, 10k Pull-up, BUILTIN_LED	GPIO2
D5	IO, SCK	GPIO14
D6	IO, MISO	GPIO12
D7	IO, MOSI	GPIO13
D8	IO, 10k Pull-down, SS	GPIO15
G	Ground	GND
5V	5V	-
3V3	3.3V	3.3V
RST	Reset	RST

All of the IO pins have interrupt/pwm/I2C/one-wire support except D0

ESP32-WROOM-32D

Especificação

- MCU de baixos custo e consumo energético, integra Wi-Fi e BlueTooth *dual-mode*.
- Microprocessor Xtensa Dual-Core 32-bit LX6, operando em 80Mhz, **160Mhz** ou 240 Mhz;
- Ultra Low Power (ULP) *co-processor*;
- Memória *flash* com 4M, 8M ou 16M;
- Implementa os protocolos TCP/IP e TCP/UDP;
- Produzido pela empresa chinesa Espressif Systems;
- Primeira versão em Setembro/2016;

ESP32-WROOM-32D

Especificação

- Possui plataforma de desenvolvimento própria;
- Consumo de apenas $10\mu A$, quando em *deep sleep mode*;
- Opera em temperaturas na faixa de -40° a $85^{\circ}C$;
- Voltagem operacional de 2.2V a 3.6V;
- Corrente máxima de saída 40mA;
- Voltagem de entrada recomendada 9-24V;
- Conexão micro USB;
- *Over The Air – Wireless Upload Program*;

ESP32-WROOM-32D

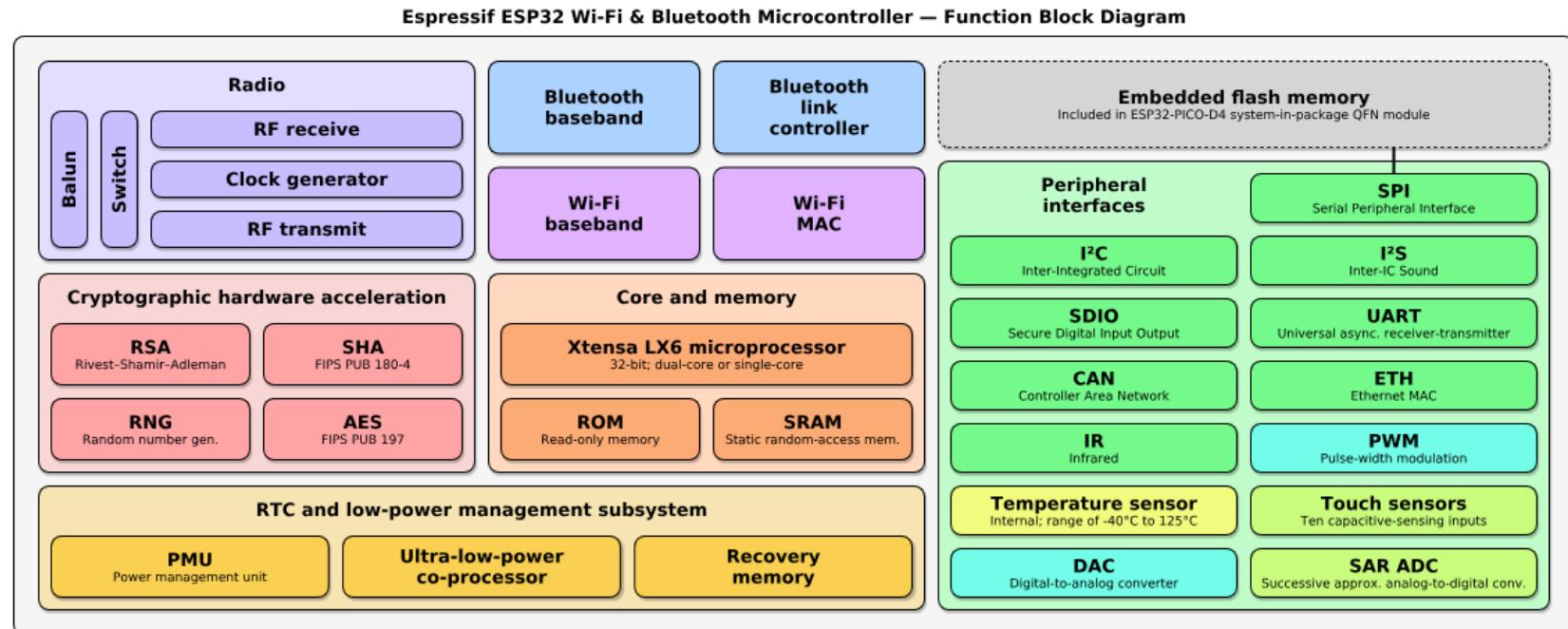
Especificação



- Pode operar em *light* e *deep sleep mode*(acorda por Tempo, Toque ou evento Externo)
- Bluetooth v4.2 BR/EDR and BLE;
- Bluetooth Class-1, class-2 e class-3;
- Wi-Fi 802.11 b/g/n;
- Wi-Fi 802.11 n (2.4 GHz), até 150 Mbps;
- Com suporte a Wi-Fi pode operar nos modos:
 - *Station*;
 - *Acess Point*;
 - *Acess Point + Station*;

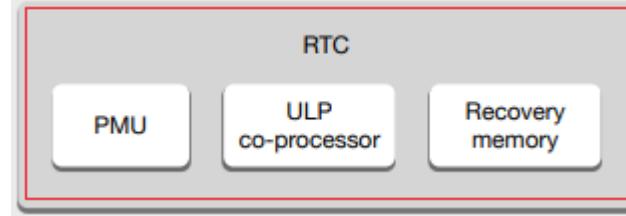
ESP32-WROOM-32D

Diagrama de Blocos



ESP32-WROOM-3

Módulo RTC



ULP (Ultra Low Power) co-processor:

Acionado quando os processadores principais estão em *deep sleep mode*;

Realiza medidas das entradas ADC, do sensor de temperatura e dos sensores em I²C;

Acessa a memória RTC_SLOW_MEM, os registradores em RTC_CNTL, RTC_IO e SAR ADC;

448Kb ROM para o *booting* e *core functions*;

520Kb SRAM para dados e instruções;

8Kb SRAM RTC SLOW Memory:

Pode ser acessada pelo *co-processor* quando em *deep-sleep mode*;

8Kb SRAM RTC FAST Memory:

Acessada pelas CPUs principais durante o RTC Boot, quando em *deep-sleep mode*;

1Kbit de EFUSE:

256 bits são usados para o sistema(MAC address e *chip configuration*);

768 bits são reservados para aplicações específicas, incluindo Flash-Encryption e Chip-ID;

ESP32-WROOM-32D

I/O Periféricos



■ Interfaces Periféricas de I/O

- 34 × GPIOs programáveis;
- 16 x 12-bits SAR(Successive Approximation Register) ADC;
- 2 x 8-bits DAC;
- 10 x *touch sensors*;
- 4 x SPI (Serial Peripheral Interface);
- 2 × I²S (Integrated Inter-IC Sound);
- 2 × I²C (Inter-Integrated Circuit);
- 3 × UART (Universal Asynchronous receiver/transmitter);
- 1 *host* (SD/eMMC/SDIO);

ESP32-WROOM-32D

I/O Periféricos



■ Interfaces Periféricas de I/O

- 1 *slave* (SDIO/SPI);
- Interface Ethernet MAC com DMA(Direct Memory Access) dedicado e suporte IEEE 1588;
- CAN 2.0;
- IR (TX/RX);
- Pulse Width Modulation(PWM) em todos os pinos;
- *Hall sensor*;
- *Ultra low power* pré-amplificador analógico;

ESP32-WROOM-32D

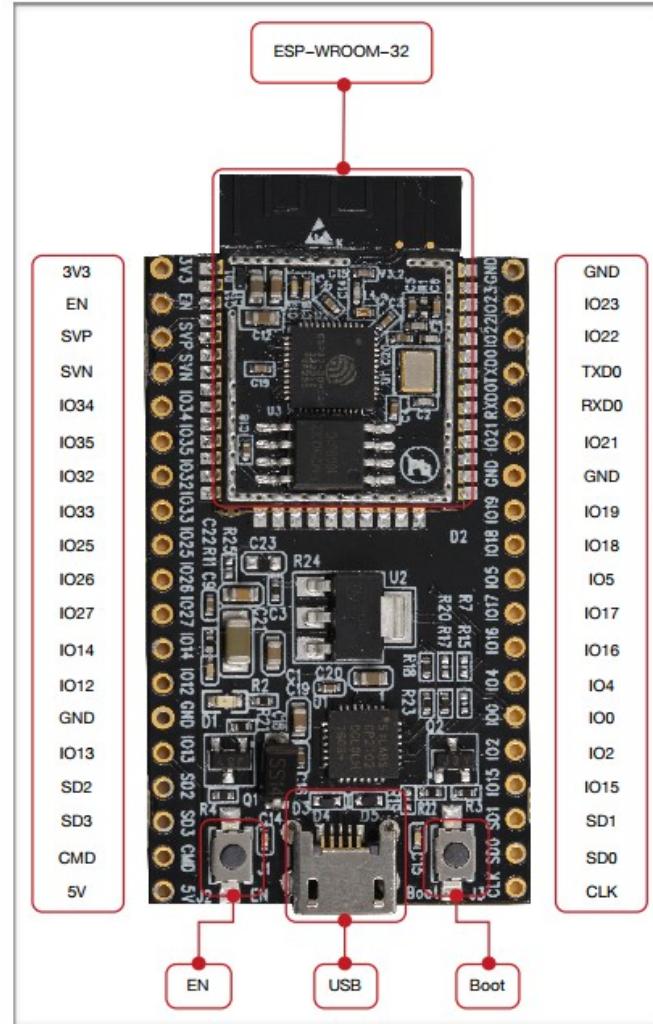
Segurança



■ Segurança

- Flash Encriptação;
- WEP, WPA/WPA2 PSK
- Cryptographic hardware acceleration:
 - AES
 - Hash(SHA-2)
 - RSA
 - ECC

ESP32-WROOM-32D/U



ESP32-WROOM-32D

Aplicações



■ Algumas Aplicações:

- *Hub* de sensores de IoT de baixa potência;
- Câmeras para *video streaming*;
- *Music players*;
- Internet *music players*;
- Dispositivos de *audio streaming*;
- Brinquedos controlados por wi-fi;
- Brinquedos sensíveis por proximidade;
- Dispositivos de reconhecimento de voz por wi-fi;
- Audio *headset*;
- Automação residencial;

ESP32-WROOM-32D

Aplicações

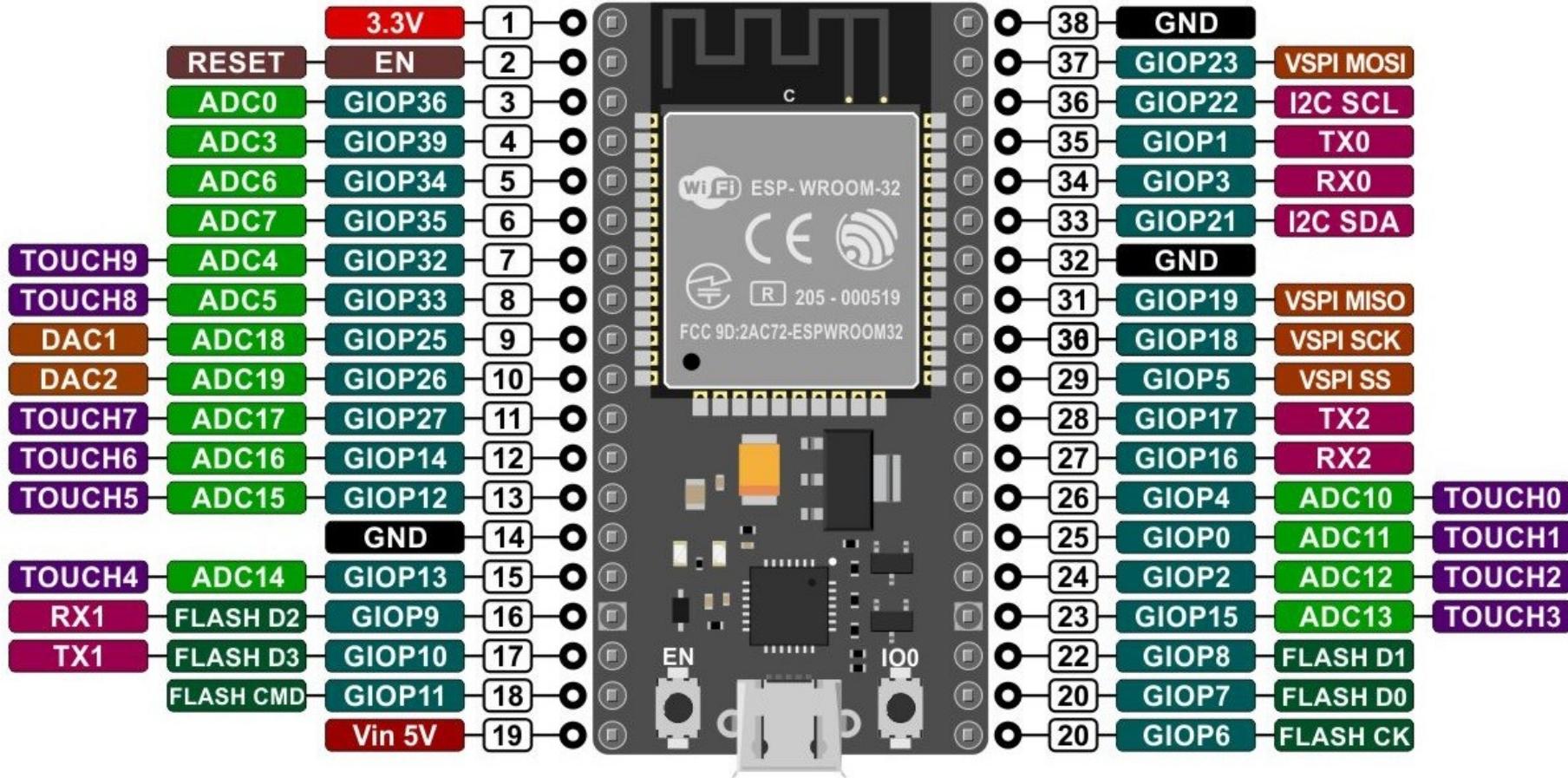


■ Algumas Aplicações:

- Redes *mesh*;
- Controle industrial por wi-fi;
- Monitoramento de bebês;
- Eletrônicos vestíveis;
- Dispositivos de rastreamento por wi-fi;
- Segurança por ID *tags*;
- Cuidados com a saúde;
- Dispositivos para monitoramento por proximidade e movimentação;
- Registros de temperaturas;

ESP-WROOM-32

PinOut

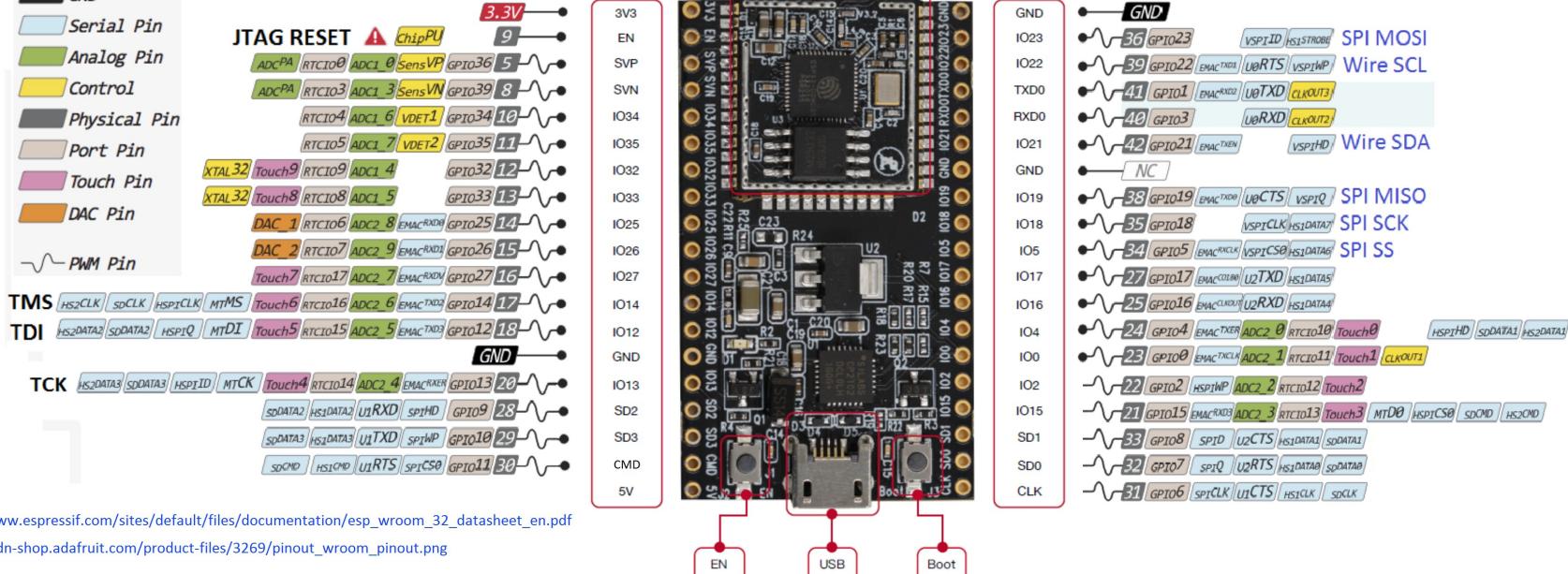


ESP-WROOM-32

PinOut



ESP32-WROOM-32 PINOUT



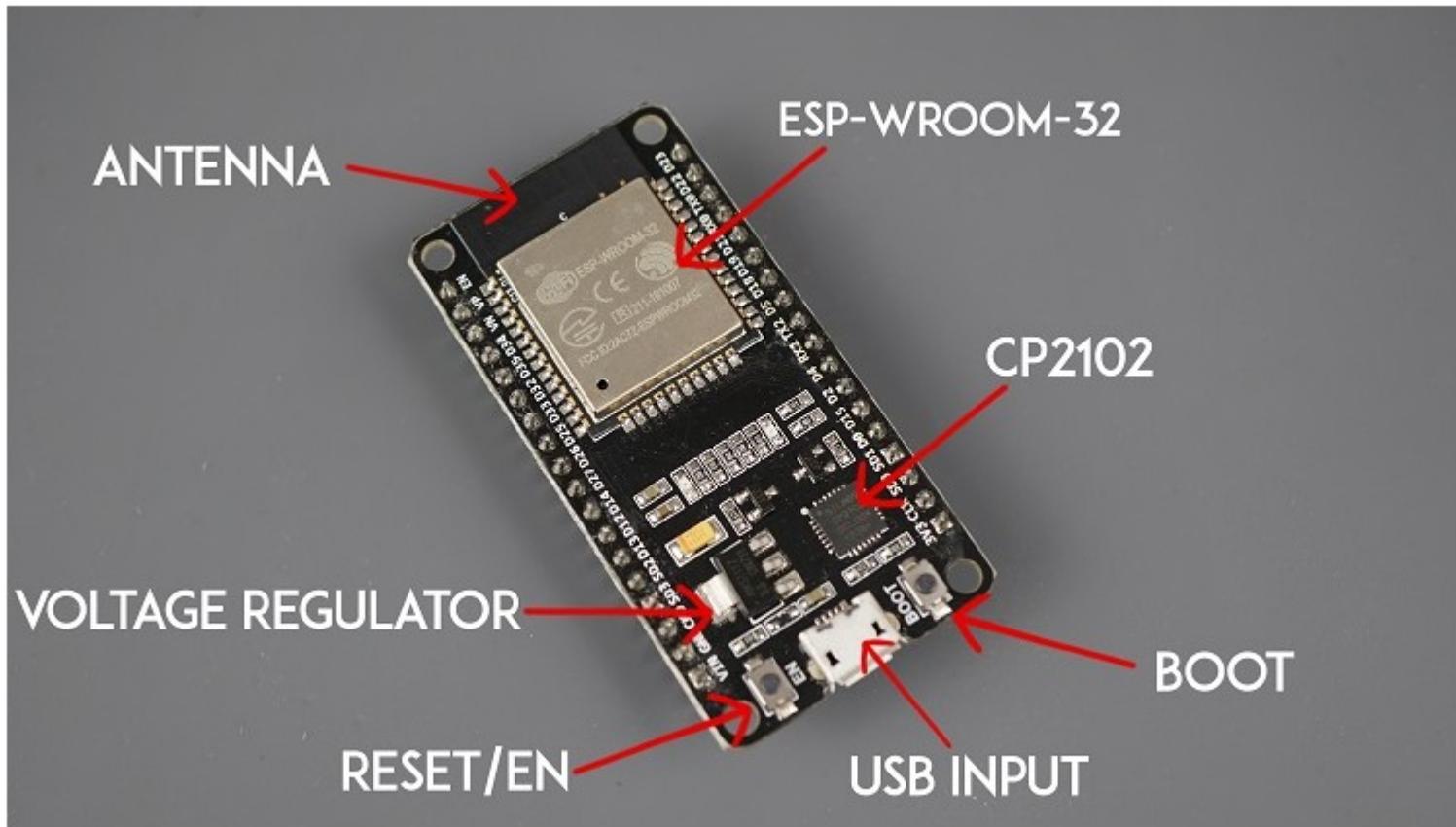
http://www.espressif.com/sites/default/files/documentation/esp_wroom_32_datasheet_en.pdf

https://cdn-shop.adafruit.com/product-files/3269/pinout_wroom_pinout.png

Important:

* Pins SCK/CLK, SD0/SD1, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on the module and are not recommended for other uses.

ESP-WROOM-32



ESP32 x ESP8266

Comparação



Comparação	ESP32	ESP8266
Clock Frequency	160 or 240mhz	80 mhz
Bluetooth	BLE	No
Hall Sensor	Yes	No
Camera Interface	No	No
Temperature Sensor	Yes	No
Touch Sensor	10	No
Security	Yes	No

ESP32 x ESP8266

Comparação



Comparação	ESP32	ESP8266
Low Power Consumption	10uA	20uA
Temperature Sensor	Yes	No
Co-Processor	ULP	No
Total GPIO	39	17
Crypto	RSA,RNG, CC,SHA-2,AES	No
SPI	4	2
Microcontroller	Dual-core 32	Single core 32-bit

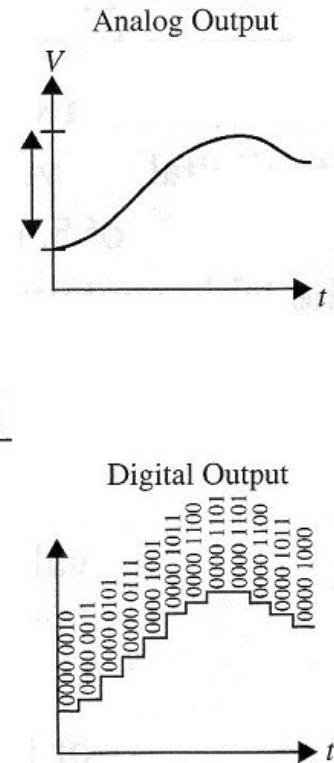
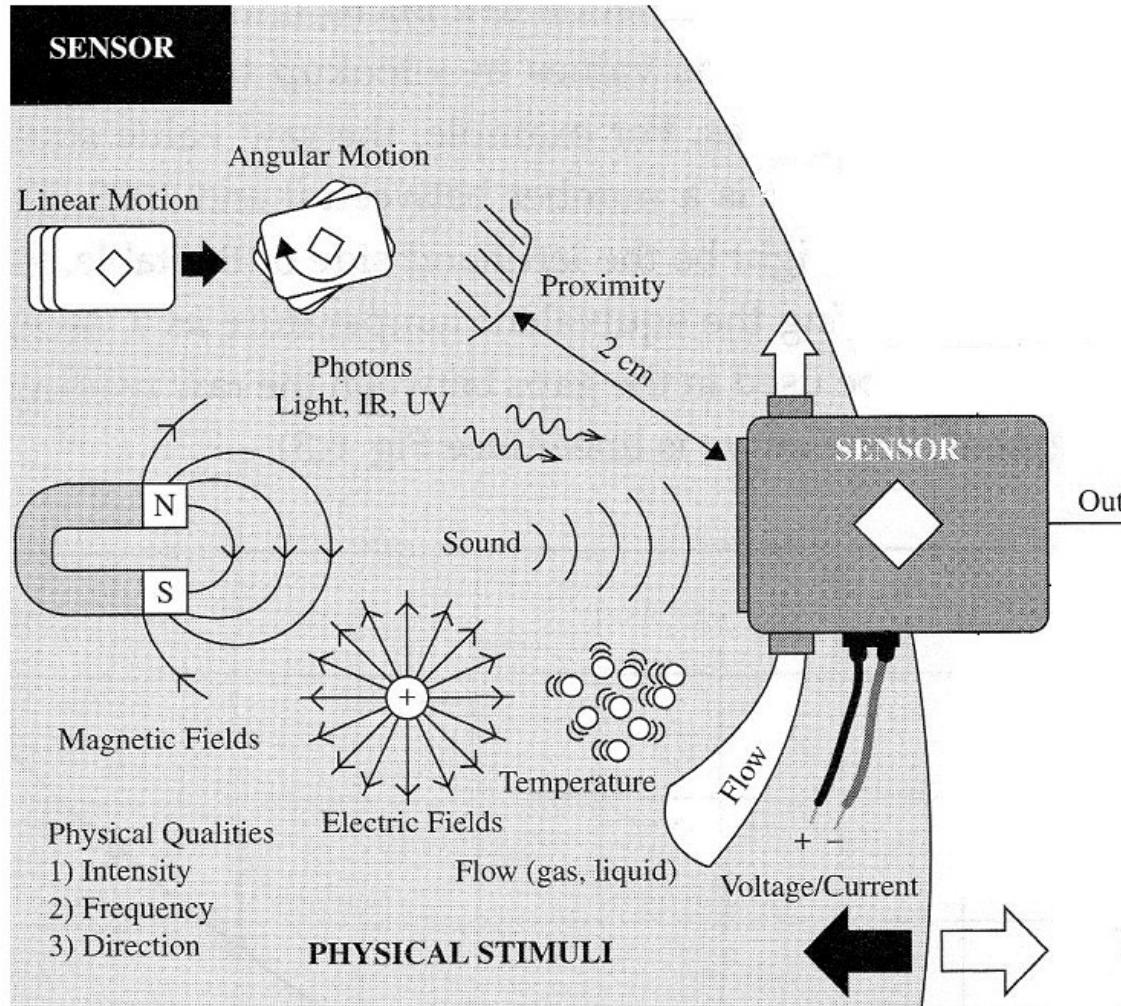
ESP32 x ESP8266

Comparação



Comparação	ESP32	ESP8266
ROM	448 KB	No
CAN	2	No
Ethernet	10/100 Mbps	No
External SPIRAM	Up to 16 MB	Up to 16 MB

Sensores



ELECTRICAL OUTPUT

ESP8266 e ESP32

IDE Arduino



■ Passos para serem utilizados pela IDE do Arduino:

- Abrir IDE do Arduino
- Clicar em Arquivo → Preferências
 - https://raw.githubusercontent.com/espressif/arduino-esp32/master/package_esp32_index.json,
http://arduino.esp8266.com/stable/package_esp8266com_index.json
 - Clicar em OK
- Clicar em Ferramentas → Placa → Gerenciador de Placas
 - Procurar a entrada **esp8266 by ESP8266 Community**
 - Clicar sobre esta entrada
 - Selecionar a versão mais recente e instalar

ESP8266 e ESP32

IDE Arduino



- Clicar em Ferramentas → Placa → Gerenciador de Placas
 - Procurar a entrada **ESP32**
 - Clicar sobre esta entrada
 - Selecionar a versão mais recente e instalar

Entradas e Saídas digitais



- Por padrão, todos os pinos do ESP8266 e do ESP32 são classificados como pino de INPUT;
- Se algum pino for utilizado como OUTPUT isto deve ser declarado no *sketch*;
- No ESP8266, os pinos **GPIO6 a GPIO11** não podem ser **utilizados**, são de uso reservado;
- No ESP32, os pinos **GPIO34 a GPIO39** são somente pinos de INPUT;
- As entradas e saídas dos pinos digitais podem acusar, ou propagar, um sinal alto(HIGH) ou baixo(LOW);

Entradas e Saídas digitais



- Os pinos de saída PWM(~) fazem um pouco mais;
- O ESP8266 e o ESP32 tem a capacidade de detectar um sinal elétrico de até 10µs de duração;
- A biblioteca padrão do Arduino disponibiliza funções para manipular estes pinos;

pinMode(), digitalWrite() e digitalRead()



Sintaxe:

`pinMode(pino,modo);`

- pino – identifica qual pino será utilizado;
- modo – indica a finalidade do pino: INPUT, OUTPUT ou INPUT_PULLUP;
- Em geral, definido na função `setup()`;

`digitalRead(pino);`

Retorna o valor HIGH(1) ou LOW(0);

`digitalWrite(pino,valor);`

valor – HIGH ou LOW;

Entradas e Saídas digitais



```
const byte ledPin = 2;          // Pino 2 é o BUILTIN_LED

void setup(){
    pinMode(ledPin, OUTPUT);
}

void loop(){
    digitalWrite(ledPin, HIGH);
    delay(1000);
    digitalWrite(ledPin, LOW);
    delay(1000);
}
```

delay() e delayMicroseconds()



■ O ESP8266 e o ESP32 interrompe qualquer atividade durante o tempo especificado na função;

■ Sintaxe:

- `delay(tempo-em-milissegundos);`
- `delayMicroseconds(tempo-em-microssegundos);`

■ Utilizado para sincronizar e temporizar tarefas;

- Exemplo: verificar a temperatura a cada 60 segundos.

■ No `delayMicroseconds()` o maior valor suportado, sem perda da acurácia, é 16383;

Exercício A1

Controlando LED - ESP8266

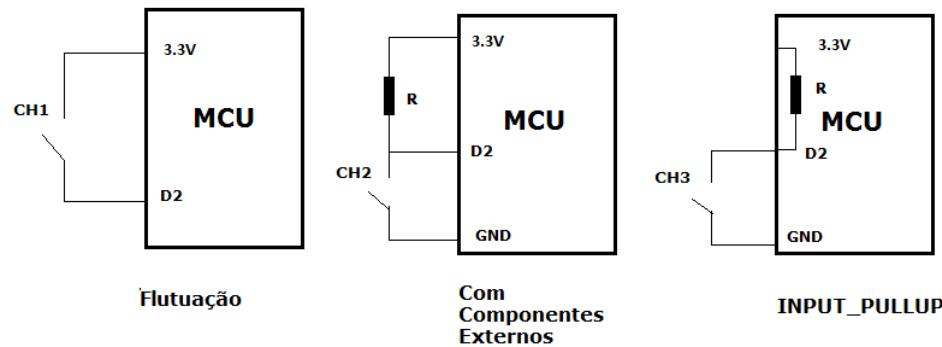


- Ligar/desligar o led localizado no pino 2 do ESP8266;
 - O led deve ficar aceso durante 2s e apagado durante 1s;
 - O ciclo se repete indefinidamente;
- Componentes utilizados;
 - Placa ESP8266;
 - LED padrão do ESP8266;

Modo INPUT_PULLUP



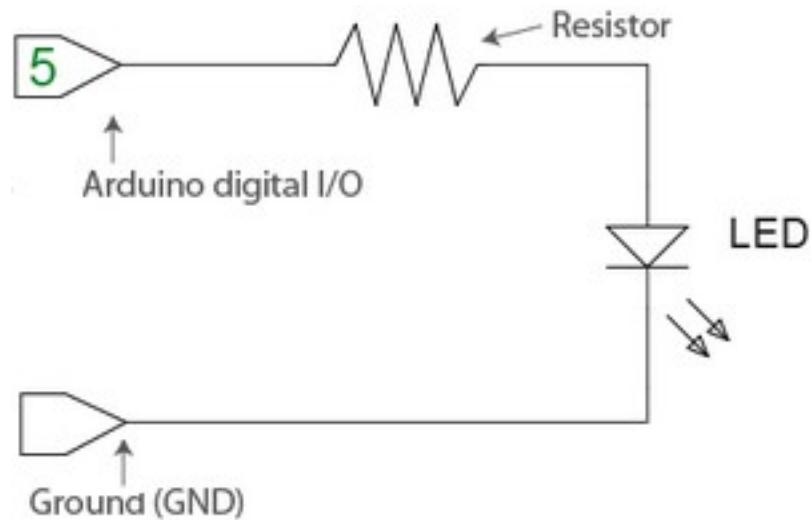
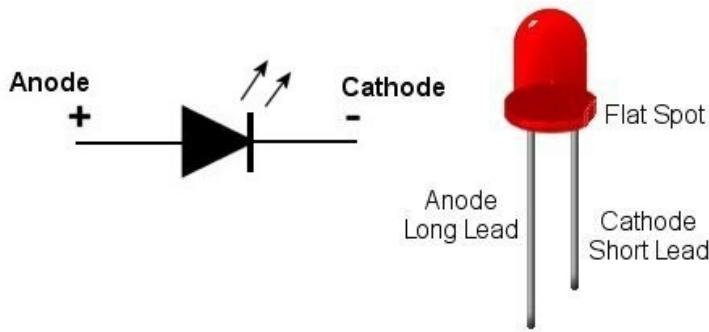
- Resistor embutido nos pinos digitais do ESP8266 e do ESP32;
- Usado para garantir o estado do sinal em 0(LOW) ou 1(HIGH);
- Quando o pinMode() indica INPUT_PULLUP o resistor é ligado a 3.3V da MCU;
- Com a chave(CH3) aberta, o sinal fica HIGH, fechada o sinal fica LOW;
- Todos os pinos GPIO (exceto os pinos GPIO34, GPIO35, GPIO36 e GPIO39) possuem resistor TNPII IT PI II IIP no FSP32.



Componente LED



- Quando conectado ao ESP8266 ou ESP32, por precaução, deve ser colocado em série com um resistor;



Exercício A2

Controlando LEDs - ESP8266



- Alternar o acendimento de 2 Leds;
 - Cada led deverá ficar aceso durante 1s, alternadamente;
 - O ciclo se repete indefinidamente;
- Componentes utilizados;
 - Placa ESP8266;
 - LEDs vermelho e verde;
 - Jumpers;

Entradas Analógicas ESP8266

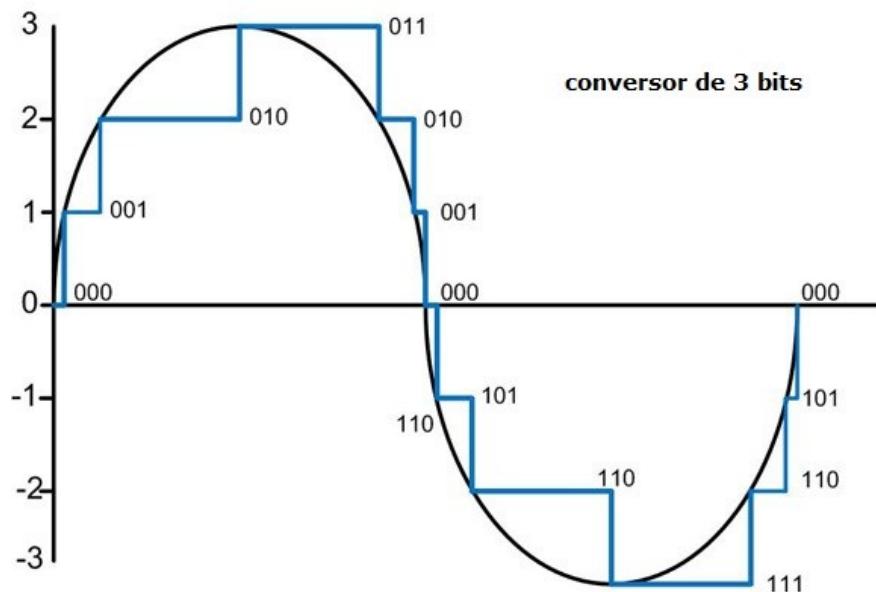


- Sinais analógicos são aqueles que não podem ser discretamente classificados;
- Podem ter um infinito número de valores, em um único intervalo;
- Um computador não pode medir um número infinito de casas decimais, possíveis em um sinal analógico;
- O Arduino converte o sinal analógico para uma representação digital(*analog-to-digital-converter*);
- O Esp8266 usa uma resolução de **10 bits**($2^{10} = 1024$) para esta operação;

Entradas Analógicas ESP8266



- O ESP8266 atribui um valor entre 0 e 1023 para qualquer valor analógico lido;
- Um valor de entrada de 3.3V(voltagem de referência) corresponderia a 1023, de 0V a 0, de 1.6V a 512;



Entradas Analógicas ESP32



- Usa uma resolução de **12 bits**($2^{12} = 4096$) para esta operação;
- O ESP32 atribui um valor entre 0 e 4095 para qualquer valor analógico lido;
- Um valor de entrada de 3.3V(voltagem de referência) corresponderia a 4095, de 0V a 0, de 1.6V a 2047;

Exercício A3

Valores Analógicos - ESP8266



■ Ler e apresentar os valores analógicos do A0.

- Repetir a leitura a cada 200ms;

■ Ligações:

ESP8266	Trimpot
3.3V	Terminal Superior
GND	Terminal Inferior
A0	Centro

■ Componentes utilizados:

- 1 ESP8266;
- 1 Trimpot 10K;
- Jumpers;

Obs.: int analogRead(A0)

PWM ESP8266



- PWM é o acrônimo de *Pulse Width Modulation*;
- Utilizado para produzir sinais que emulam saídas analógicas;
- Os pinos aptos à PWM são identificados pelo sinal ~;
- No ESP8266 todos os pinos GPIO são PWM;
- A saída PWM usa 8bits($2^8 = 256$) para esta operação;
- A saída, então, produzirá um sinal com largura de pulso entre 0 e 255;
- 255 corresponde a 3.3V, 122 a 1.6V, 0 a 0V;

PWM

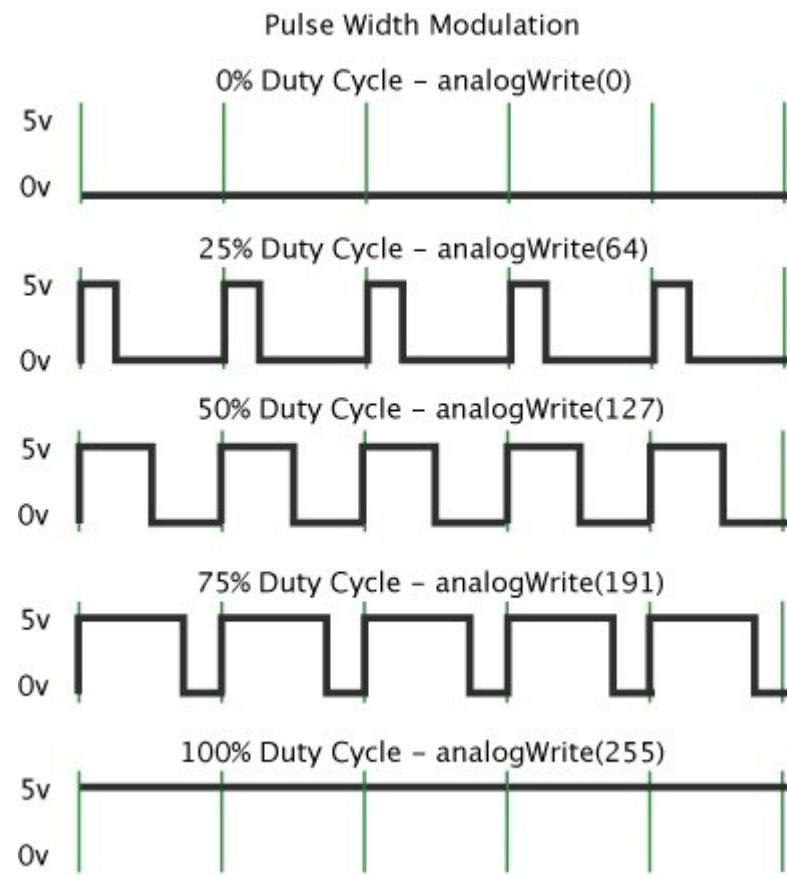


Figura extraída de <https://www.arduino.cc/en/Tutorial/PWM>

analogWrite() e analogRead()



■ Sintaxe:

```
analogWrite(pino,valor); // Somente para o ESP8266
```

pino – produz um sinal PWM neste pino;

valor – valor da largura de pulso, entre 0 e 255;

```
analogRead(pino);
```

retorna um valor entre 0 e 1023;

Obs.: O pino analógico deve ser declarado iniciando com a letra A;

map() e constrain()



- Função map();
- Utilizada para mapear faixas de valores em outras faixas;
- Sintaxe:

```
map  (nome-da-variavel,menor-valor-faixa-origem,maior-valor-faixa-  
      origem,menor-valor-faixa-alvo,maior-valor-faixa-alvo);
```

- Exemplo:

```
val = analogRead(A0);  
val = map (val,0,1023,0,255);  
analogWrite(6,val);
```

map() e constrain()



■ Função constrain();

■ Utilizada para restringir um número em uma faixa;

■ Sintaxe:

- constrain (nome-variavel,menor-valor-faixa,maior-valor-faixa);

■ Exemplo:

```
val = analogRead(A0);
val = constrain (val,10,500);
```

- Caso val<10, o valor de val será ajustado para 10;
- Caso val>500 o valor de val será ajustado para 500;
- Em seguida, deve ser usada a função map(), caso necessária;

Exercício A4

Controlar LED - ESP8266



- Controlar a luminosidade de um LED com um trimpot;
- Ligações:

ESP8266	Trimpot
3.3V	Limite Superior
GND	Limite Inferior
A0	Centro

- Componentes utilizados:

- 1 ESP8266;
- 1 Trimpot 10K;
- 1 LED;
- Jumpers;

PWM Controller ESP32



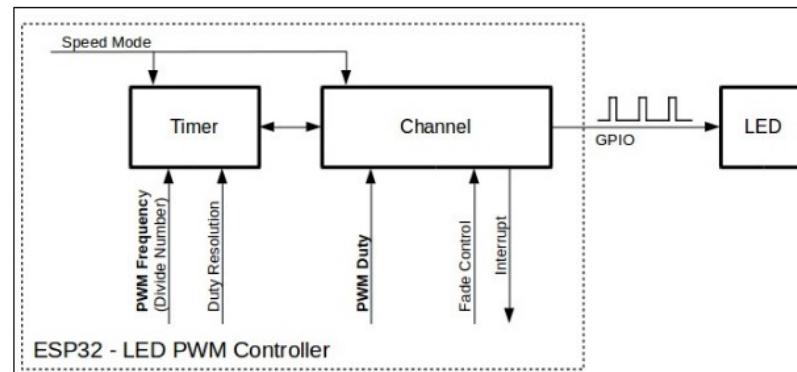
- Projetado, em princípio, para controlar luminosidade de LEDs;
- Pode, também, ser utilizado para gerar sinais para outros componentes;
- Implementado em todos os pinos do ESP32;
- Possui 16 canais que podem gerar formas de ondas independentes;
- Como o ESP32 não possui a função tone(), também é usado para tal;
- Não implementa a função analogWrite();

PWM Controller ESP32



Passos para usar:

- Escolher um canal. São 16 canais de 0 a 15.
- Definir a frequência do sinal PWM.
- Definir a resolução do ciclo de trabalho do sinal. Resoluções de 1 a 16 bits.
- Especificar em qual GPIO, ou GPIOs, o sinal aparecerá.



PWM Controller ESP32

Funções para uso



■ ledcSetup(canal,frequência,resolução) – Seleciona o canal, a frequência e a resolução do sinal PWM;

- Canal – valor entre 0 e 15;
- Frequência – valor entre 1 e 40MHz(para LEDs, um bom valor é 5KHz);
- Resolução – valor entre 1 e 16bits(para LEDs, resolução de 8bits nos atende);
- Retorna OK ou o tipo de erro;

■ ledcAttachPin(pino,canal) – associa um gpio a um canal;

- pino - qualquer gpio;
- canal – canal definido pela função ledcSetup();
- Retorna OK ou o tipo de erro;

■ ledcWrite(canal,duty-cycle) – dirige o sinal para o canal, segundo o duty-cycle;

- canal - canal definido na função ledcSetup();
- duty-cycle – duty-cycle em acordo com a resolução definida na função ledcSetup();
- Retorna OK ou o tipo de erro;

Obs.:

- O valor da resolução é inversamente proporcional ao valor da frequência. Para uma frequência de 5khz, o valor máximo da resolução é de 13bits(8192 níveis discretos de intensidade).
- Uma relação desproporcional levará ao erro “ledc: requested frequency and duty resolution can not be achieved, **try reducing freq_hz or duty_resolution**”;

■ Ou ao erro “ledc: requested frequency and duty resolution can not be achieved **try**

LED PWM Controller ESP32

Funções para uso



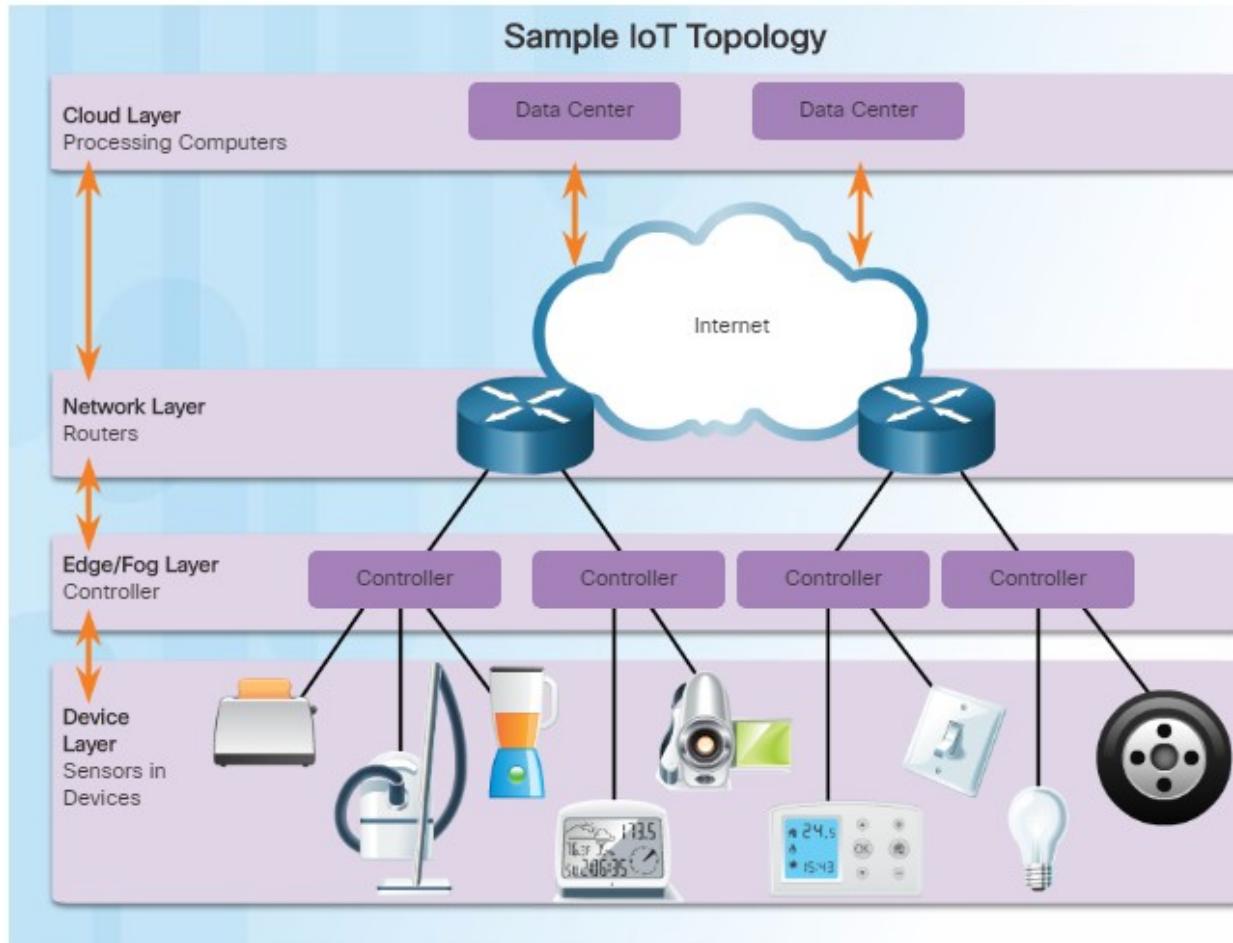
- `int ledcRead(canal)` – retorna o valor do duty-cycle associado ao canal;
 - `double ledcReadFreq(canal)` – retorna o valor da frequência associada ao canal;
 - `ledDetachPin(gpio)` – desassocia o pino do canal;
 - `ledcWriteNote(canal,nota-musical,oitava)` – seleciona o canal, a nota musical e a oitava;
 - Canal – valor entre 0 e 15;
 - Nota musical – valor da nota musical;
 - Oitava – oitava musical;
 - Retorna OK ou o tipo de erro;
 - `ledcWriteTone(canal,frequência)` – seleciona o canal e a frequência da nota musical;
 - Canal – valor entre 0 e 15;
 - Frequência – frequência da nota musical;
 - Retorna OK ou o tipo de erro;
- Obs.:
- Valores da notas musicais:
 - NOTE_C(dó), NOTE_Cs(dó sustenido), NOTE_D(ré), NOTE_Eb(mi bemol), NOTE_E, (mi) NOTE_F(fá), NOTE_Fs(fá sustenido), NOTE_G(sol), NOTE_Gs(sol sustenido), NOTE_A(lá), NOTE_Bb(si bemol), NOTE_B(si);

Computação na Névoa



- Tem como propósito reduzir a concentração de todo esforço computacional na nuvem;
- Dados podem ser transformados em informações ou ações já nesta camada;
- Ajuda a reduzir os custos de aplicações de IoT;
- Menor necessidade de banda;
- Melhora o tempo de resposta do Sistema;
- Minimiza a latência da internet;

Computação na Névoa



Fonte: Cisco System

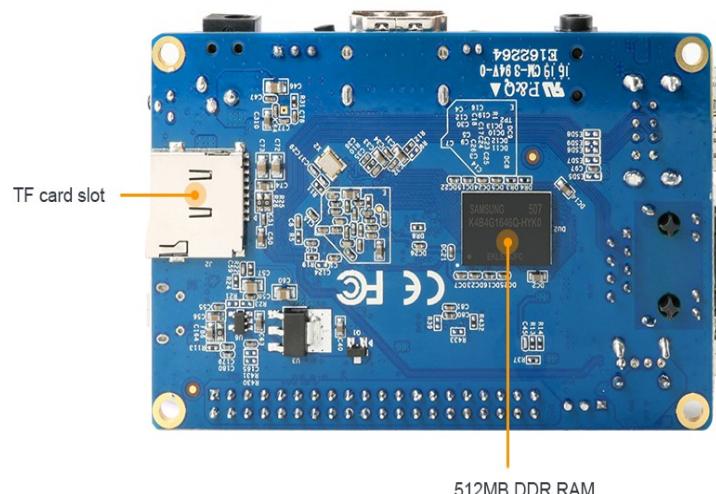
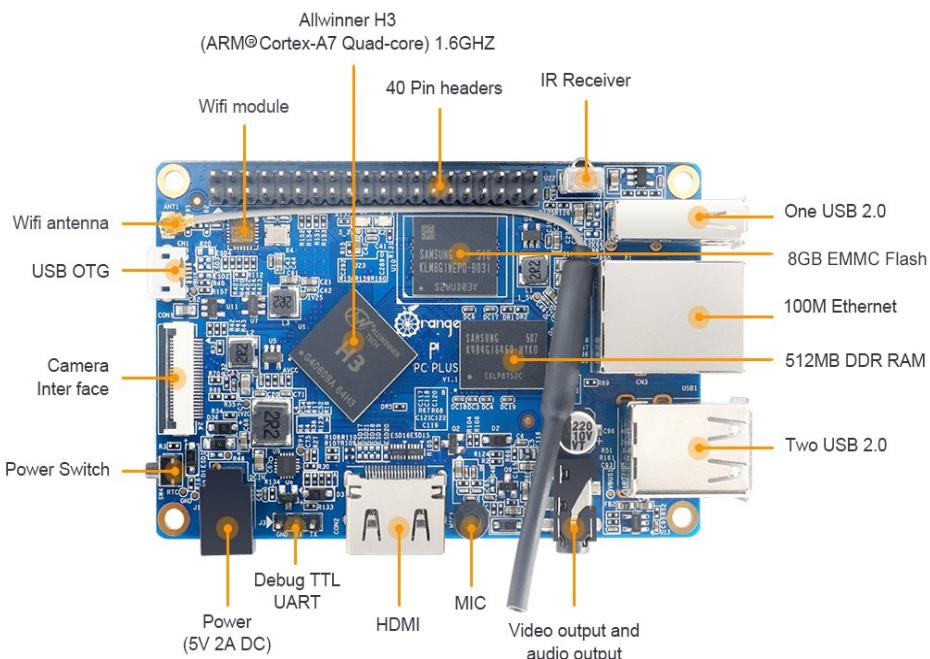
SBC - Orange Pi



Produzido pela empresa chinesa Shenzhen Xunlong

Em Março/2018 já oferecia cerca de 20 modelos de SBCs(Single Board Computers) diferentes.

Roda Linux, Android e Windows IoT



Orange Pi PC PLUS

Preço < US\$25.00

SBC - Raspberry Pi 3 B+



Raspberry Pi 3 Model B+ (lançado em Março/2018)

Processador 1.4GHz 64-bit quad-core ARM Cortex-A53 CPU

Memória RAM: 1GB

Dual-band 802.11ac wireless LAN and Bluetooth 4.2

Faster Ethernet (Gigabit Ethernet over USB 2.0)

Power-over-Ethernet support (with separate PoE HAT)

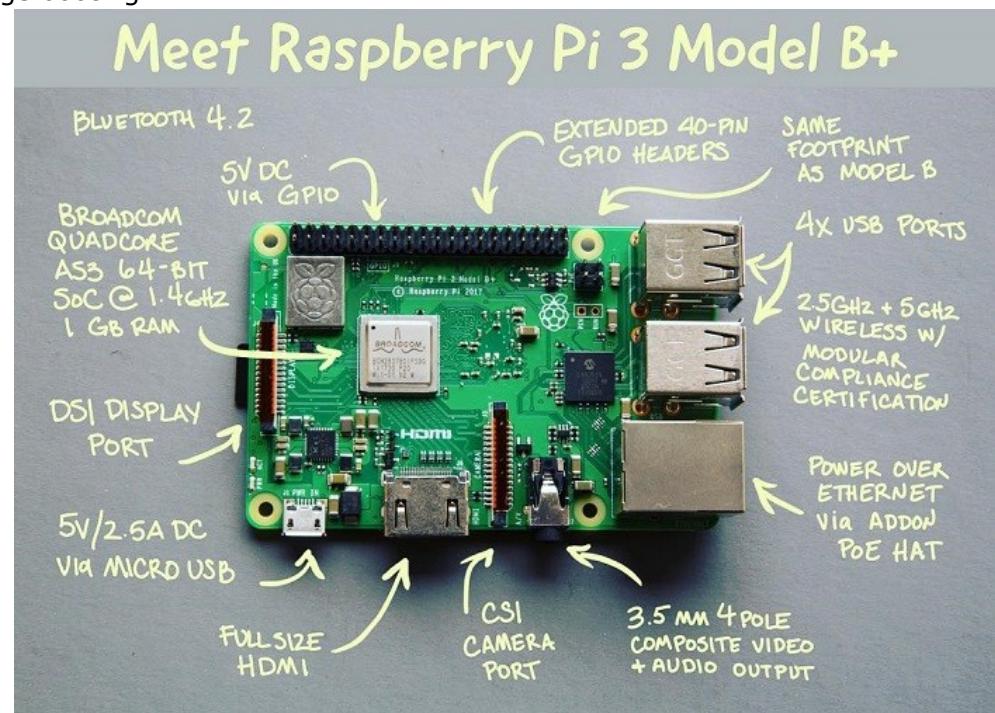
Improved PXE network and USB mass-storage booting

Improved thermal management

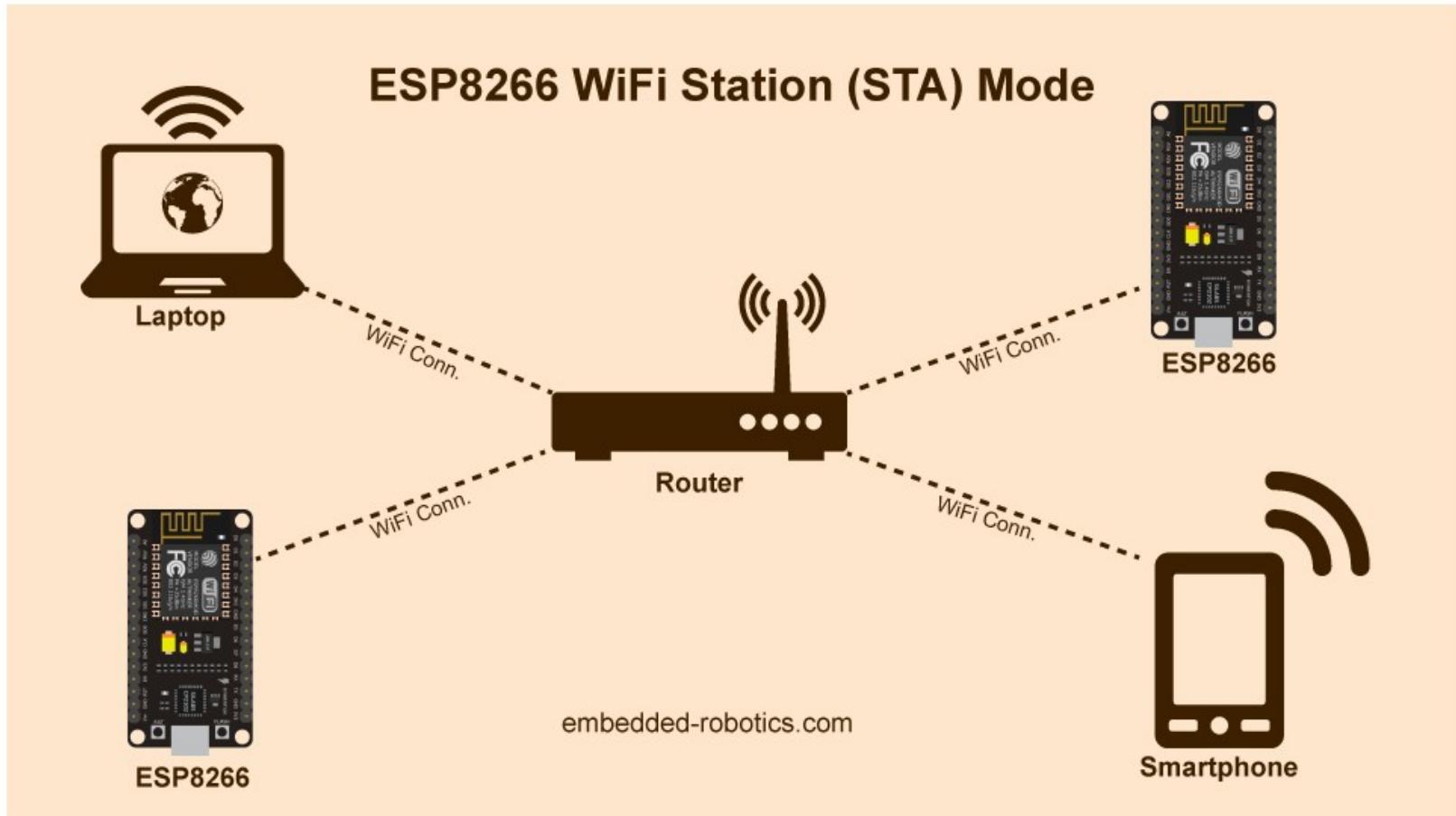
Roda Linux e Windows IoT

Dimensões: 85 x 56 x 17mm

Preço < US\$40.00

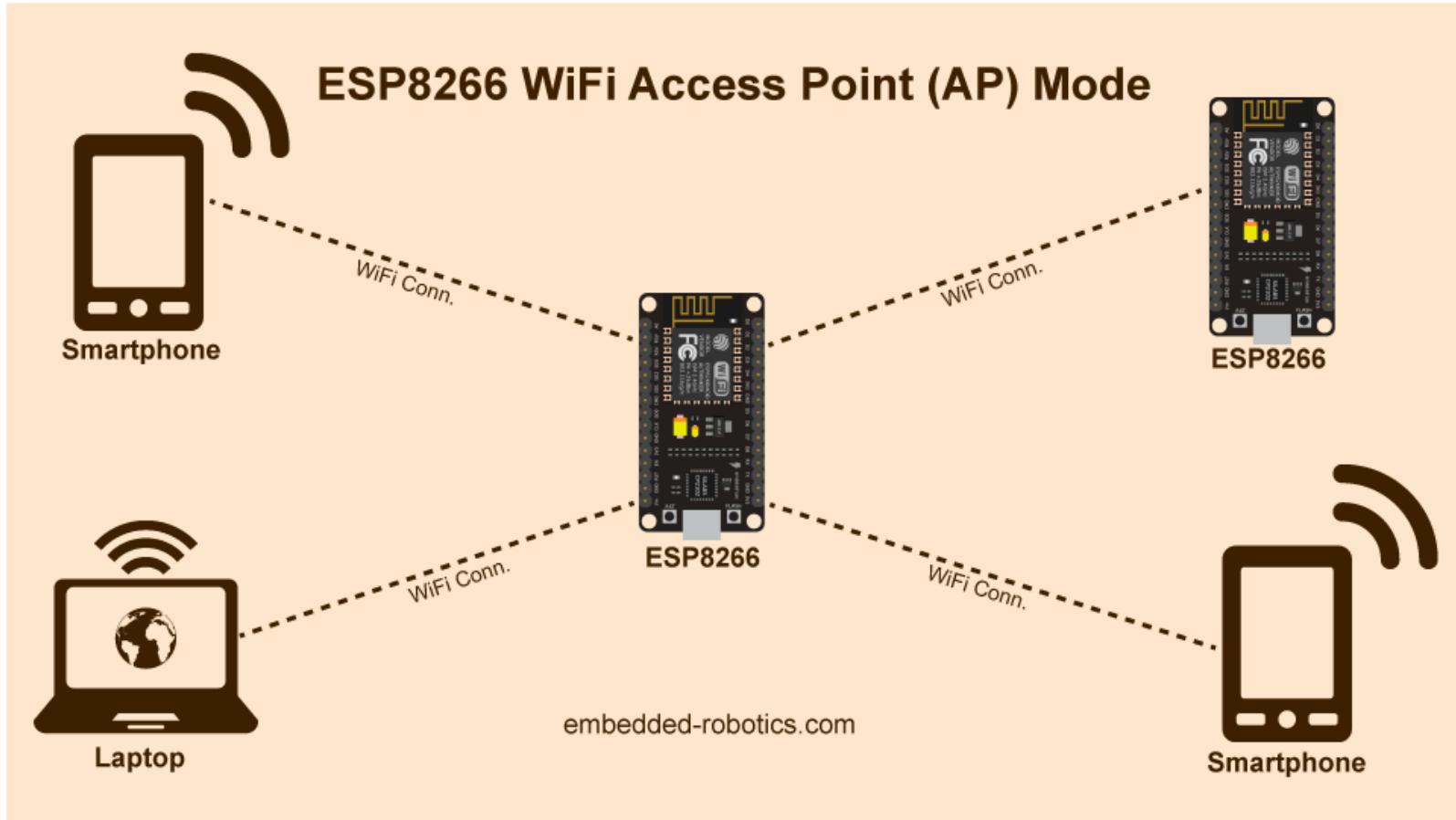


ESP8266 ou ESP32 WiFi station



ESP8266 WiFi Station (STA) Mode

ESP8266 ou ESP32 WiFi access point



ESP8266 WiFi Access Point (AP) Mode

ESP8266 e ESP32

Biblioteca Wifi



■ Algumas das funções oferecidas por sua biblioteca padrão;

- **Wifi.begin(ssid[,senha]);**
 - Para se conectar a uma rede wi-fi;
 - ssid – identifica o nome da rede wi-fi;
 - senha – senha para acesso a rede;
- **Wifi.status();**
 - Retorna o estado da conexão:
 - WL_CONNECTED → quando conectado
 - WL_NO_SSID_AVAIL → quando o ssid não está disponível
- **WiFiServer()**
 - Agirá como um servidor que ficará ouvindo, por determinada porta, a chegada de conexões;
- **server(porta);**
 - Ativa o serviço de servidor por determinada porta;
- **server.available();**
 - Verifica se chegou dados de algum cliente;
- **server.print(dados);**
 - Envia dados para o cliente. Dados podem ser tipos primitivos ou String:

ESP8266 e ESP32

Biblioteca Wifi



■ Algumas de suas funções;

- server.begin();
 - Inicia o serviço de escuta;
- server.write(dados);
 - Envia dados para o cliente. Dados podem ser do tipo char ou byte;
- **WiFiClient();**
 - Cria um cliente que se conectará a um específico endereço IP e porta;
- client.connect(ip,porta) ou client.connect(URL,porta);
 - Conecta o cliente a determinado servidor;
- client.connected();
 - Verifica se a conexão está ativa;
- client.write(dados);
 - Envia dados para o servidor. Dados podem ser do tipo char ou byte;
- client.print(dados);
 - Envia dados para o servidor. Dados podem ser tipos primitivos ou String;
- client.available();
 - Informa a quantidade de dados, retornada pelo servidor, disponíveis para leitura;

ESP8266 e ESP32

Biblioteca Wifi



■ Algumas de suas funções;

- client.read();
 - Lê o próximo byte enviado pelo servidor;
- client.flush();
 - Descarta qualquer byte que tenha sido enviado pelo servidor e que ainda não foi lido;
- client.stop();
 - Desconecta o cliente do servidor;
- **WiFi.mode(modo-de-operação);**
 - WIFI_OFF → desabilitará o wi-fi
 - WIFI_STA → o dispositivo agirá como uma estação wi-fi
 - WIFI_AP → o dispositivo agirá como um ponto de acesso
 - WIFI_AP_STA → o dispositivo agirá como uma estação wi-fi e como ponto de acesso

ESP8266

Conectando à rede WiFi



■ Código básico para a conexão como cliente:

– Área de definições:

```
#include <ESP8266WiFi.h>                                // Incluindo a Biblioteca
const char* ssid = "nome_da_rede_wiFi";                  // Nome da rede WiFi
const char* password = "senha-da-rede-wifi";             // Senha da rede WiFi
const char* host = "192.168.1.65";                        // Endereço do servidor, pode ser ipaddress
WiFiClient client;                                       // Agirá como cliente
int contaFalhas=0;
String dados ="?nome=Maria&cpf=1234&senha=12&confirmaSenha=1234";
```

– Área de set-up:

```
Serial.print("Conectando a ");
Serial.println(ssid);
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
    Serial.print(".");
    delay(500);
    contaFalhas++;
    if (contaFalhas>5){
        Serial.println("--> Mais de 5 falhas consecutivas!\n");
        contaFalhas=0:
```

ESP8266

Conectando à rede WiFi



■ Código básico para a conexão como cliente:

```
if (contaFalhas>0) {  
    contaFalhas--;  
}  
Serial.println('Conectado');  
}
```

Área do serviço de conexão:

```
if (client.connect(host,8080)) { // Tenta conectar-se ao servidor pela porta 8080  
    client.println("POST /CadastroPrimario/CadastroInicial" + dados); // Manda os dados  
    client.println("Cache-Control: no-cache");  
    client.println("Content-Type: application/x-www-form-urlencoded");  
    Serial.println("[Resposta:]");  
    while (client.connected()){  
        if (client.available()){  
            String line = client.readStringUntil('\n');  
            Serial.println(line);  
        }  
    }  
}  
  
delay(10000);
```

ESP8266 e ESP32

Biblioteca **HTTPClient**



- Biblioteca mais amigável para operações cliente servidor;
- Algumas das funções oferecidas por sua biblioteca padrão;
 - **HTTP.begin(client,host);**
 - Para se conectar como um cliente HTTP;
 - client – nome do objeto client
 - host – URL do destino;
 - **HTTP.addHeader(key,value);**
 - Adiciona dados ao cabeçalho HTTP:
 - key – nome da chave do cabeçalho;
 - value - conteúdo;
 - **int HTTP.POST(dados_para_enviar);**
 - **int HTTP.GET();**
 - Estas duas operações retornam o HTTP Response Code;

ESP8266 e ESP32 Biblioteca HTTPClient



- **int HTTP.GET();**
 - Estas duas operações retornam o HTTP Response Code;
- **HTTP.end();**
 - Chamado depois do tratamento do retornado;
- **String HTTP.getStream(); //Utilizar no ESP32**
 - Para capturar o conteúdo retornado;
 - Só funcionará em conjunto com a instrução:
 - `HTTP.useHTTP10(true); // Deve ser colocada antes do HTTP.begin()`
- **String HTTP.getString(); //Utilizar no ESP8266**
 - Para capturar o conteúdo retornado;

■ Para baixar a bliblioteca HttpClient:

- Baixar HttpClient-2.2.0 do Google Drive
- Clicar em Sketch → Incluir Biblioteca → Adicionar Biblioteca .ZIP
- Selecionar a biblioteca baixada

ESP8266

O Exemplo



```
#include <ESP8266HTTPClient.h>
WiFiClient client;
HTTPClient http;

void loop(){

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

    http.begin(client,host);
    http.addHeader("Content-Type", "application/x-www-form-urlencoded");
    String httpRequestData = "nome=Maria&cpf=1234&senha=12&confirmaSenha=1234";

    int httpResponseCode = http.POST(httpRequestData);

    Serial.print("Response Code: ");
    Serial.println(httpResponseCode);
    Serial.print(http.getString());

    http.end();
}

delay(10000);
```

Exercício B1

Executar com o ESP32



- Alterar o Exemplo_ESP8266_WiFi_Web_Client_02, para ser executado pelo ESP32;

ArduinoJSON



- Biblioteca para promover serialização e desserialização de objetos JSON;
 - Exemplos e aplicações em <https://arduinojson.org/>;
 - Suporta placas Arduinos diversas, ESP8266 e ESP32;
 - Passos para serem utilizados pela IDE do Arduino:
 - Abrir IDE do Arduino
 - Clicar em Sketch → Incluir Biblioteca → Gerenciar Bibliotecas
 - Procurar a entrada ArduinoJSON → Selecionar ArduinoJSON by Benoit Blanchon
 - Clicar sobre esta entrada
 - Selecionar a versão mais recente e instalar
- ou [baixar do Google Drive](#) e copiar para o diretório libraries do Arduino, vamos aprender



ArduinoJSON

Algumas Classes

JsonDocument

- Armazena um documento JSON na memória.

StaticJsonDocument

- É JsonDocument que usa uma quantidade pré-determinada de memória para uso(pilha), portanto, não depende da alocação dinâmica de memória.
- Como não chama malloc() e free(), StaticJsonDocument é um pouco mais rápido que DynamicJsonDocument.
- Recomendado para documentos pequenos, de até 1KB(nosso negócio!).
- Mude para DynamicJsonDocument se o documento for maior.

ArduinoJSON

Algumas Classes



- Sintaxe:
 - StaticJsonDocument<bytes-alocados-na-pilha> variavel;

■ DynamicJsonDocument

- É um JsonDocument que aloca a memória no heap.
- Sintaxe:
 - DynamicJsonDocument variavel(bytes-alocados);

ArduinoJSON função deserializeJson()



■ deserializeJson

- Função que analisa uma entrada JSON e coloca o resultado em um JsonDocument.
- Retorna um DeserializationError.
- Sintaxe:
 - `deserializeJson(JsonDocument,entrada_JSON);`

Executar Exemplo_ESP8266_WiFi_Web_Client_JSON.ino

Neste código foram utilizadas, como ilustrações, as funções `capacity()` e `memoryUsage()`.



ArduinoJSON ESP32 função deserializeJson()

■ No ESP32 utilizar as instruções:

- `HTTP.useHTTP10(true);` // Colocar antes do `HTTP.begin()`.
- `HTTP.getStream();` // Em vez de `getString()`

Executar Exemplo_ESP32_WiFi_Web_Client_JSON.ino

Neste código também foram utilizadas, como ilustrações, as funções capacity() e memoryUsage().



ArduinoJSON

função serializeJson()

serializeJson

- Serializa um JsonDocument para criar um documento sem espaços ou quebra de linha entre os valores
- Sintaxe:
 - `serializeJson(JsonDocument,variavel);`
- Exemplo:

```
String saida;
StaticJsonDocument<200> doc;
doc["erro"] = true;
doc["msg"] = "Tudo Certo!";
JsonArray data = doc.createNestedArray("temperaturas");
data.add(48.756080);
data.add(2.302038);
serializeJson(doc, saida);
```

Exercício B2

Executar com o ESP8266



- Consultar um *end-point* que passará ações sobre o estado de 2 leds. Este *end-point* retornará um objeto JSON com o seguinte conteúdo:
 - {'led01':estado,'led02':estado}
 - O estado pode ser '1'(para ligar) ou '0'(para desligar)
- Caminho → /CadastroPrimario/EstadosLeds
- O *end-point* gerará estados aleatórios;
- Os leds deverão reagir conforme os seus estados;
- Utilizar a biblioteca HTTPClient;
- Utilizar o método GET;

ESP8266 e ESP32 Biblioteca WebServer



- Permite ao ESP8266/ESP32 agirem como um servidor Web;
- Funções:
 - `ESP8266WebServer(porta)` ou `WebServer server(porta);`
 - porta - indica a porta ouvidora
 - `server.on(caminho,[método,]função)`
 - caminho - indica o caminho da requisição
 - método – `HTTP_GET` ou `HTTP_POST`
 - função - função que tratará a requisição
 - `server.onNotFound(função)`
 - função - indica qual função tratará um caminho inexistente(error 404!)

ESP8266 e ESP32 Biblioteca WebServer



Funções:

- server.**hasArg**('argumento')
 - Verifica se na requisição tem determinado argumento.
- server.**arg**('argumento')
 - Obtém o valor do argumento.

ESP8266 e ESP32 Biblioteca WebServer



■ Funções:

- server.**send**(código, tipo-retorno, mensagem);
 - código - HTTP response code
 - tipo retorno – content type do HTTP
 - mensagem – texto da resposta
- server.**handleClient()**
 - Para ficar ouvindo o meio

■ Para baixar a biblioteca:

- Baixar do Google Drive
- Clicar em Sketch → Incluir Biblioteca → Adicionar Biblioteca .ZIP
- Selecionar a biblioteca baixada

ESP8266 e ESP32

Biblioteca WebServer



Exemplo:

```
server.on("/",caminhoRaiz);
server.on("/led1on",caminhoLed1on);
server.on("/led1off",caminhoLed1off);
server.on("/led2on",caminhoLed2on);
server.on("/led2off",caminhoLed2off);
server.onNotFound(caminho404);

server.send(200,"text/html",SendHTML(LED1status,LED2status));

server.send(404,"text/plain","Erro - Caminho Inexistente!!!");
```

Coloquem leds nos pinos 12 e 14(D5 e D6).

Executar Exemplo_ESP8266_WiFi_Web_Server_01.ino.

Digitem, no navegador, o endereço IP obtido pelo ESP8266.

Digitem, no celular(tem que estar na rede INOVUERJ_LACIPI!), o endereço IP obtido pelo ESP8266.

ESP8266 e ESP32

Trecho do Exemplo



```
String ptr = "<!DOCTYPE html> <html>\n";
ptr += "<head><meta name=\"viewport\" content=\"width=device-width, initial-scale=1.0, user-scalable=no\">\n";
ptr += "<title>LED Control</title>\n";
ptr += "<style>html { font-family: Helvetica; display: inline-block; margin: 0px auto; text-align: center; }\n";
ptr += "body{margin-top: 50px;} h1 {color: #444444; margin: 50px auto 30px;} h3 {color: #444444; margin-bottom: 50px;}\n";
ptr += ".button {display: block; width: 80px; background-color: #1abc9c; border: none; color: white; padding: 13px 30px; text-decoration: none; font-size: 25px; margin: 0px auto 35px; cursor: pointer; border-radius: 4px; }\n";
ptr += ".button-on {background-color: #1abc9c; }\n";
ptr += ".button-on:active {background-color: #16a085; }\n";
ptr += ".button-off {background-color: #34495e; }\n";
ptr += ".button-off:active {background-color: #2c3e50; }\n";
ptr += "p {font-size: 14px; color: #888; margin-bottom: 10px; }\n";
ptr += "</style>\n";
ptr += "</head>\n";
ptr += "<body>\n";
```

ESP8266 e ESP32

Trecho do Exemplo



```
ptr +=<h1>ESP8266 como Web Server - Station Mode</h1>\n;\n\nif(led1stat)\n{ptr +=<p>LED1 Status: ON</p><a class=\"button button-off\" href=\"/led1off\">OFF</a>\n;}\nelse\n{ptr +=<p>LED1 Status: OFF</p><a class=\"button button-on\" href=\"/led1on\">ON</a>\n;}\n\nif(led2stat)\n{ptr +=<p>LED2 Status: ON</p><a class=\"button button-off\" href=\"/led2off\">OFF</a>\n;}\nelse\n{ptr +=<p>LED2 Status: OFF</p><a class=\"button button-on\" href=\"/led2on\">ON</a>\n;}
```

É desagradável incluir códigos HTML, CSS e JavaScript aqui.

ESP8266 e ESP32 Biblioteca WebServer



■ Exemplo com POST:

```
server.on("/", HTTP_GET,caminhoRaiz);
server.on("/login", HTTP_POST,caminhoLogin);

void caminhoLogin() {
    if( ! server.hasArg("usuario") || ! server.hasArg("senha") || server.arg("usuario") == NULL || server.arg("senha") == NULL) {
        server.send(400, "text/plain", "400: Usuario ou Senha nao especificado");
        return;
    }
    if(server.arg("usuario") == "InovUERJ" && server.arg("senha") == "123456") {
        server.send(200, "text/html", "<h1>Agora vai, " + server.arg("usuario") + "!</h1><p>Acesso Autorizado</p>");
    } else {
        server.send(401, "text/plain", "401: Login ou Senha Invalidos");
    }
}
```

Executar Exemplo_ESP8266_WiFi_Web_Server_02.ino.

SPIFFS File-System



- É um sistema de arquivos voltado para dispositivos do tipo SPI NOR *flash*;
- Não suporta diretórios, por enquanto;
- Uma operação de *write* pode ter tempos distintos;
- Projetado para dispositivos com pouca memória;
- Não recomendado para memórias superiores a 128Mbytes;
- Sem capacidade de detectar *bad blocks*;
- Promovido pela biblioteca LittleFS(sucessora da SPIFFS), que herda da biblioteca FS.h;

SPIFFS File-System

Funções para o uso



■ Iniciar o sistema de arquivos SPIFFS:

- boolean LittleFS.begin()
 - Retorna true ou false

■ Para abrir um arquivo:

- File variavel = LittleFS.open(nome-do-arquivo, operação)
 - Nome-do-arquivo - nome do arquivo a ser aberto
 - Operação – “w+” para escrever; “r” para ler; “a” para acrescentar;

■ Verificar se o arquivo já existe:

- boolean File-variavel.exists()
 - Retorna true ou false

■ Para fechar um arquivo:

- File-variavel.close()

■ Para remover um arquivo:

- LittleFS.remove(nome-do-arquivo)

SPIFFS File-System

Funções para o uso



Para verificar se tem mais dados para serem lidos:

- File-variavel.available()

Para ler o registro:

- File-variavel.read() ou File-variavel.readString() ou File-variavel.readStringUntil(limite)

 Para gravar o registro:

- File-variavel.println(conteúdo-a-escrever)

Para ver o tamanho do arquivo:

- File-variavel.size();

Para formatar a memória:

- ### - LittleFS.format()

SPIFFS ESP8266

LittleFS Baixar



Passos necessários:

- Download, do Google Drive, da biblioteca ESP8266LittleFS-X.zip;
- Clicar em Arquivo → Preferências e checar o Local do SketchBook;
- Ir para o local do sketchbook e criar o diretório de nome **tools**;
- Unzipar o ESP8266LittleFS-X.zip e copiar a pasta ESP8266LittleFS para o diretório tools;
- O caminho final será /tools/ESP8266LittleFS/tool/esp8266littlefs.jar;
- Reiniciar o Arduino;
- Clicar em Ferramentas, deverá aparecer a opção “ESP8266 LittleFS Data Upload”;

SPIFFS ESP32 ESP32FS Baixar



Passos necessários:

- Download, do Google Drive, da biblioteca ESP32FS-1.0.zip;
- Unzipar o ESP32FS-1.0.zip e copiar a pasta ESP32FS para o diretório tools;
- O caminho final será /tools/ESP32FS/tool/esp32fs.jar;
- Reiniciar o Arduino;
- Clicar em Ferramentas, deverá aparecer a opção “ESP32 Sketch Data Upload”;

SPIFFS ESP8266

Upload de arquivos



■ Facilidade que possibilita a carga de arquivos, do computador, diretamente para o *file-system* do ESP8266;

■ Passos seguintes:

- Criar, e salvar, o *sketch* que tratará o arquivo;
- Criar, no diretório onde reside o *sketch*, um subdiretório de nome **data**;
- Copiar, para este subdiretório **data**, os arquivos objetos do *upload*;
- Na IDE do Arduino, abrir o *sketch* que tratará os dados;
- Em seguida, teclar a aba *tools*(ou ferramentas) que aparecerá a entrada “**ESP8266 LittleFS Data Upload**”;
- Clicar sobre esta entrada. A operação de *upload* será iniciada, copiando os arquivos para o ESP8266;

SPIFFS ESP32

Upload de arquivos



■ Facilidade que possibilita a carga de arquivos, do computador, diretamente para o *file-system* do ESP32;

■ Passos seguintes:

- Criar, e salvar, o *sketch* que tratará o arquivo;
- Criar, no diretório onde reside o *sketch*, um subdiretório de nome **data**;
- Copiar, para este subdiretório **data**, os arquivos objetos do *upload*;
- Na IDE do Arduino, abrir o *sketch* que tratará os dados;
- Em seguida, teclar a aba *tools*(ou ferramentas) que aparecerá a entrada “**ESP32 Sketch Data Upload**”;
- Clicar sobre esta entrada. A operação de *upload* será iniciada, copiando os arquivos para o ESP32;

SPIFFS File-System Exemplo



```
if (!LittleFS.begin(true)) {  
    Serial.println("Erro na montagem do SPIFFS");  
    return;  
}  
// Método para a escrita  
void writeFile(String algumacoisa, String path) {  
    File file = LittleFS.open(path,"w+");  
    if(!file){  
        Serial.println("Erro ao abrir o arquivo!");  
    }  
    else {  
        file.println(algumacoisa);  
    }  
    file.close();  
}
```

SPIFFS File-System Exemplo



```
// Método para a leitura
void readFile(String path) {
    File file = LittleFS.open(path,"r");
    if (!file) {
        Serial.println("Erro ao abrir o arquivo!");
        return;
    }
    else{
        while(file.available()){
            Serial.write(file.read());
        }
        file.close();
    }
}
```

SPIFFS File-System Exemplo



```
// Método para a leitura
void readFile(String path) {
    File file = LittleFS.open(path,"r");
    if (!file) {
        Serial.println("Erro ao abrir o arquivo!");
        return;
    }
    else{
        while(file.available()){
            Serial.write(file.read());
        }
        file.close();
    }
}
```

SPIFFS *File-System* Exemplo



■ Executar Exemplo_ESP8266_WiFi_Web_Server_File_System_01.ino

- Este é o Exemplo_ESP8266_WiFi_Web_Server_01 modificado para ler o arquivo HTML do File System;

Comunicação com a Nuvem ou Névoa



■ Serviço web padrão;

- Protocolo HTTP
- Modelo cliente-servidor
- Através de GET/POST
- Extremamente popular
- Necessidade de verificar regularmente se há dados
- Informações desnecessárias são transmitidas
- Manuseio excessivo de conexões

■ WebSockets;

- Protocolo baseado em TCP/IP
- Full-duplex
- Projetado para navegadores que suportam HTML5
- Conexão permanece ativa após a entrega do dado
- Menos tráfego de dados

Comunicação com a Nuvem ou Névoa



■ Message Queue Telemetry Transport(MQTT);

- Protocolo de transporte de mensagens no modelo publicador-subscritor
- Ideal para a comunicação em IoT
- Baseado em TCP/IP
- Leve, aberto e de fácil implementação
- Criado em 1999 por Andy Stanford-Clark (IBM) and Arlen Nipper (Arcom, agora Cirrus Link)
- Qualidade no serviço da entrega do dado

Comunicação com a Nuvem ou Névoa



■ Serviço web padrão:

- Alguns serviços gratuitos para armazenamento de dados;
 - dweet.io
 - thingspeak.com
 - sensorcloud.com
 - ubidots.com

Serviços web padrão



Utilizando o dweet.io:

- URL para enviar dados;
 - dweet.io/dweet/for/nome-da-coisa?arg1=val1&...&argn=valn
- URL para ler o mais recente dado enviado;
 - dweet.io/get/latest/dweet/for/nome-da-coisa
- URL para ler as últimas ocorrências;
 - dweet.io/get/dweets/for/nome-da-coisa
 - São retornadas até as 5 últimas ocorrências das últimas 24 horas

Exemplo – ESP8266 Serviços *web* padrão



- Uso do dweet.io, enviando dados com o ESP8266:

```
const char* host = "www.dweet.io";

if (client.connect(host, 80)) {
    contaFalhas=0;
    String caminho = "GET /dweet/for/B4HUE RJIoT?temperatura=36"; // Fixei um valor de temperatura
    client.println(caminho);
    client.println();
    Serial.println("[Response]");
    while (client.connected()){
        if (client.available()) {
            String line = client.readStringUntil('\n');
            Serial.println(line);
        }
    }
    client.stop();
    Serial.println("\n[Disconnected]");
}
else {
    client.stop();
    contaFalhas++;
    if (contaFalhas>5) resetFunc();
}
delay(30000);
}
```

Exemplo – ESP8266 Serviços *web* padrão



■ Uso do dweet.io, recebendo dados com o ESP8266, usando JSON:

```
String caminho = "GET /get/latest/dweet/for/B4HUE RJIoT";
client.println(caminho);
client.println();
Serial.println("[Response:]");
while (client.connected()){
    if (client.available()) {
        String line = client.readStringUntil('\n');
        Serial.println(line);
        DeserializationError error = deserializeJson(doc,line);
        if (error) {
            Serial.println("JSON parsing failed!");
        }
        else{
            String entraram = doc["with"][0];
            Serial.println(entraram);
            error = deserializeJson(doc,entraram);

            String content = doc["content"];
            Serial.println(content);

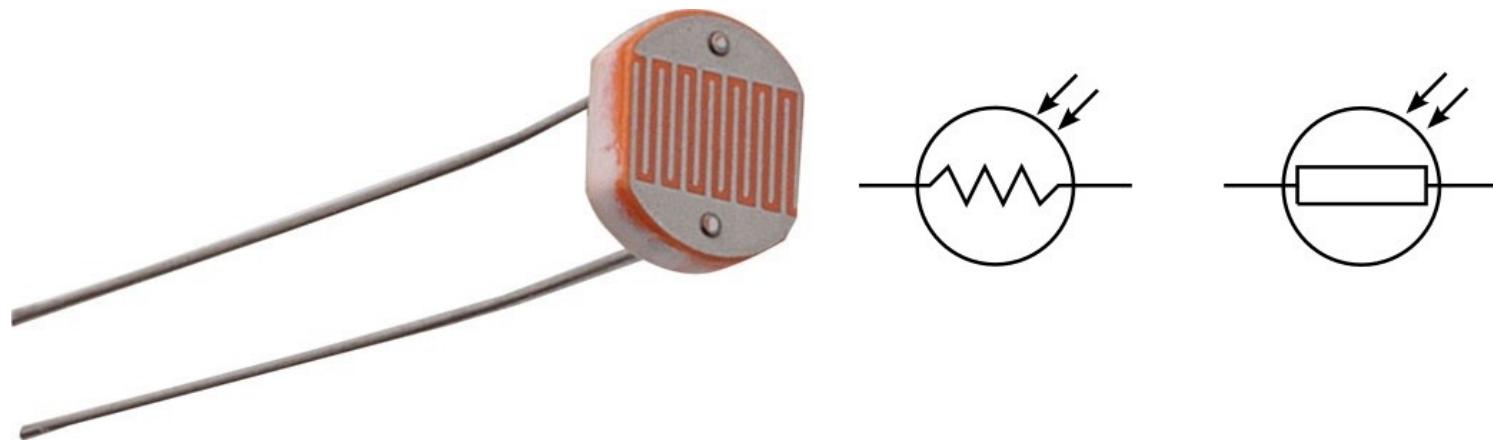
            error = deserializeJson(doc,content);

            String temperatura = doc["temperatura"];
            Serial.println(temperatura);
        }
    }
}
```

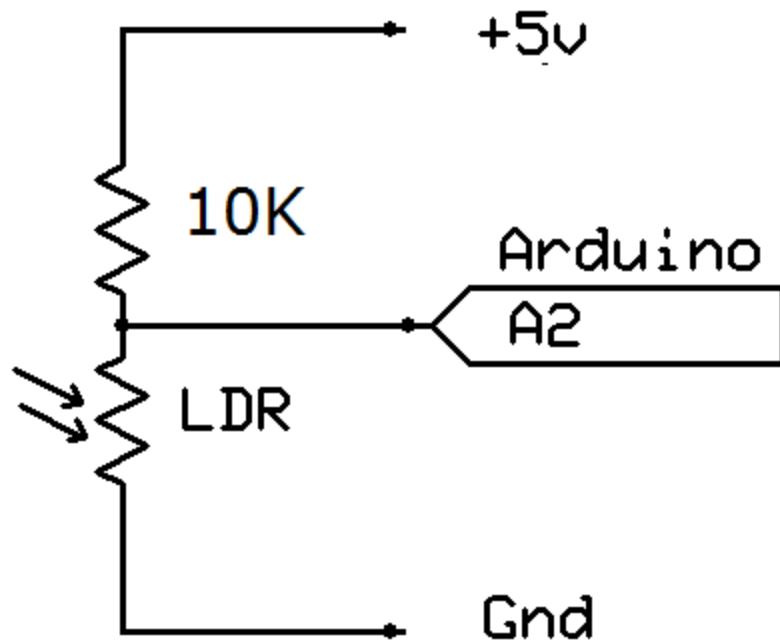
Componente LDR



- LDR é o acrônimo de Light Dependent Resistor;
- Sua resistência é inversamente proporcional a quantidade de luz percebida;
- Em geral, pode variar de 400Ω até $1M\Omega$;
- Age como um sensor analógico de luminosidade;



Componente LDR



Circuito sensor de luminosidade

Exercício B3

dweet publicando



■ Publicar no dweet.io, utilizando o ESP32, o valor da luminosidade do ambiente.

■ Parâmetros:

- Nome-da-Coisa → nome-do-aluno
- Nome-do-Argumento → valorLuminosidade

■ Componentes utilizados:

ESP32

Sensor de luminosidade

Jumpers

Resistor de 10k

Exercício B4

dweet recebendo



■ Receber do dweet.io, utilizando o ESP32, o valor da luminosidade do ambiente. Caso esteja escuro(vocês definem a escuridão!), acenda um led.

■ Parâmetros:

- Nome-da-Coisa → nome-do-aluno
- Nome-do-Argumento → valorLuminosidade

■ Componentes utilizados:

ESP32

led

Jumpers

ESP32- Server Modo Access Point

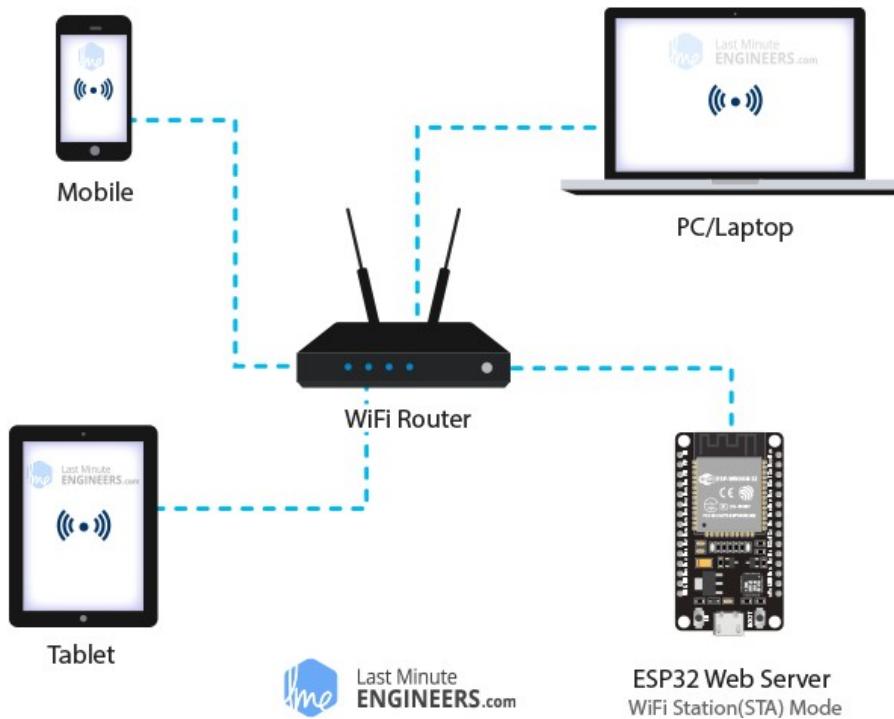


ESP32 - Server Modo Access Point



- Torna a rede com propósito mais específico;
- Isola-se de outras redes;
- SSID e senha próprias;

ESP32- Server Modo Estação



Last Minute
ENGINEERS.com

ESP32 - Server Modo Estação



- Utiliza uma rede pré-existente;
- Mais vulnerável a acesso de intrusos;
- SSID e senha da rede geral;

ESP32/ESP8266 AsyncWebServer



- Permite o tratamento de mais de uma conexão ao mesmo tempo;
- Trata conexões enquanto, em segundo plano, o servidor se encarrega de enviar a resposta para a máquina cliente;
- Possui, também, mecanismo de processamento de templates;
- Como é totalmente assíncrono não é executado na função `loop()`;
- Para usar:
 - Baixar, do Google Drive, a biblioteca **ESPAsyncWebServer**.
 - Baixar, do Google Drive, as bibliotecas **AsyncTCP(ESP32)** e **ESPAsyncTCP(ESP8266)**.

ESP32/ESP8266 AsyncWebServer.h



- AsyncWebServer objeto(porta);
 - Endereço da porta ouvinte do meio;
 - objeto.**begin();**
 - Inicia o serviço;
 - objeto.**onNotFound(função);**
 - objeto.**on(caminho,metodo_HTTP,[]handling_function{callback})**
 - Fará o tratamento das requisições que chegarão. Incorpora, também, as funções de callback;
 - Caminho – é a URL do tratamento da requisição;
 - metodo_HTTP –
 - HTTP_GET,
 - HTTP_POST,
 - HTTP_DELETE,
 - HTTP_PUT,
 - HTTP_HEAD,
 - HTTP_PATCH,
 - **HTTP_OPTIONS**,
 - HTTP_ANY
 - send(codigo_retorno,content-type,conteudo) ou

ESP8266

Exemplo AsyncWebServer



■ Servindo HTML:

```
#include <ESP8266WiFi.h>
#include <ESPAsyncTCP.h>
#include <ESPAsyncWebServer.h>

const char* ssid = "ssssssssssssssssssss";
const char* password = "pppppppppppppp";

int i=0;
AsyncWebServer server(80);

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

    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(1000);
        Serial.println("Connecting to WiFi..");
    }

    Serial.println(WiFi.localIP());

    server.on("/", HTTP_GET, [](){AsyncWebRequest *request){
        Serial.println(i++);
        request->send(200, "text/html", "<font size='40'><p><b>Aqui o HTML!</b></p></font>");
    });
}
```

ESP8266/ESP32

WiFi.softAP()



■ Funcionalidade para definir parâmetros para atuar como access point;

■ Sintaxe:

- softAP(const char* **ssid**, const char* **password**, int **canal** = 1, int **ssid_hidden** = 0, int **max_connection** = 4);
 - **ssid** define o SSID da rede Wi-Fi.
 - **password** define a senha da rede Wi-Fi. Se a rede estiver aberta, defina como NULL.
 - **canal** configura o canal Wi-Fi.
 - **ssid_hidden** define a rede como oculta(int 1).
 - **max_connection** define o número máximo de conexões simultâneas. O padrão é 4.

ESP32

Exemplo como Access Point



■ Servindo HTML como Access Point:

```
#include <WiFi.h>
#include <AsyncTCP.h>
#include <ESPAsyncWebServer.h>

const char* ssid = "ESP32-UERJ-Access-Point";      // Mudem para não haver conflitos
const char* password = "123456789";
int i=0;

AsyncWebServer server(80);

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

  WiFi.mode(WIFI_AP);
  Serial.println("Criando o Access Point");
  WiFi.softAP(ssid,password,7,0,10);
  Serial.print("IP Address:\t");
  Serial.println(WiFi.softAPIP());

server.on("/html", HTTP_GET, [](){
  Serial.println(i++);
  request->send(200, "text/html", "<p><b>This is HTML!</b></p>");
});
server.begin();
}
```



ESP32/ESP8266 AsyncWebServer +

■ Tratando parâmetros passados na URL:

- Para saber se tem o argumento:
 - request -> **hasParam('argumento')**
 - Retorna true ou false.
- Para obter o valor do argumento:
 - AsyncWebParameter* variavel1 = request->**getParam("argumento")**;
 - String variavel2 = variavel1 ->value();

■ Devolvendo JSON:

```
server.on("/", HTTP_GET, [](AsyncWebRequest* request) {  
    AsyncResponseStream* response = request->beginResponseStream("application/json");  
    StaticJsonDocument doc(1024);  
    doc["led1"] = statusLed1;  
    doc["led2"] = statusLed2;  
    serializeJson(doc, *response);  
    request->send(response);  
});
```

■ Redirecionando para outra URL:

```
server.on("/", HTTP_GET, [](AsyncWebRequest * request) {  
    request->redirect("/page2");  
});  
server.on("/page2", HTTP_GET, [](AsyncWebRequest * request) {  
    request->send(200, "text/plain", "Voce esta na page2!");
```

AsyncWebServer com SPIFFS



Para enviar um arquivo:

- request -> send(SPIFFS,'nome-do-arquivo','tipo-do-arquivo',)
- ou
- request -> send(SPIFFS,'nome-do-arquivo','tipo-do-arquivo',false, processor)
 - Utilizado quando o arquivo contém variáveis(%placeholder%) que serão tratadas pela função de nome processor.

Exemplo:

```
<tr>
    <td>Local 1</td>
    <td>%LOCAL1%</td>
</tr>
<tr>
    <td>Local 2</td>
    <td>%LOCAL2%</td> </tr>
<tr>
    <td>Local 3</td>
    <td>%LOCAL3%</td>
</tr>

String processor(const String& var){
    if(var == "LOCAL1"){
        return String(random(10,25));
    }
    if(var == "LOCAL2"){
        return String(random(20,45));
    }
    if(var == "LOCAL3"){

    }
}
```

AsyncWebServer com SPIFFS



Passos para o upload dos arquivos:

- Abrir o *sketch* que tratará os dados;
- Em seguida, teclar a aba *tools/ferramentas* que aparecerá a entrada “**ESP32 Sketch Data Upload**”;
- Clicar sobre esta entrada. A operação de *upload* será iniciada, copiando os arquivos para o ESP32;

Resultado Final(mudará a cada 30 segundos):

Controle Temperaturas

Ambiente	Temperatura
Local 1	18
Local 2	32
Local 3	8

ESP32 AsyncWebServer

Outro Exemplo



- Controlando ambientes usando SPIFFS, arquivos HTML e CSS:
 - Baixar do Google Drive e descompactar a pasta:
 - ESP32_AsyncWebServer_SPIFFS_Ativo_HTM_CSS
 - Colocar leds nos GPIOs 12, 13 e 14;

ESP32 Controle

Status Local 1: **LIGADO**

LIGAR

DESLIGAR

Status Local 2: **DESLIGADO**

LIGAR

DESLIGAR

Status Local 3: **LIGADO**

LIGAR

DESLIGAR

SPIFFS *File-System* *Upload* de arquivos



Passos para o upload dos arquivos:

- Abrir o *sketch* que tratará os dados;
- Em seguida, teclar a aba *tools/ferramentas* que aparecerá a entrada “**ESP32 Sketch Data Upload**”;
- Clicar sobre esta entrada. A operação de *upload* será iniciada, copiando os arquivos para o ESP32;

ESP32 AsyncWebServer

Outro Exemplo



Controlando ambientes usando arquivos HTML e CSS:

```
#include "WiFi.h"
#include "ESPAAsyncWebServer.h"
#include <AsyncTCP.h>
#include "SPIFFS.h"

const char* ssid = "ssssssssssssssss";
const char* password = "pppppppppppppppp";

const int ledPin12 = 12;
const int ledPin13 = 13;
const int ledPin14 = 14;

String ledState12,ledState13,ledState14;

AsyncWebServer server(80);

String processor(const String& var){

  if(var == "LOCAL1"){
    if(digitalRead(ledPin12)){
      ledState12 = "LIGADO";
    }
    else{
      ledState12 = "DESLIGADO";
    }
    return ledState12;
  }
}
```

ESP32 AsyncWebServer

Outro Exemplo



Controlando ambientes usando arquivos HTML e CSS:

```
if(var == "LOCAL2"){
    if(digitalRead(ledPin13)){
        ledState13 = "LIGADO";
    }
    else{
        ledState13 = "DESLIGADO";
    }
    return ledState13;
}
if(var == "LOCAL3"){
    if(digitalRead(ledPin14)){
        ledState14 = "LIGADO";
    }
    else{
        ledState14 = "DESLIGADO";
    }
    return ledState14;
}
return String();
}

void setup(){
Serial.begin(115200);
pinMode(ledPin12, OUTPUT);
pinMode(ledPin13, OUTPUT);
pinMode(ledPin14, OUTPUT);
```

ESP32 AsyncWebServer

Outro Exemplo



Controlando ambientes usando arquivos HTML e CSS:

```
WiFi.begin(ssid, password);
while (WiFi.status() != WL_CONNECTED) {
  delay(1000);
  Serial.println("Conectando ao WiFi..");
}

Serial.println(WiFi.localIP());

// Pagina index.html
server.on("/", HTTP_GET, [](){AsyncWebRequest *request){
  request->send(SPIFFS, "/index.html", String(), false, processor);
});

// Arquivo css
server.on("/style.css", HTTP_GET, [](){AsyncWebRequest *request){
  request->send(SPIFFS, "/style.css", "text/css");
});

// Tratamento GPIO12 - Ligar
server.on("/on1", HTTP_GET, [](){AsyncWebRequest *request){
  digitalWrite(ledPin12, HIGH);
  request->send(SPIFFS, "/index.html", String(), false, processor);
});

// Tratamento GPIO12 - Desligar
server.on("/off1", HTTP_GET, [](){AsyncWebRequest *request){
  digitalWrite(ledPin12, LOW);
```

ESP32 AsyncWebServer

Outro Exemplo



Controlando ambientes usando arquivos HTML e CSS:

```
// Tratamento GPIO13 - Ligar
server.on("/on2", HTTP_GET, [](AsyncWebServerRequest *request){
    digitalWrite(ledPin13, HIGH);
    request->send(SPIFFS, "/index.html", String(), false, processor);
});

// Tratamento GPIO13 - Desligar
server.on("/off2", HTTP_GET, [](AsyncWebServerRequest *request){
    digitalWrite(ledPin13, LOW);
    request->send(SPIFFS, "/index.html", String(), false, processor);
});

// Tratamento GPIO14 - Ligar
server.on("/on3", HTTP_GET, [](AsyncWebServerRequest *request){
    digitalWrite(ledPin14, HIGH);
    request->send(SPIFFS, "/index.html", String(), false, processor);
});

// Tratamento GPIO14 - Desligar
server.on("/off3", HTTP_GET, [](AsyncWebServerRequest *request){
    digitalWrite(ledPin14, LOW);
    request->send(SPIFFS, "/index.html", String(), false, processor);
});

server.onNotFound(error404);

server.begin();
```

Exercício B5

AsyncWebServer



■ Com o ESP8266(Estação):

- Detectar a presença “humana” com o sensor PIR.
- Uma vez detectada, informar ao ESP32.

■ Com o ESP32(Access Point):

- Uma vez recebida a informação da presença, acionar um relé.
- Usar o serviço de rede WiFi do próprio ESP32.

■ Componentes utilizados:

ESP32

ESP8266

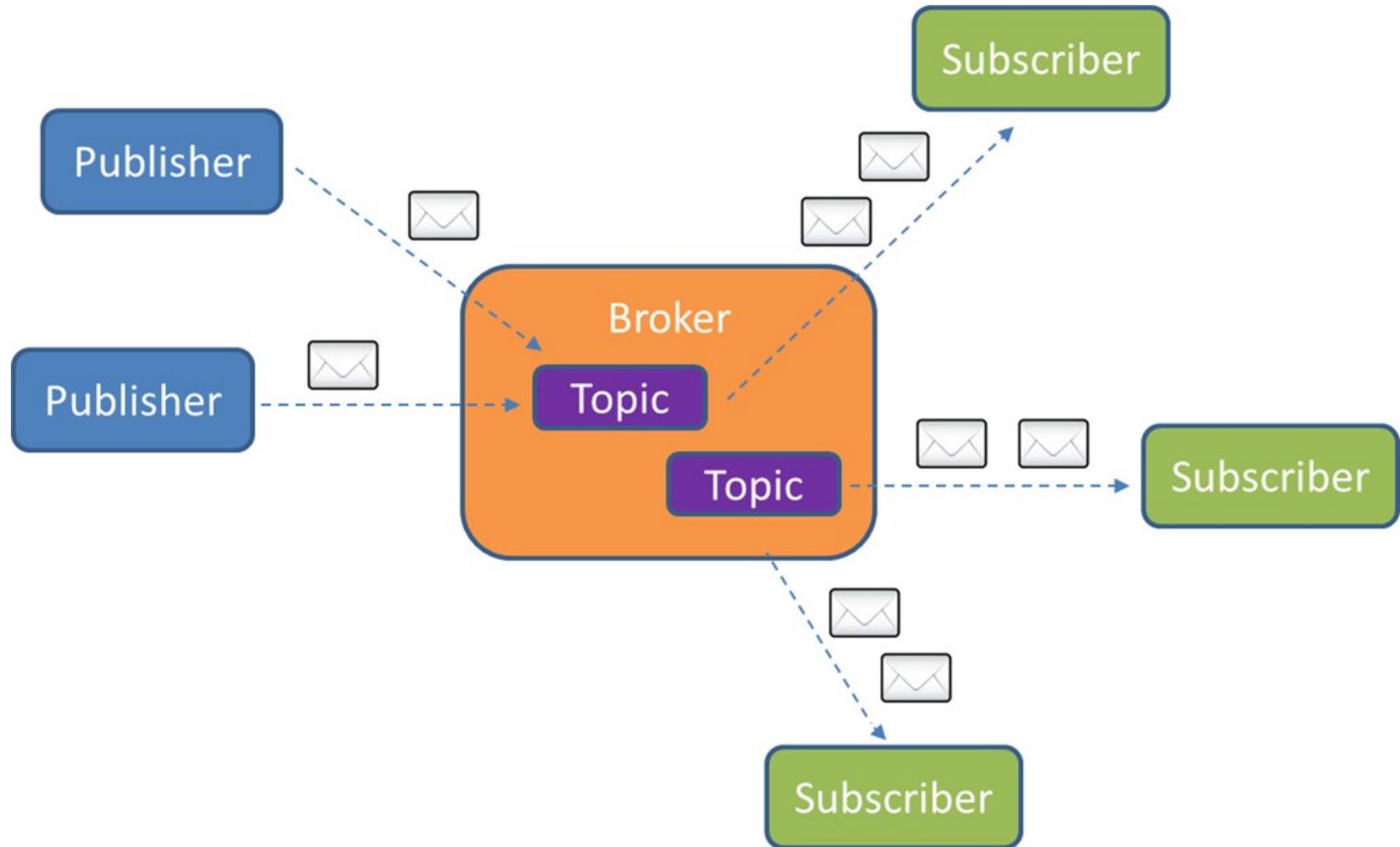
sensor PIR

relé

Jumpers

MQTT

Message Queue Telemetry Transport

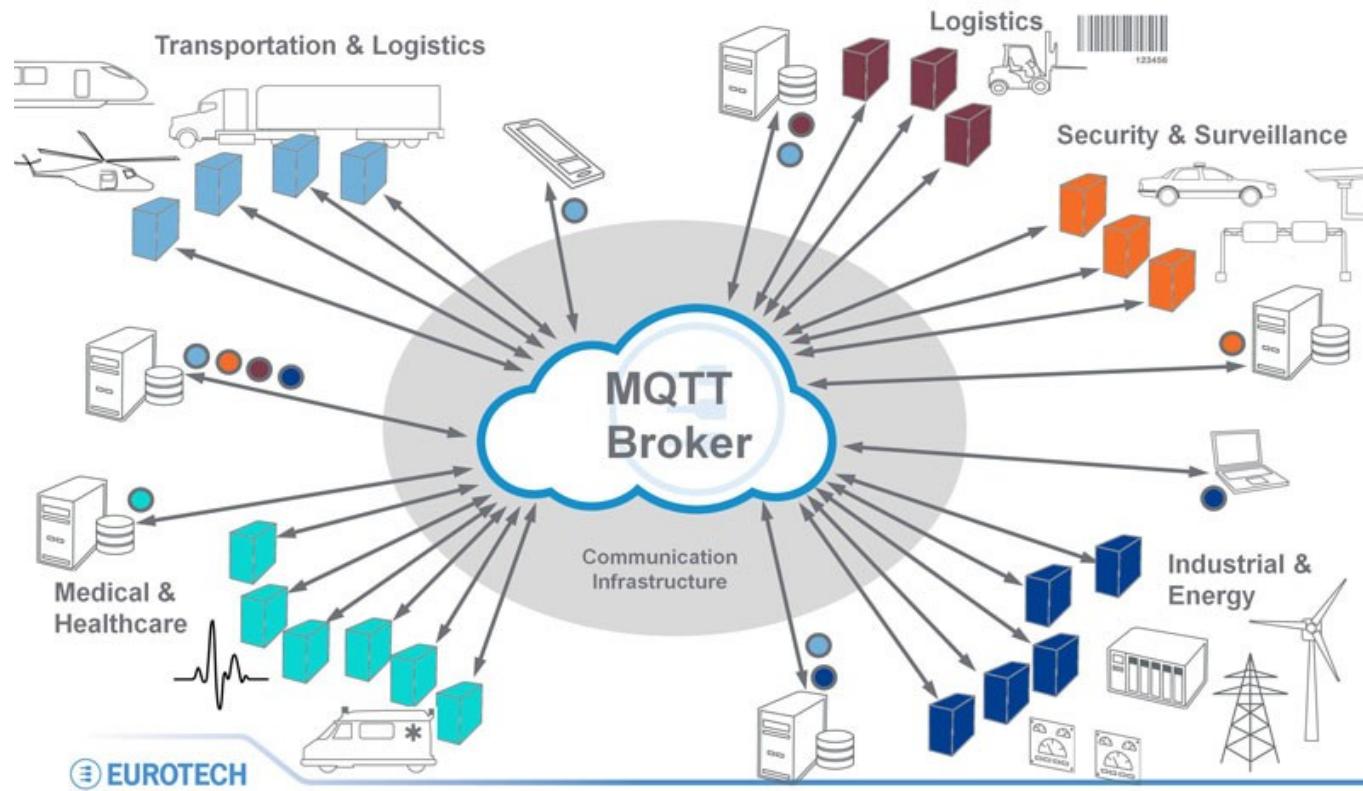


MQTT



The Internet of Things

Decoupling Producers & Consumers of M2M Device Data



MQTT - *Brokers*



■ Alguns *Brokers* disponíveis:

- HiveMq
- CloudMQTT
- **Mosquitto**
- iot.eclipse.org
- Amazon's Aws IoT
- Adafruit IO

MQTT - Mosquitto



- Disponível para *download* em mosquitto.org;
- Mantido pelo grupo do eclipse;
- Suporte às linguagens:
 - C
 - C++
 - C# e .Net
 - Java/Android
 - Phyton
 - JavaScript
 - Go

MQTT - Mosquitto



■ Algumas das funcionalidades:

- Autenticação por usuário/senha
- Especificação da qualidade da transmissão(QoS):
 - 0 – manda, no máximo, uma vez;
 - Não há garantia de entrega. O destinatário não confirma o recebimento da mensagem e a mensagem não é armazenada e retransmitida pelo remetente. **O nível 0 de QoS costuma ser chamado de “dispare e esqueça”.**

MQTT - Mosquitto



■ Especificação da qualidade da transmissão(QoS):

1 – manda, pelo menos uma vez, com a garantia de entrega;

- Garante que uma mensagem seja entregue pelo menos uma vez ao receptor. O remetente armazena a mensagem até receber um pacote PUBACK do destinatário que confirma o recebimento da mensagem. É possível que uma mensagem seja enviada ou entregue várias vezes.
- O remetente usa o identificador de pacote em cada pacote para corresponder o pacote PUBLISH ao pacote PUBACK correspondente. Se o remetente não receber um pacote PUBACK em um período de tempo razoável, o remetente reenvia o pacote PUBLISH. Quando um receptor recebe uma mensagem com QoS 1, ele pode processá-la imediatamente.
- Se o cliente de publicação enviar a mensagem novamente, ele definirá um sinalizador duplicado (DUP). No QoS 1, este sinalizador DUP é usado apenas para fins internos e não é processado pelo broker ou cliente. O destinatário da mensagem envia um PUBACK, independentemente do sinalizador DUP

MQTT - Mosquitto



■ Especificação da qualidade da transmissão(QoS):

2 – manda, somente uma vez, com a garantia de entrega;

- QoS 2 é o mais alto nível de serviço em MQTT. **Este nível garante que cada mensagem seja recebida apenas uma vez pelos destinatários pretendidos. É mais seguro e mais lento.** A garantia é fornecida por pelo menos dois fluxos de solicitação/resposta (um handshake de quatro partes) entre o remetente e o destinatário. O remetente e o destinatário usam o identificador de pacote da mensagem PUBLISH original para coordenar a entrega da mensagem.
- Quando um receptor recebe um pacote de um remetente, ele processa a mensagem de publicação de acordo e responde ao remetente com um pacote PUBREC que reconhece o pacote PUBLISH. Se o remetente não receber um pacote PUBREC do destinatário, ele enviará o pacote PUBLISH novamente com um sinalizador duplicado (DUP) até receber uma confirmação.
- Depois que o remetente recebe um pacote PUBREC do destinatário, o remetente pode descartar com segurança o pacote PUBLISH inicial. O remetente armazena o pacote PUBREC do destinatário e responde com um pacote PUBREL.
- Depois que o receptor recebe o pacote PUBREL, ele pode descartar todos os estados armazenados e responder com um pacote PUBCOMP (o mesmo ocorre quando o remetente recebe o PUBCOMP). Até que o destinatário conclua o processamento e envie o pacote PUBCOMP de volta ao remetente, o destinatário armazena uma referência ao identificador de pacote do pacote PUBLISH original. Esta etapa é importante para evitar o processamento da mensagem uma segunda vez. Após o remetente receber o pacote PUBCOMP, o identificador de pacote da mensagem publicada fica disponível para reutilização.

MQTT - Mosquitto



■ QoS - Considerações:

- O cliente que publica a mensagem para o *broker* define o nível de QoS da mensagem ao enviar a mensagem para o *broker*;
- O *broker* transmite esta mensagem aos clientes assinantes usando o nível de QoS que cada cliente assinante define durante o processo de assinatura;
- Se o cliente assinante definir uma QoS inferior à do cliente publicador, o **broker** transmitirá a mensagem com qualidade de serviço inferior;
- O QoS dá ao cliente o poder de escolher um nível de serviço que corresponda à confiabilidade da rede e à lógica do aplicativo;
- Como o MQTT gerencia a retransmissão de mensagens e garante a entrega (mesmo quando o transporte subjacente não é confiável), a QoS facilita muito a comunicação em redes não confiáveis.

MQTT - Mosquitto



■ QoS – Qual escolher:

Use QoS 0 quando:

Você tem uma conexão completamente ou quase estável entre o remetente e o destinatário. Você não se importa se algumas mensagens forem perdidas ocasionalmente. **A perda de algumas mensagens pode ser aceitável se os dados não forem tão importantes ou quando os dados forem enviados em intervalos curtos.** Você não precisa de enfileiramento de mensagens. As mensagens são enfileiradas apenas para clientes desconectados se tiverem QoS 1 ou 2 e uma sessão persistente.

Use QoS 1 quando:

Você precisa obter todas as mensagens e seu caso de uso pode lidar com duplicatas. O nível 1 de QoS é o nível de serviço usado com mais frequência porque garante que a mensagem chegue pelo menos uma vez, mas permite várias entregas. **Obviamente, seu aplicativo deve tolerar duplicatas e ser capaz de processá-las adequadamente.**

Use QoS 2 quando:

É fundamental para seu aplicativo receber todas as mensagens exatamente uma vez. Geralmente, esse é o caso se uma entrega duplicada puder prejudicar os usuários do aplicativo ou os clientes assinantes. Esteja ciente da sobrecarga e de que a interação QoS 2 leva mais tempo para ser concluída.

MQTT - Mosquitto



■ Algumas das funcionalidades:

- Guarda da última mensagem recebida(Retain);
 - Até que a próxima mensagem seja publicada, o cliente assinante está totalmente no escuro sobre o status atual do tópico. Essa situação é onde as mensagens retidas entram em ação.
- Mensagem específica para indisponibilidade(willMessage);
 - O *broker* informa que o publicador está fora do ar.
- Arquivo com opções de configuração;
- Suporte à criptografia usando as opções baseadas em certificados SSL / TLS;
- Porta padrão 1883;
- Tamanho padrão das mensagens → 128 bytes
- *Keep alive* padrão → 15 segundos(para alterar, editar PubSubClient.h)

MQTT - PubSubClient



- Biblioteca padrão, PubSubClient, para ESP32/ESP8266, em github;
 - Baixar do Google Drive.
- Documentação da API em pubsubclient.knolleary.net/api.html;
- Algumas de suas funções:
 - PubSubClient() ou PubSubClient(client);
 - connect(clientID); ou connect(clientID,username, pswd); ou connect(clientID,username,pswd,willTopic,willQoS,willRetain,willMessage);
 - clientID – identifica o cliente na rede;
 - username – nome do usuário;
 - pswd – senha;
 - willTopic – nome do tópico associado a uma queda;
 - willQoS - qualidade do serviço(0, 1 ou2);
 - willRetain – se é para reter a mensagem;
 - willMessage – mensagem a ser enviada quando ocorrer a queda;
 - disconnect();

MQTT - PubSubClient



- publish(topic,payload) ou publish(topic,payload,retain);
 - topic – nome do tópico para publicar a mensagem
 - payload – mensagem a ser publicada
 - retained – se mensagem é para ficar retida ou não
 - Retorna true ou false;
- subscribe(topic[,qos]);
 - topic – tópico no qual se inscreve
 - Retorna true ou false;
- unsubscribe(topic);
- loop();
 - Chamado para possibilitar ao cliente processar as mensagens que chegam do tópico assinado;
 - Retorna true ou false, em relação à conexão. Se há mensagens é acionada a função de callback ;
 - Podem existir instruções seguintes ao loop().**
- connected();
 - Checa se o cliente está conectado ao servidor. Retorna true ou false;
- state();
 - Retorna o estado corrente do cliente;
- setCallback(callback);
 - Nome da função que será acionada quando existirem mensagens para o assinante;

MQTT

Códigos de Retorno



```
int state()
```

Returns the current state of the client. If a connection attempt fails, this can be used to get more information about the failure.

All of the values have corresponding constants defined in `PubSubClient.h`.

Returns

- -4 : `MQTT_CONNECTION_TIMEOUT` - the server didn't respond within the keepalive time
- -3 : `MQTT_CONNECTION_LOST` - the network connection was broken
- -2 : `MQTT_CONNECT_FAILED` - the network connection failed
- -1 : `MQTT_DISCONNECTED` - the client is disconnected cleanly
- 0 : `MQTT_CONNECTED` - the client is connected
- 1 : `MQTT_CONNECT_BAD_PROTOCOL` - the server doesn't support the requested version of MQTT
- 2 : `MQTT_CONNECT_BAD_CLIENT_ID` - the server rejected the client identifier
- 3 : `MQTT_CONNECT_UNAVAILABLE` - the server was unable to accept the connection
- 4 : `MQTT_CONNECT_BAD_CREDENTIALS` - the username/password were rejected
- 5 : `MQTT_CONNECT_UNAUTHORIZED` - the client was not authorized to connect

ESP8266 Mosquitto

Exemplo 1 - Publicando



■ Publicando no Mosquitto, um tópico, com o ESP8266:

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

const char* ssid = "ssssssssssssssssssssss";
const char* password = "pppppppppppppppppp";
//const char* mqtt_server = "192.168.1.65";      // Endereço do Mosquitto - Broker
const char* mqtt_server = "test.mosquitto.org";   // Endereço do Mosquitto – Broker - Gratuito
char msg[50];

WiFiClient espClient;
PubSubClient client(espClient);

void setup(){
    Serial.begin(115200);
    Serial.print("Conectando a Rede WiFi ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);
    while (WiFi.status() != WL_CONNECTED) {
        Serial.print('.');
        delay(500);
    }
    Serial.println("Conectado ");
    Serial.print("IP Address ");
    Serial.println(WiFi.localIP());
    Serial.println("Conectando ao Mosquitto");
    client.setServer(mqtt_server, 1883);
```

ESP8266 Mosquitto

Exemplo 1 - Publicando



■ Publicando no Mosquitto, um tópico, com o ESP8266:

```
void loop(){
    String temperatura =(String) random(10,45);
    temperatura.toCharArray(msg,temperatura.length()+1);
publica(msg);
    delay(10000);
}

void publica(char* msg){
    if (!client.connected()){
        conecta();
    }
    client.publish("temperatura_ambiente_01",msg,true);
    Serial.println("Mensagem publicada");
    //client.disconnect(); Utilizar quando for um publicação que independe do tempo, ex: invasão de ambiente
}
void conecta(){
    while (!client.connected()) {
        if (client.connect("Esp8266 Publicando","","","", "Controle_Comodo_01",1,false,"Fora do Ar")){
            Serial.println("Cliente Conectado ao Mosquitto");
        }
        else{
            Serial.println(client.state());
            delay(2000);
        }
    }
}
```

ESP32 Mosquitto

Exemplo 1 - Assinando



■ Assinando no Mosquitto com o ESP32, um tópico:

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

const char* ssid = "ssssssssssssssssssss";
const char* password = "pppppppppppppppp";
//const char* mqtt_server = "192.168.1.65"; // Endereço do Mosquitto - Broker
const char* mqtt_server = "test.mosquitto.org"; // Endereço do Mosquitto - Broker

WiFiClient espClient;
PubSubClient client(espClient);

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

    WiFi.begin(ssid,password);

    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
    }
    client.setServer(mqtt_server, 1883); // Conecta-se a porta 1883 do Mosquitto
    client.setCallback(callback); // Indica que vai tratar o retorno
}
```

ESP32 Mosquitto

Exemplo 1 - Assinando



■ Assinando no Mosquitto com o ESP32, um tópico:

```
void reconnect() {  
    while (!client.connected()) {  
        Serial.print("Conectando.....");  
  
        if (client.connect("ESP32 Assinando")) { // Trocar pela sua identificação  
            Serial.println("Conectado");  
            client.subscribe("temperatura_ambiente_01"); // Subscrevendo  
            client.subscribe("Controle_Comodo_01"); // Subscrevendo  
        } else {  
            Serial.print("failed, rc=");  
            Serial.print(client.state());  
            delay(5000);  
        }  
    }  
}  
  
void loop(){  
    if (!client.connected()) {  
        reconnect();  
    }  
}
```

ESP32 Mosquitto

Exemplo 1 - Assinando



Assinando no Mosquitto com o ESP32, um tópico:

```
void callback(char* topic, byte* payload, unsigned int length) {  
    String valor;  
    Serial.print("Messagem do topico :");  
    Serial.print(topic);  
    Serial.print(" - ");  
    if (String(topic) == "temperatura_ambiente_01"){  
        for (int i=0;i<length;i++) {  
            valor.concat((char)payload[i]);  
        }  
        Serial.println(valor);  
    }  
    else{  
        Serial.println("Publicador fora do AR!!!");  
    }  
}
```

Executar Exemplo_ESP32_WiFi_Mosquitto_Aassinando_1_Topicoino

ESP8266 Mosquitto

Exemplo 2 - Publicando



■ Publicando no Mosquitto, dois tópicos, com o ESP8266:

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

const char* ssid = "ssssssssssssssssssssssssss";
const char* password = "pppppppppppppppppp";
//const char* mqtt_server = "152.92.181.90"; // Endereço do Mosquitto – Broker
const char* mqtt_server = "test.mosquitto.org"; // Endereço do Mosquitto - Broker
char msg1[50];
char msg2[50];

WiFiClient espClient;
PubSubClient client(espClient);

void setup(){
    Serial.begin(115200);
    Serial.print("Conectando a Rede WiFi ");
    Serial.println(ssid);
    WiFi.begin(ssid, password);

    while (WiFi.status() != WL_CONNECTED) {
        Serial.println(client.state());
        delay(500);
    }
    Serial.println("Conectado ");
    Serial.print("IP Address do ESP8266: ");
    Serial.println(WiFi.localIP());
```

ESP8266 Mosquitto

Exemplo 2 - Publicando



■ Publicando no Mosquitto, dois tópicos, com o ESP8266:

```
void publica(char* msg1, char* msg2){  
    if (!client.connected()){  
        conecta();  
    }  
    client.publish("temperatura_ambiente_01",msg1,true);  
    client.publish("umidade_solo_01",msg2,true);  
    Serial.println("Mensagens publicadas!!!");  
    //client.disconnect(); Utilizar quando for um publicação que independe do tempo, ex: invasão de ambiente  
}  
  
void conecta(){  
    while (!client.connected()) {  
        if (client.connect("Esp8266 Publicando","","","", "Controle_Ambiente_01",1,true,"Fora do Ar")){  
            Serial.println("Cliente Conectado ao Mosquitto");  
        }  
        else{  
            Serial.println(client.state());  
            delay(2000);  
        }  
    }  
}  
void loop(){  
    String temperatura =(String) random(10,45);      // Valor arbitrado randomicamente  
    temperatura.toCharArray(msg1,temperatura.length()+1);  
    String umidadeSolo =(String) random(80,320);      // Valor arbitrado randomicamente  
    umidadeSolo.toCharArray(msg2,umidadeSolo.length()+1);  
}
```

ESP32 Mosquitto

Exemplo 2 - Assinando



■ Assinando no Mosquitto com o ESP32, dois tópicos:

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

const char* ssid = "ssssssssssssssssssssssss";
const char* password = "pppppppppppppppppp";
//const char* mqtt_server = "192.168.1.65"; // Endereço do Mosquitto - Broker
const char* mqtt_server = "test.mosquitto.org"; // Endereço do Mosquitto - Broker

WiFiClient espClient;
PubSubClient client(espClient);

void setup(){
    Serial.begin(115200);
    WiFi.begin(ssid,password);
    while (WiFi.status() != WL_CONNECTED) {
        delay(500);
    }
    client.setServer(mqtt_server, 1883); // Conecta-se a porta 1883 do Mosquitto
    client.setCallback(callback); // Indica o nome da função que vai tratar o retorno
}
```

ESP32 Mosquitto

Exemplo 2 - Assinando



■ Assinando no Mosquitto com o ESP32, dois tópicos:

```
void callback(char* topic, byte* payload, unsigned int length) {  
    String valorTemperatura;  
    String valorUmidade;  
    Serial.print("Messagem do topico :");  
    Serial.print(topic);  
    Serial.print(" - ");  
    if (String(topic) == "temperatura_ambiente_01"){  
        for (int i=0;i<length;i++) {  
            valorTemperatura.concat((char)payload[i]);  
        }  
        Serial.println(valorTemperatura);  
    }  
    else if (String(topic) == "umidade_solo_01"){  
        for (int i=0;i<length;i++) {  
            valorUmidade.concat((char)payload[i]);  
        }  
        Serial.println(valorUmidade);  
    }  
    else{  
        Serial.println("Publicador fora do AR!!!");  
    }  
}
```

Exemplo - Mosquitto com Java



■ Código Java para publicar e subscrever:

```
import org.eclipse.paho.client.mqttv3.MqttClient;
import org.eclipse.paho.client.mqttv3.MqttConnectOptions;
import org.eclipse.paho.client.mqttv3.MqttException;
import org.eclipse.paho.client.mqttv3.MqttMessage;
import org.eclipse.paho.client.mqttv3.MqttCallback;
import org.eclipse.paho.client.mqttv3.IMqttDeliveryToken;
import org.eclipse.paho.client.mqttv3.persist.MemoryPersistence;
import java.util.Arrays;

public class TestaMQTTCliente_02 {
    public static void main(String[] args) {
        String topic      = "acender_comodo_01";
        String topicOut    = "offline";
        String content     = "1";
        int qos           = 2;
        String broker      = "tcp://localhost:1883";
        String clientId    = "Java_Acendedor_Commando";
```

Exemplo - Mosquitto com Java



■ Código Java para publicar e subscrever:

```
try {  
    MqttClient sampleClient = new MqttClient(broker, clientId);  
    MqttConnectOptions connOpts = new MqttConnectOptions();  
    connOpts.setPassword("usuario".toCharArray());  
    connOpts.setUserName("senha");  
    System.out.println("Conectando ao Mosquitto: " + broker);  
    sampleClient.connect(connOpts);  
    System.out.println("Conectado");  
    System.out.println("Publicando a mensagem: " + content);  
    MqttMessage message = new MqttMessage(content.getBytes());  
    message.setQos(qos);          // Estabelece nível de qualidade  
    message.setRetained(true);    // Estabelece a retenção da mensagem  
    sampleClient.publish(topic, message); // Publica a mensagem  
    System.out.println("Mensagem publicada.");  
    sampleClient.subscribe(topicOut); // Subscreve para quando alguém ficar offline
```

Exemplo - Mosquitto com Java



■ Código Java para publicar e subscrever:

```
sampleClient.setCallback(new MqttCallback() {  
    @Override  
        public void connectionLost(Throwable cause) { //Called when the client lost the connection to  
        the broker  
            }  
    @Override  
        public void messageArrived(String topic, MqttMessage message) throws Exception {  
            System.out.println(topic + ": " + Arrays.toString(message.getPayload()));  
            }  
    @Override  
        public void deliveryComplete(IMqttDeliveryToken token) {//Called when a outgoing publish is  
        complete  
            }  
    );  
    // sampleClient.disconnect();  
    // System.exit(0);  
} catch(MqttException me) {  
    me.printStackTrace();  
}  
}  
}
```

Obs.: Este mesmo código foi utilizado para publicar por uma aplicação Android.

Mosquitto - prompt



```
1491333618: Sending PINGRESP to Arduino Ethernet
1491333622: Received PINGREQ from weMos Cliente_B4H
1491333622: Sending PINGRESP to weMos Cliente_B4H
1491333633: Received PINGREQ from Arduino Ethernet
1491333633: Sending PINGRESP to Arduino Ethernet
1491333637: Received PINGREQ from weMos Cliente_B4H
1491333637: Sending PINGRESP to weMos Cliente_B4H
1491333644: Received PINGREQ From Java_Acendedor_Commando
1491333644: Sending PINGRESP to Java_Acendedor_Commando
1491333648: New connection from 192.168.0.6 on port 1883.
1491333648: Client weMos Cliente_B4H already connected, closing old connection.
1491333648: New client connected from 192.168.0.6 as weMos Cliente_B4H (c2, k15,
ukikopereira).
1491333648: Sending CONNACK to weMos Cliente_B4H (0)
1491333648: Received SUBSCRIBE from weMos Cliente_B4H
1491333648: acender_comodo_01 (QoS 0)
1491333648: weMos Cliente_B4H 0 acender_comodo_01
1491333648: Sending SUBACK to weMos Cliente_B4H
1491333648: Sending PUBLISH to weMos Cliente_B4H (d0, q0, r1, m0, 'acender_comod
0_01', ... (1 bytes))
1491333654: Client Arduino Ethernet has exceeded timeout, disconnecting.
1491333654: Sending PUBLISH to Java_Acendedor_Commando (d0, q2, r0, m9, 'offline'
..., (1 bytes))
1491333654: Received PUBREC from Java_Acendedor_Commando (Mid: 9)
1491333654: Sending PUBREL to Java_Acendedor_Commando (Mid: 9)
1491333654: Received PUBCOMP from Java_Acendedor_Commando (Mid: 9)
1491333663: Received PINGREQ from weMos Cliente_B4H
1491333663: Sending PINGRESP to weMos Cliente_B4H
1491333671: New connection from 192.168.0.100 on port 1883.
1491333671: New client connected from 192.168.0.100 as Arduino Ethernet (c2, k15
, ukikopereira).
1491333671: Sending CONNACK to Arduino Ethernet (0)
1491333671: Received SUBSCRIBE from Arduino Ethernet
1491333671: acender_comodo_01 (QoS 0)
1491333671: Arduino Ethernet 0 acender_comodo_01
1491333671: Sending SUBACK to Arduino Ethernet
1491333671: Sending PUBLISH to Arduino Ethernet (d0, q0, r1, m0, 'acender_comodo
_01', ... (1 bytes))
1491333675: New connection from 192.168.0.6 on port 1883.
1491333675: Client weMos Cliente_B4H already connected, closing old connection.
1491333675: New client connected from 192.168.0.6 as weMos Cliente_B4H (c2, k15,
ukikopereira).
1491333675: Sending CONNACK to weMos Cliente_B4H (0)
1491333675: Received SUBSCRIBE from weMos Cliente_B4H
1491333675: acender_comodo_01 (QoS 0)
1491333675: weMos Cliente_B4H 0 acender_comodo_01
1491333675: Sending SUBACK to weMos Cliente_B4H
1491333675: Sending PUBLISH to weMos Cliente_B4H (d0, q0, r1, m0, 'acender_comod
0_01', ... (1 bytes))
1491333686: Received PINGREQ from Arduino Ethernet
1491333686: Sending PINGRESP to Arduino Ethernet
1491333690: Received PINGREQ from weMos Cliente_B4H
1491333690: Sending PINGRESP to weMos Cliente_B4H
1491333701: Received PINGREQ from Arduino Ethernet
1491333701: Sending PINGRESP to Arduino Ethernet
1491333705: Received PINGREQ from weMos Cliente_B4H
1491333705: Sending PINGRESP to weMos Cliente_B4H
1491333714: Received PINGREQ from Java_Acendedor_Commando
1491333714: Sending PINGRESP to Java_Acendedor_Commando
```

Exercício B6

MQTT



Com o ESP8266(Estação):

- Detectar uma presença humana com o sensor PIR.
- Uma vez detectada, publicar um tópico no Mosquitto.

Com o ESP32(Estação):

- Assinar o tópico no Mosquitto.
- Verificar se houve invasão, caso positivo, acionar o relé.

Componentes utilizados:

ESP32

ESP8266

sensor PIR

relé

Jumpers

ESP-NOW



- Protocolo desenvolvido pela Espressif ;
- Suportado pelo ESP8266 e pelo ESP32;
- Permite que múltiplos dispositivos se comuniquem na ausência de, ou sem usar, uma rede WiFi ;
- Atua na frequência de baixo consumo de 2.4GHz, com velocidade até 10 vezes superior à do WiFi;
- Implementa o conceito de *master-slave*, para a comunicação;
- Um *master* suporta até 20 *slaves*;
- O pareamento entre dispositivos é necessário antes de sua comunicação;
- Após o pareamento, a conexão é segura e ponto a ponto, sem necessidade do *handshake*;

ESP-NOW



- Mensagem enviada pode ter até 250 bytes;
- Pares criptografados limitados;
- No máximo 10 pares criptografados são suportados no modo Estação;
- No máximo 6 no modo SoftAP ou SoftAP + Estação;
- Pode ter comunicação unidirecional ou bidirecional em diferentes configurações;
- Para se comunicar, necessário saber o endereço MAC da máquina receptora;
- Após emparelhar um dispositivo entre si, a conexão é persistente;
- Caso uma de suas placas perca energia ou reinicie, ao reiniciar, ela se conectará automaticamente ao seu par para continuar a comunicação;

ESP-NOW

Modos de Comunicação



ESP-NOW macAddress



- Código exemplo para descobrir o macAddress da máquina:

```
#include "WiFi.h"  
ou  
#include "ESP8266WiFi.h"  
  
void setup(){  
    Serial.begin(115200);  
    Serial.println();  
    Serial.println(WiFi.macAddress());  
}  
void loop(){  
}
```



ESP-NOW

funções de uso

- **esp_now_init();**

Inicia o serviço;

Se OK retorna ESP_OK;

- **esp_now_add_peer(uint8_t* mac_addr, uint8_t role, uint8_t channel, uint8_t* key, uint8_t key_len);**

Adiciona um *slave* à tabela de comunicação;

*uint8_t** *mac_addr*: MAC *address* do par de comunicação;

uint8_t *role*: propósito do par de comunicação (0=ocioso, 1=*master*, 2=*slave* e 3=*master + slave*);

uint8_t *channel*: canal de comunicação, de 1 a 13;

*uint8_t** *key*: 16 bytes com a chave de comunicação. NULL indica sem chave;

uint8_t *key_len*: tamanho da chave de comunicação. Atualmente 16 bytes;

- **esp_now_is_peer_exist(uint8_t* mac_addr);**

Verifica se o *slave* está na tabela de comunicação;

*uint8_t** *mac_addr*: MAC *address* do par que se deseja comprovar se está incluído na tabela de comunicação;

Retorna *true* ou *false*;

- **esp_now_del_peer(uint8_t* mac_addr);**

Apaga uma entrada da tabela de comunicação;

*uint8_t** *mac_addr*: MAC *address* a ser apagado da tabela de comunicação;



ESP-NOW

funções de uso

- **esp_now_send(uint8_t* da, uint8_t* data, uint8_t* len);**

Envia os dados para os destinatários;

uint8_t* da: MAC address para onde irão os dados.

uint8_t* data: pacote de dados para enviar;

uint8_t len: tamanho do pacote de dados;

Retorna resultado da operação(*)

- **esp_now_register_send_cb([](uint8_t* mac_addr, uint8_t status){instruções})**

Comprova se o pacote enviado foi recebido com sucesso pelo par;

uint8_t* mac_addr: MAC address do par de comunicação;

uint8_t status: informação sobre a recepção (0 = OK; 1 = NOK);

- **esp_now_register_recv_cb([](uint8_t* mac_addr, uint8_t* data, uint8_t len){instruções})**

Trata um pacote de dados enviado por um par;

uint8_t* mac_addr: MAC address do par de comunicação;

uint8_t* data: pacote de dados recebido;

uint8_t len: tamanho do pacote de dados recebido;

instruções: o que fazer com o pacote de dados;

Retorna *true* ou *false*;

ESP-NOW

Para enviar os dados



■ Instruções para montar a tabela com os receptores dos dados:

- `esp_now_peer_info_t` peerInfoVariavel
 - Contém três propriedades:
 - `peer_addr` – são os endereços mac de quem recebe
 - `peer_channel` – canal de transmissão
 - `peer_encrypt` – se há encriptação, true ou false
 - Instrução para alimentar o `peer_addr` com os mac
 - `memcpy(peerInfoVariavel.peer_addr,macSlaves,6);`
 - » `macSlaves` – array com os endereços mac

■ Verificando se montou corretamente, exemplo:

- ```
if (esp_now_add_peer(&peerInfoVariavel) != ESP_OK){
 Serial.println("Erro na montagem do peer!!!");
 return;
}
```

# ESP-NOW

## Para enviar os dados



- Exemplo de montagem dos mac addresses:

```
uint8_t macSlaves[][][6] = {{0x30, 0xAE ,0xA4, 0x88, 0x98, 0xD4},
{0x24,0x0A, 0xC4,0xAE,0xAE,0x6C},{0x30, 0xAE ,0xA4, 0x88, 0x9C, 0xE8}};
```

Ou , se for broadcast:

```
uint8_t macSlaves[][][6] = {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF};
```

# ESP-NOW

# Para enviar os dados



- Estrutura para abrigar os dados, exemplo:

```
typedef struct dados_struct {
 float temperatura;
 float umidade_solo;
} dados_struct;
```

```
dados_struct dados;
dados.temperatura = 36.7;
dados.umidade_solo = 89.5;
```

```
esp_now_send(macSlaves,(uint8_t *) &dados,sizeof(dados_struct));
```

# ESP-NOW

# Para receber os dados



- Instrução para receber os dados, exemplo:

```
typedef struct dados_struct {
 float temperatura;
 float umidade_solo;
} dados_struct;

dados_struct dados;

esp_now_register_recv_cb(callback);

void callback(uint8_t * mac_addr, uint8_t *incomingData, uint8_t len) {
 memcpy(&dados, incomingData, sizeof(dados));
 Serial.println(dados.temperatura);
 Serial.println(dados.umidade_solo);
}
}
```

# ESP-NOW



## ■ Passos para agir como *master*:

- iniciar o serviço;
- montar a tabela de *slaves*, com seus respectivos MAC addresses;
- indicar se quer comprovar a chegada das mensagens nos *slaves*(callback);
- enviar a mensagem;

## ■ Passos para agir como *slave*:

- iniciar o serviço;
- receber e tratar a mensagem;

# ESP – NOW - Exemplo 1 mandando - *broadcast*



```
#include <esp_now.h>
#include <WiFi.h>

uint8_t macSlaves[][6] = { {0xFF, 0xFF, 0xFF, 0xFF, 0xFF, 0xFF} };

typedef struct dados_struct {
 float temperatura;
 float umidade_solo;
} dados_struct;

esp_now_peer_info_t peerInfo;

int slavesCount;
```

# ESP – NOW - Exemplo 1 mandando - *broadcast*



```
void setup() {

Serial.begin(115200);
WiFi.mode(WIFI_STA);

if (esp_now_init() != ESP_OK) {
 Serial.println("Erro Iniciação do ESP-NOW!!!");
 ESP.restart();
}
else {
 Serial.println("ESP-NOW OK!!!");
}
slavesCount = sizeof(macSlaves)/6/sizeof(uint8_t);

for(int i=0; i<slavesCount; i++){
 peerInfo.channel = 1;
 peerInfo.encrypt = false;
 memcpy(peerInfo.peer_addr, macSlaves[i], sizeof(macSlaves[i]));
 if (esp_now_add_peer(&peerInfo) != ESP_OK){
 Serial.println("Erro na montagem do peer!!!");
 return;
 }
 else{
 Serial.println("Montagem OK!!!");
 }
}
```

# ESP – NOW - Exemplo 1 mandando - *broadcast*



```
void send(){

esp_err_t result;
dados_struct dados;
dados.temperatura = random(26,42);
dados.umidade_solo = random(-4,100);

for(int i=0; i<slavesCount; i++){
 result = esp_now_send(macSlaves[i],(uint8_t *) &dados, sizeof(dados_struct));
 if (result == ESP_OK) {
 Serial.println("Envio Sucesso - ");
 }
 else {
 Serial.println("Envio Fracasso - ");
 }
}

void loop() {
 send();
 delay(10000);
}
```

# ESP – NOW - Exemplo 1 recebendo - *broadcast*



```
#include <espnow.h> // Atenção!!!
#include <ESP8266WiFi.h>

typedef struct dados_struct {
 float temperatura;
 float umidade_solo;
} dados_struct;

dados_struct dados;
```

# ESP – NOW - Exemplo 1 recebendo - *broadcast*



```
#include <espnow.h> // Atenção!!!
#include <ESP8266WiFi.h>
```

```
typedef struct dados_struct {
 float temperatura;
 float umidade_solo;
} dados_struct;
```

```
dados_struct dados;
```

# ESP – NOW - Exemplo 1 recebendo - *broadcast*



```
void setup() {

 Serial.begin(115200);
 WiFi.mode(WIFI_STA);

 Serial.print("Mac Address Slave:");
 Serial.println(WiFi.macAddress());

 if (esp_now_init() != 0) {
 Serial.println("Erro Iniciação do ESP-NOW!!!");
 ESP.restart();
 }
 else {
 Serial.println("ESP-NOW OK!!!");
 }

 esp_now_register_recv_cb(OnDataRecv);

}
```

# **ESP - NOW - Exemplo 1**

## **recebendo - *broadcast***



```
void OnDataRecv(uint8_t * mac_addr, uint8_t *incomingData, uint8_t len) {

 char macStr[18];

 sprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x", mac_addr[0],
 mac_addr[1],mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);

 Serial.print("Tamanho da Mensagem : ");
 Serial.println(len);
 Serial.println("");
 Serial.print("Mac do Master : ");
 Serial.println(macStr);
 Serial.println("");
 memcpy(&dados, incomingData, sizeof(dados));

 Serial.println(dados.temperatura);
 Serial.println(dados.umidade_solo);

}

void loop() {
}
```

# ESP – NOW - Exemplo 2 mandando por mac



```
#include <esp_now.h>
#include <WiFi.h>

uint8_t macSlaves[][6] = { {0x5C, 0xCF, 0x7F, 0x53, 0xD4, 0x16}, {0x84, 0xF3,
0xEB, 0x58, 0x42, 0x73}};

typedef struct dados_struct {
 float temperatura;
 float umidade_solo;
} dados_struct;

esp_now_peer_info_t peerInfo;

int slavesCount;
```

# ESP – NOW - Exemplo 2 mandando por mac



```
void setup() {

Serial.begin(115200);
WiFi.mode(WIFI_STA);

if (esp_now_init() != ESP_OK) {
 Serial.println("Erro Iniciação do ESP-NOW!!!");
 ESP.restart();
}
else {
 Serial.println("ESP-NOW OK!!!");
}
slavesCount = sizeof(macSlaves)/6/sizeof(uint8_t);

for(int i=0; i<slavesCount; i++){
 peerInfo.channel = 1;
 peerInfo.encrypt = false;
 memcpy(peerInfo.peer_addr, macSlaves[i], sizeof(macSlaves[i]));
 if (esp_now_add_peer(&peerInfo) != ESP_OK){
 Serial.println("Erro na montagem do peer!!!");
 return;
 }
 else{
 Serial.println("Montagem OK!!!");
 }
}
```

# ESP – NOW - Exemplo 2 mandando por mac



```
void send(){

esp_err_t result;
dados_struct dados;
dados.temperatura = random(26,42);
dados.umidade_solo = random(-4,100);
char macStr[18];
for(int i=0; i<slavesCount; i++){
 result = esp_now_send(macSlaves[i],(uint8_t *) &dados, sizeof(dados_struct));
 sprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x", macSlaves[i][0],
 macSlaves[i][1], macSlaves[i][2], macSlaves[i][3], macSlaves[i][4], macSlaves[i][5]);
 Serial.print(macStr);
 if (result == ESP_OK) {
 Serial.println(" Envio Sucesso - ");
 }
 else {
 Serial.println(" Envio Fracasso - ");
 }
}
void loop() {
 delay(1000);
}
```

# ESP – NOW - Exemplo 2 mandando por mac



```
void OnDataSent(const uint8_t *mac_addr, esp_now_send_status_t status) {

 char macStr[18];

 snprintf(macStr, sizeof(macStr), "%02x:%02x:%02x:%02x:%02x:%02x", mac_addr[0],
 mac_addr[1], mac_addr[2], mac_addr[3], mac_addr[4], mac_addr[5]);

 Serial.print("Enviado para: ");
 Serial.print(macStr);
 Serial.print(" - Status: ");
 Serial.println(status == ESP_NOW_SEND_SUCCESS ? "Recebeu!!!" : "Não recebeu!!!");

}
```

**Executar: Exemplo\_ESP32\_NOW\_Mestre\_Para\_N\_Escravos\_01.ino**

# Exercício C1

## NOW



### ■ Com o ESP32(Master):

- Enviará as informações de temperatura em *broadcast*.
- Eu, professor, faço o código e executo.

### ■ Com o ESP8266(Slave):

- Receber a temperatura.
- Caso seja maior do que 32 graus, acionar o relé.
- Vocês farão os *slaves*.

### ■ Componentes utilizados pelo alunos:

ESP8266

relé

Jumpers

# ESP32 - Sensores Nativos

## Efeito Hall



- O elemento Hall é construído com uma fina camada de um material condutivo;
- Quando submetido a um campo magnético reage com uma voltagem proporcional à força deste campo;
- A voltagem gerada é em torno de  $\mu\text{V}$ , exigindo circuitos auxiliares para a sua ampliação;
- Implementado pela função hallRead();
- Inúmeras aplicações:

- Detector de RPM/velocidade;
- Abertura e fechamento de portas;
- Detector de proximidade;
- Equipamentos bancários;
- Sensor de papel;
- Medida de corrente elétrica;
- Motores DC sem escovas;
- Relés;
- Joy stick;
- Sistema de ignição;

# Exemplo – Efeito Hall



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

void loop() {
 Serial.println(hallRead());
 delay(100);
}
```

# ESP32 -Sensores Nativos Toque Capacitivo



- Suporta até 10 pinos sensíveis ao toque/GPIOs;
- Quando o usuário toca a superfície, uma variação de capacitância é disparada;
- Um sinal binário é gerado para indicar se o toque é válido;
- Suporte à operação de baixo consumo:
  - A CPU em *deep-sleep mode* acordará, gradualmente, após o toque;
  - A detecção do toque é administrada pelo ULP co-processor;
- A informação de toque pode ser obtida:
  - Checando os registradores dos sensores de toque por *software*(`touchRead()`);
  - Por uma interrupção disparada pelo módulo detector do toque(`touchAttachInterrupt()`);
  - Acordando a CPU pelo módulo detector de toque, quando em *deep-sleep mode*(`touchAttachInterrupt()`);

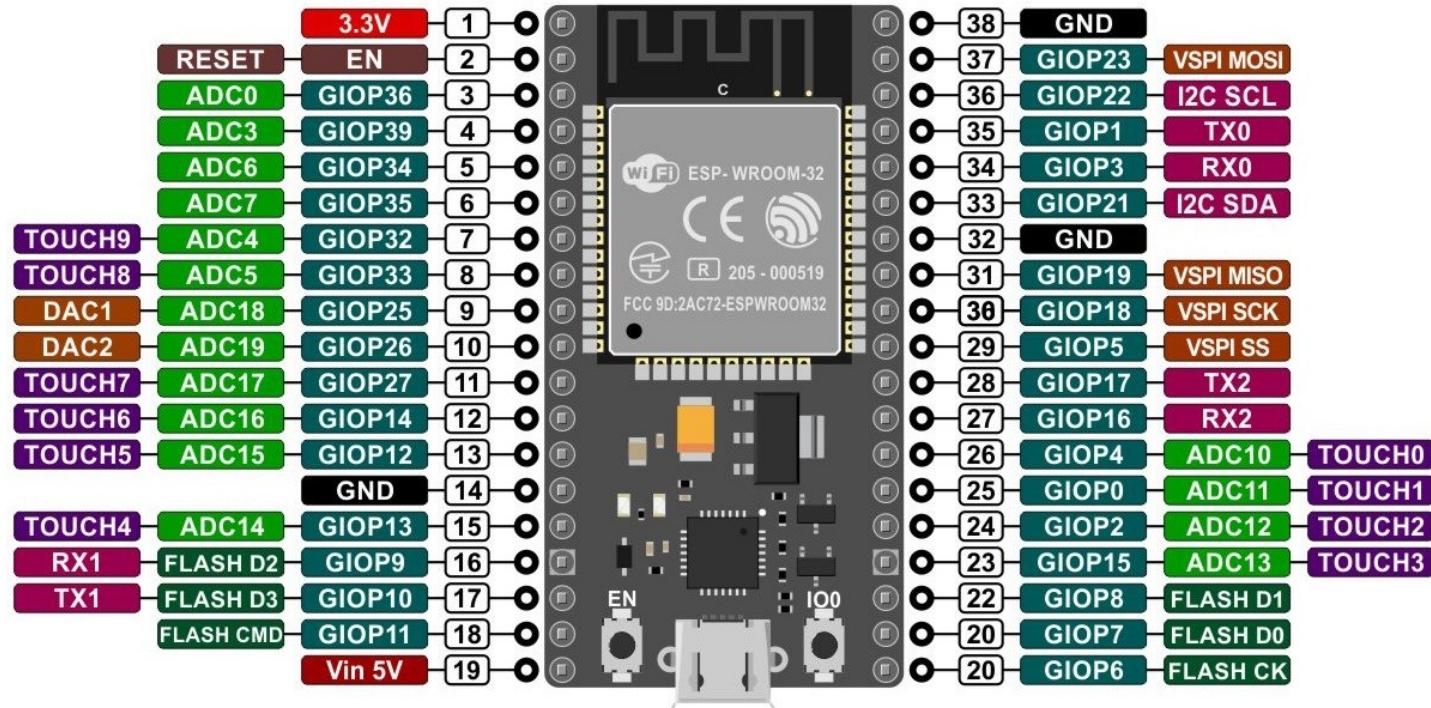
# ESP32 - Sensores Nativos Toque Capacitivo



Função para a leitura:

- int touchRead(pino)
  - pino – identifica o giop ou o canal

A identificação do canal comeca com a letra T. Fx: T4 == GPIO13:



# ESP32 - Exemplo Entrada Toque Capacitivo



- Realizar a leitura, após interrupção, de toques no GIOP12 ou no T3.  
Resultado na saída serial;
- Ligações:

| Jumper | ESP32   |
|--------|---------|
| 1      | GIOP12  |
| 2      | Pino T3 |

- Componentes utilizados;

- 1 Placa ESP32-WROOM;
- Jumpers;

Obs.: touchAttachInterrupt(pino ou canal, nome-da-função-de-retorno, threshold)

**Executar: Exemplo\_ESP32\_Touch\_Interrupt.ino (ver o código!)**

# ESP32 -Sensores Nativos Temperatura Operacional



- Utilizado para verificar a temperatura operacional do ESP32;
- Utiliza sensor inserido na própria placa ESP32;

```
#ifdef __cplusplus
 extern "C" {
#endif
 uint8_t temprature_sens_read();
#ifndef __cplusplus
}
#endif

uint8_t temprature_sens_read();

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

void loop() {
 Serial.print("Temperatura: ");
 Serial.print((temprature_sens_read() - 32) / 1.8); // Converte para graus Celcius
 Serial.println(" C");
 delay(5000);
}
```

# *Deep-Sleep*



- Funcionalidade utilizada para poupar o consumo de energia;
- Aumenta a autonomia do projeto, quando são usadas baterias;
- Viabilizado pela funcionalidade RTC;
- Os módulos de Wi-Fi e *bluetooth* e os processadores principais são expressivos consumidores de energia;
- A ideia é utilizá-los apenas no momento necessário;
- Somente o módulo RTC permanece ativo;

# *Deep-Sleep*



- Quando em estado de *deep-sleep*, os dados em memória são perdidos;
- Apenas os dados no RTC permanecem intactos e podem ser lidos quando os processadores são despertados;
- O ULP *co-processor* realiza medidas das entradas ADC, do sensor de temperatura e dos sensores em I<sup>2</sup>C;

# Deep-Sleep Power Modes



O ESP32 possui os seguintes modos operacionais, além do *deep-sleep*:

*Active mode*

*Modem Sleep mode*

*Light Sleep mode*

*Hibernation mode*

| Power mode                     | Active                    | Modem-sleep | Light-sleep | Deep-sleep                   | Hibernation |
|--------------------------------|---------------------------|-------------|-------------|------------------------------|-------------|
| Sleep pattern                  | Association sleep pattern |             |             | ULP sensor-monitored pattern | -           |
| CPU                            | ON                        | ON          | PAUSE       | OFF                          | OFF         |
| Wi-Fi/BT baseband and radio    | ON                        | OFF         | OFF         | OFF                          | OFF         |
| RTC memory and RTC peripherals | ON                        | ON          | ON          | ON                           | OFF         |
| ULP co-processor               | ON                        | ON          | ON          | ON/OFF                       | OFF         |

# Deep-Sleep Power Consumptions



Table 5: Power Consumption by Power Modes

| Power mode          | Description                                           |                      |                     | Power consumption                        |  |
|---------------------|-------------------------------------------------------|----------------------|---------------------|------------------------------------------|--|
| Active (RF working) | Wi-Fi Tx packet                                       |                      |                     | Please refer to<br>Table 14 for details. |  |
|                     | Wi-Fi/BT Tx packet                                    |                      |                     |                                          |  |
|                     | Wi-Fi/BT Rx and listening                             |                      |                     |                                          |  |
| Modem-sleep         | The CPU is powered on.                                | 240 MHz*             | Dual-core chip(s)   | 30 mA ~ 68 mA                            |  |
|                     |                                                       |                      | Single-core chip(s) | N/A                                      |  |
|                     |                                                       | 160 MHz*             | Dual-core chip(s)   | 27 mA ~ 44 mA                            |  |
|                     |                                                       |                      | Single-core chip(s) | 27 mA ~ 34 mA                            |  |
|                     |                                                       | Normal speed: 80 MHz | Dual-core chip(s)   | 20 mA ~ 31 mA                            |  |
|                     |                                                       |                      | Single-core chip(s) | 20 mA ~ 25 mA                            |  |
| Light-sleep         | -                                                     |                      |                     | 0.8 mA                                   |  |
| Deep-sleep          | The ULP co-processor is powered on.                   |                      |                     | 150 $\mu$ A                              |  |
|                     | ULP sensor-monitored pattern                          |                      |                     | 100 $\mu$ A @1% duty                     |  |
|                     | RTC timer + RTC memory                                |                      |                     | 10 $\mu$ A                               |  |
| Hibernation         | RTC timer only                                        |                      |                     | 5 $\mu$ A                                |  |
| Power off           | CHIP_PU is set to low level, the chip is powered off. |                      |                     | 0.1 $\mu$ A                              |  |

[ESP32 Espressif datasheet](#)

Table 14: RF Power-Consumption Specifications

| Mode                                            | Min | Typ      | Max | Unit |
|-------------------------------------------------|-----|----------|-----|------|
| Transmit 802.11b, DSSS 1 Mbps, POUT = +19.5 dBm | -   | 240      | -   | mA   |
| Transmit 802.11b, OFDM 54 Mbps, POUT = +16 dBm  | -   | 190      | -   | mA   |
| Transmit 802.11g, OFDM MCS7, POUT = +14 dBm     | -   | 180      | -   | mA   |
| Receive 802.11b/g/n                             | -   | 95 ~ 100 | -   | mA   |
| Transmit BT/BLE, POUT = 0 dBm                   | -   | 130      | -   | mA   |
| Receive BT/BLE                                  | -   | 95 ~ 100 | -   | mA   |

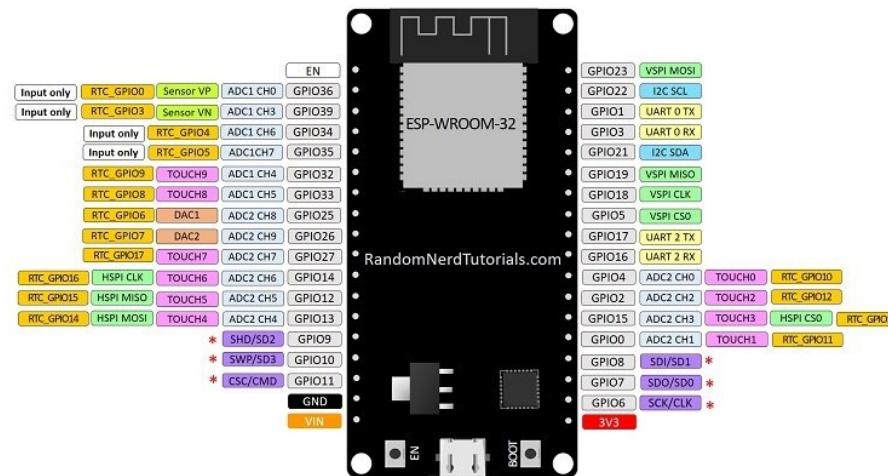
# *Deep-Sleep*



- O módulo pode acordar por tempo, pelo toque capacitivo ou por um evento externo(GPIO);
  - Antes de entrar em *deep-sleep mode* a aplicação precisa desabilitar o Wi-Fi e o Bluetooth, caso ativos;
  - Além dos GPIOs de toque, apenas os GPIOs com a identificação RTC podem ser utilizados;

# ESP32 DEVKIT V1 – DOIT

version with 36 GPIOs



\* Pins SCK/CLK, SDO/SD0, SDI/SD1, SHD/SD2, SWP/SD3 and SCS/CMD, namely, GPIO6 to GPIO11 are connected to the integrated SPI flash integrated on ESP-WROOM-32 and are not recommended for other uses.

# Deep-Sleep



- Quando utilizar o toque capacitivo, é necessário utilizar a sua função de interrupção, `touchAttachInterrupt()`;
- Passos para utilizar o *deep-sleep*;
  - Habilitar a fonte que irá acordar o módulo;
  - Decidir quais periféricos ficarão habilitados ou desabilitados;

Obs.: O ESP32 desabilita, por padrão, os periféricos não necessários para acordar a CPU;

# *Deep-Sleep* Funções para o uso



## Habilitar a fonte tempo:

- **esp\_err\_t esp\_sleep\_enable\_timer\_wakeup(uint64\_t time\_in\_us)**
    - uint64\_t time\_in\_us – tempo, em µs, para acordar
    - Retorna ESP\_OK ou ESP\_ERR\_INVALID\_ARG

## Habilitar a fonte toque capacitivo:

- `esp_err_t esp_sleep_enable_touchpad_wakeup()`
    - Retorna ESP\_OK ou ESP\_ERR\_INVALID\_STATE

### Verificar qual pino foi tocado:

- touch\_pad\_t **esp\_sleep\_get\_touchpad\_wakeup\_status()**
    - Retorna o pino que foi tocado

💡 Habilitar a fonte GPIO do tipo RTC, usando somente um pino:

- **esp\_err\_t esp\_sleep\_enable\_ext0\_wakeup(gpio\_num\_t gpio\_num, tgpio\_num, int level)**
    - gpio\_num - número do GPIO utilizado como fonte. GPIOs válidos são 0,2,4,12-15,25-27,32-39(pinos com prefixo RTC)
    - level - nível do sinal de entrada que acordará a CPU (0=low, 1=high)
    - Retorna ESP\_OK, ESP\_ERR\_INVALID\_ARG ou ESP\_ERR\_INVALID\_STATE

# Deep-Sleep

## Funções para o uso



■ Habilitar a fonte GPIO do tipo RTC, usando vários pinos ao mesmo tempo

- **esp\_err\_t esp\_sleep\_enable\_ext1\_wakeup(uint64\_t mask, esp\_sleep\_ext1\_wakeup\_mode\_tmode)**
  - mask - bit mask com os pinos GPIO que acordarão a CPU. GPIOs válidos são 0,2,4,12-15,25-27,32-39(pinios com prefixo RTC)
  - esp\_sleep\_ext1\_wakeup\_mode\_tmode - seleciona a função lógica usada para determinar a condição de acordar:
    - ESP\_EXT1\_WAKEUP\_ALL\_LOW: acorda quando todos GPIOs estão LOW
    - ESP\_EXT1\_WAKEUP\_ANY\_HIGH: acorda quando qualquer dos GPIOs está HIGH
    - Retorna ESP\_OK ou ESP\_ERR\_INVALID\_ARG

■ Desabilitar determinada fonte:

- **esp\_err\_t esp\_sleep\_disable\_wakeup\_source(esp\_sleep\_source\_tsource)**
  - esp\_sleep\_source\_tsource - número da fonte a ser desabilitada do tipo type esp\_sleep\_source\_t
  - Retorna ESP\_OK ou ESP\_ERR\_INVALID\_STATE

■ Colocar a CPU para dormir:

- **void esp\_deep\_sleep\_start()**

■ Descobrir quem acordou a CPU:

- **esp\_sleep\_wakeup\_cause\_t esp\_sleep\_get\_wakeup\_cause()**

# *Deep-Sleep* quem acordou?



enum esp\_sleep\_source\_t – quem acordou

Valores possíveis:

ESP\_SLEEP\_WAKEUP\_UNDEFINED

In case of deep sleep, reset was not caused by exit from deep sleep.

ESP\_SLEEP\_WAKEUP\_ALL

Not a wakeup cause, used to disable all wakeup sources with esp\_sleep\_disable\_wakeup\_source.

ESP\_SLEEP\_WAKEUP\_EXT0

Wakeup caused by external signal using RTC\_IO.

ESP\_SLEEP\_WAKEUP\_EXT1

Wakeup caused by external signal using RTC\_CNTL.

ESP\_SLEEP\_WAKEUP\_TIMER

Wakeup caused by timer.

ESP\_SLEEP\_WAKEUP\_TOUCHPAD

Wakeup caused by touchpad.

ESP\_SLEEP\_WAKEUP\_ULP

Wakeup caused by ULP program.

ESP\_SLEEP\_WAKEUP\_GPIO

Wakeup caused by GPIO (light sleep only)

# *Deep-Sleep* Dados na memória RTC



- Quando a CPU acorda o *sketch* é reiniciado;
- Dados armazenados nas memórias regulares são perdidos;
- Para preservar os dados, eles devem ser armazenados na memória do módulo RTC;
- O dado a ser salvo é do tipo RTC\_DATA\_ATTR;

# Deep-Sleep Exemplo - Timer



```
/* Simple Deep Sleep with Timer Wake Up
```

```
=====
```

ESP32 offers a deep sleep mode for effective power saving as power is an important factor for IoT applications. In this mode CPUs, most of the RAM, and all the digital peripherals which are clocked from APB\_CLK are powered off. The only parts of the chip which can still be powered on are: RTC controller, RTC peripherals ,and RTC memories. This code displays the most basic deep sleep with

a timer to wake it up and how to store data in RTC memory to use it over reboots. This code is under Public Domain License.

**Author:** Pranav Cherukupalli <cherukupallip@gmail.com>\*/

```
#define uS_TO_S_FACTOR 1000000 /* Conversion factor for micro seconds to seconds */
#define TIME_TO_SLEEP 5 /* Time ESP32 will go to sleep (in seconds) */
```

```
RTC_DATA_ATTR int bootCount = 0;
```

```
/* Method to print the reason by which ESP32 has been awaken from sleep */
```

```
void print_wakeup_reason(){
 esp_sleep_wakeup_cause_t wakeup_reason;

 wakeup_reason = esp_sleep_get_wakeup_cause();

 switch(wakeup_reason)
 {
 case 1 : Serial.println("Wakeup caused by external signal using RTC_IO"); break;
 case 2 : Serial.println("Wakeup caused by external signal using RTC_CNTL"); break;
 case 3 : Serial.println("Wakeup caused by timer"); break;
 case 4 : Serial.println("Wakeup caused by touchpad"); break;
 case 5 : Serial.println("Wakeup caused by ULP program"); break;
 default : Serial.println("Wakeup was not caused by deep sleep"); break;
 }
}
```

# *Deep-Sleep Exemplo - Timer*



```
void setup(){
 Serial.begin(115200);
 delay(1000); //Take some time to open up the Serial Monitor

 //Increment boot number and print it every reboot
 ++bootCount;
 Serial.println("Boot number: " + String(bootCount));

 //Print the wakeup reason for ESP32
 print_wakeup_reason();

/* First we configure the wake up source. We set our ESP32 to wake up every 5 seconds. */
esp_sleep_enable_timer_wakeup(TIME_TO_SLEEP * uS_TO_S_FACTOR);
Serial.println("Setup ESP32 to sleep for every " + String(TIME_TO_SLEEP) + " Seconds");

Serial.println("Going to sleep now");
esp_deep_sleep_start();
Serial.println("This will never be printed");

void loop(){}
}
```

# Deep-Sleep Exemplo - Toque



```
/*
 * Deep Sleep with Touch Wake Up
 =====
```

This code displays how to use deep sleep with a touch as a wake up source and how to store data in RTC memory to use it over reboots. This code is under Public Domain License. **Author:Pranav Cherukupalli <cherukupallip@gmail.com>**\*/

```
#define Threshold 40 /* Greater the value, more the sensitivity
RTC_DATA_ATTR int bootCount = 0;
touch_pad_t touchPin;

/* Method to print the reason by which ESP32 has been awoken from sleep */
void print_wakeup_reason(){
 esp_sleep_wakeup_cause_t wakeup_reason;

 wakeup_reason = esp_sleep_get_wakeup_cause();

 switch(wakeup_reason)
 {
 case 1 : Serial.println("Wakeup caused by external signal using RTC_IO"); break;
 case 2 : Serial.println("Wakeup caused by external signal using RTC_CNTL"); break;
 case 3 : Serial.println("Wakeup caused by timer"); break;
 case 4 : Serial.println("Wakeup caused by touchpad"); break;
 case 5 : Serial.println("Wakeup caused by ULP program"); break;
 default : Serial.println("Wakeup was not caused by deep sleep"); break;
 }
}
```

# Deep-Sleep Exemplo - Toque



```
/*Method to print the touchpad by which ESP32 has been awaken from sleep */

void print_wakeup_touchpad(){
 touch_pad_t touchPin;

 touchPin = esp_sleep_get_touchpad_wakeup_status();

 switch(touchPin) {
 case 0 : Serial.println("Touch detected on GPIO 4"); break;
 case 1 : Serial.println("Touch detected on GPIO 0"); break;
 case 2 : Serial.println("Touch detected on GPIO 2"); break;
 case 3 : Serial.println("Touch detected on GPIO 15"); break;
 case 4 : Serial.println("Touch detected on GPIO 13"); break;
 case 5 : Serial.println("Touch detected on GPIO 12"); break;
 case 6 : Serial.println("Touch detected on GPIO 14"); break;
 case 7 : Serial.println("Touch detected on GPIO 27"); break;
 case 8 : Serial.println("Touch detected on GPIO 33"); break;
 case 9 : Serial.println("Touch detected on GPIO 32"); break;
 default : Serial.println("Wakeup not by touchpad"); break;
 }
}

void callback(){
 //aqui, tratará o que fazer após o toque
}
```

# *Deep-Sleep* Exemplo - Toque



```
void setup(){
 Serial.begin(115200);
 delay(1000); //Take some time to open up the Serial Monitor

 //Increment boot number and print it every reboot
 ++bootCount;
 Serial.println("Boot number: " + String(bootCount));

 //Print the wakeup reason for ESP32 and touchpad too
 print_wakeup_reason();
 print_wakeup_touchpad();

 //Setup interrupt on Touch Pad 3 (GPIO15)
 touchAttachInterrupt(T3, callback, Threshold);

 //Configure Touchpad as wakeup source
 esp_sleep_enable_touchpad_wakeup();

 //Go to sleep now
 Serial.println("Going to sleep now");
 esp_deep_sleep_start();
 Serial.println("This will never be printed");
}

void loop(){}
```

# Exercício C2

## NOW + Toque capacitivo



■ Com o ESP32(Master):

- Caso o pino T3 seja tocado, acordar o ESP32 e informar a um slave específico que ocorreu o toque.

■ Com o ESP8266(Slave):

- Receber a informação de que o pino foi tocado e acender um led.

■ Componentes utilizados pelo alunos:

ESP8266

ESP32

led

Jumpers

# *Deep-Sleep* Exemplo - Externa 1 pino



```
void setup(){
 Serial.begin(115200);
 delay(1000); //Take some time to open up the Serial Monitor

 //Increment boot number and print it every reboot
 ++bootCount;
 Serial.println("Boot number: " + String(bootCount));

 //Print the wakeup reason for ESP32
 print_wakeup_reason();

 esp_sleep_enable_ext0_wakeup(GPIO_NUM_33,1); // 1 = High, 0 = Low

 //Go to sleep now
 Serial.println("Going to sleep now");
 esp_deep_sleep_start();
 Serial.println("This will never be printed");
}

void loop(){}
```

# *Deep-Sleep*

## Exemplo – Externa + pinos



■ **`esp_sleep_enable_ext1_wakeup(uint64_t mask, esp_sleep_ext1_wakeup_mode_tmode)`**

■ Cálculo do bitmask:

- Bitmask =  $2^{(\text{rtcpin1})} + 2^{(\text{rtcpin2})} + \dots + 2^{(\text{rtcpin}n)}$ ;
- Converter resultado para valor em hexadecimal;
- Colocar 0x como prefixo do valor obtido;

Obs.: Conversão em <https://www.rapidtables.com/convert/number/decimal-to-hex.html>

# Deep-Sleep Exemplo - Externa + pinos



```
/* Deep Sleep with External Wake Up
```

```
=====
```

This code displays how to use deep sleep with an external trigger as a wake up source and how to store data in RTC memory to use it over reboots. This code is under Public Domain License.

Hardware Connections: Push Button to GPIO 33 pulled down with a 10K Ohm resistor

NOTE: Only IO can be used as a source for external wake source. They are pins: 0,2,4,12-15,25-27,32-39(**RTC pins**).

Author: **Pranav Cherukupalli <cherukupallip@gmail.com>** \*/

```
#define BUTTON_PIN_BITMASK 0x8004 // GPIOs 2 and 15
RTC_DATA_ATTR int bootCount = 0;
```

```
/* Method to print the reason by which ESP32 has been awoken from sleep */
```

```
void print_wakeup_reason(){
 esp_sleep_wakeup_cause_t wakeup_reason;

 wakeup_reason = esp_sleep_get_wakeup_cause();

 switch(wakeup_reason){
 case 1 : Serial.println("Wakeup caused by external signal using RTC_IO"); break;
 case 2 : Serial.println("Wakeup caused by external signal using RTC_CNTL"); break;
 case 3 : Serial.println("Wakeup caused by timer"); break;
 case 4 : Serial.println("Wakeup caused by touchpad"); break;
 case 5 : Serial.println("Wakeup caused by ULP program"); break;
 default : Serial.println("Wakeup was not caused by deep sleep"); break;
 }
}
```

# Deep-Sleep Exemplo - Externa + pinos



```
void setup(){
 Serial.begin(115200);
 delay(1000); //Take some time to open up the Serial Monitor

 //Increment boot number and print it every reboot
 ++bootCount;
 Serial.println("Boot number: " + String(bootCount));

 //Print the wakeup reason for ESP32
 print_wakeup_reason();

esp_sleep_enable_ext1_wakeup(BUTTON_PIN_BITMASK, ESP_EXT1_WAKEUP_ANY_HIGH);

 //Go to sleep now
 Serial.println("Going to sleep now");
 esp_deep_sleep_start();
 Serial.println("This will never be printed");
}

void loop(){
}
```

# Exercício C3 - ESP32 NOW -Deep Sleep – 1 pino



## ■ Com o ESP32(Master):

- Fica dormindo até que seja detectada uma presença humana.
- Quando detectada, acorda e informa uma máquina *slave*.

## ■ Com o ESP8266(Slave):

- Recebe a informação de que alguém entrou e aciona um relé.

## ■ Componentes utilizados pelo alunos:

ESP8266

ESP32

Sensor PIR

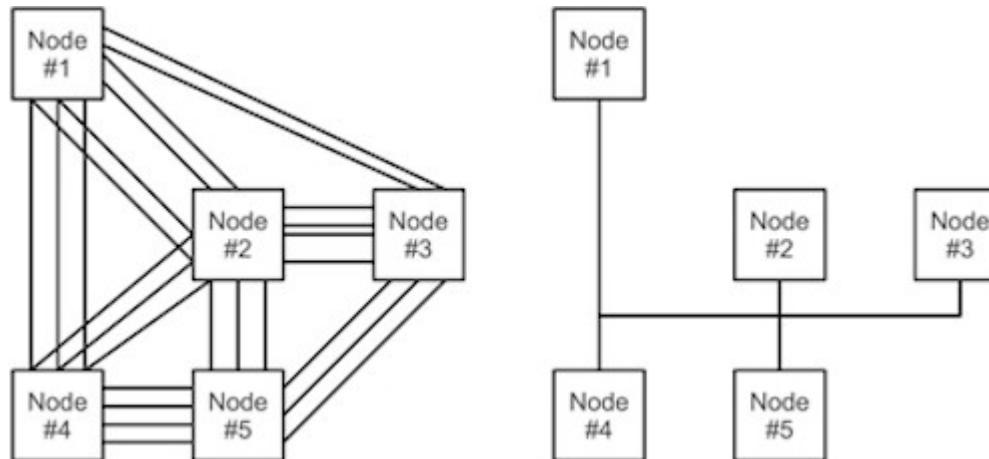
relé

Jumpers

# ESP32 - CAN Controller Area Network



- Protocolo de comunicação serial desenvolvido por Robert Bosch no início da década de 1980;
- Direcionado para a indústria automobilística;
- Objetivo de simplificar as complicadas ligações de fios entre os microcontroladores diversos;



# ESP32 - CAN Controller Area Network

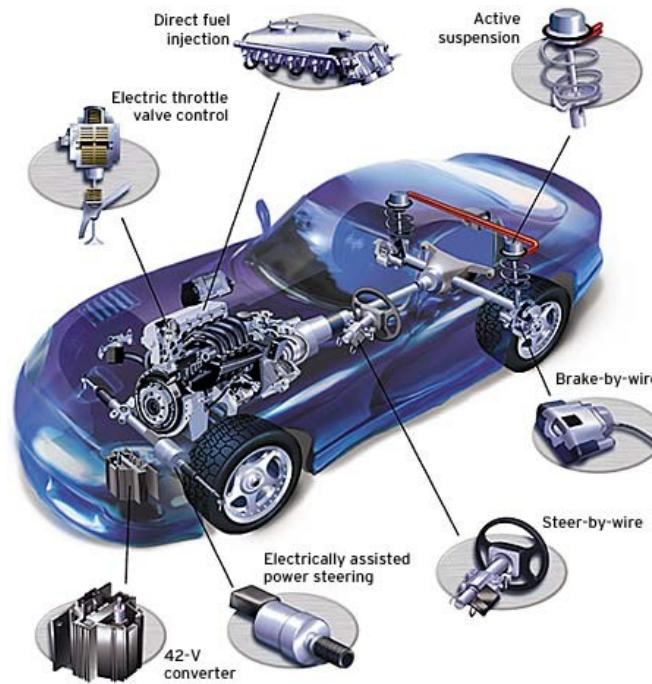


- Comunicação *half-duplex* realizada por um par de fios;
- Cada nó é composto de um MCU e um CAN *transceiver*;
- Nós distintos podem, ao mesmo tempo, pedir acesso à rede(sistema multi-mestre);
- Suporta, também, transmissão em *multicast*;
- Velocidade máxima de transmissão é de 1Mbps para distâncias de até 40m;
- Com velocidade de 50Kbps pode chegar até 1Km;
- Na rede, não existe endereço do destinatário;
- Uma mensagem é enviada a todos os nós;
- Cada nó decide o que fazer;
- Cada mensagem enviada pode conter até 8bytes de tamanho;
- A quantidade de nós depende do fabricante do módulo CAN, chegando até 120 nós;

# ESP32 - CAN Controller Area Network



- Expressiva imunidade à ruídos de transmissão;
- Sinalização de erros;
- Retransmissão automática de mensagens;
- Latência mínima;



# ESP32 - CAN Controller Area Network



- Biblioteca de apoio em <https://github.com/sandeepmistry/arduino-CAN>;
- Regulado pela ISO 11989 foi adotado por outras indústrias, como a robótica;
- Baixo custo, cada circuito CAN custa em torno de US\$1.00;

# **ESP32 – Dual Core multi-threads**



- Permite a execução de trechos do código em *cores* diferentes;
- Viabiliza a funcionalidade de multitarefas;
- Faz parte das funcionalidades padrões do ESP32, através do freeRTOS;
- Torna o monitoramento por vários sensores mais eficiente;

# ESP32 – *Dual Core multi-threads*



## ■ Passos para utilizar:

- Definir a *thread* como uma variável do tipo TaskHandle\_t;
- Definir os parâmetros da *thread* na função xTaskCreatePinnedToCore. São eles:
  - *Thread* a ser invocada;
  - Nome descritivo da *thread*;
  - *Stack size*;
  - Parâmetro de entrada para a *thread*;
  - *Handle* para qual a *thread* pode fazer referência;
  - Prioridade da *thread*;
  - *Core* onde a *thread* será executada;

# ESP32 – *Dual Core* Exemplo



```
TaskHandle_t Thread_1;
TaskHandle_t Thread_2;

const int redled = 12;
const int greenled = 13;

void setup() {
 Serial.begin(115200);
 pinMode(redled, OUTPUT);
 pinMode(greenled , OUTPUT);

xTaskCreatePinnedToCore(Thread_1Codigo,"Thread_1",10000,NULL,1,&Thread_1,0);
 delay(500);
xTaskCreatePinnedToCore(Thread_2Codigo,"Thread_2",10000,NULL,1,&Thread_2,1);
 delay(500);
}

void Thread_1Codigo(void * pvParameters){
 Serial.print("Core ");
 Serial.println(xPortGetCoreID());
 for(;;){
 digitalWrite(redled, HIGH);
 delay(1000);
 digitalWrite(redled, LOW);
 delay(1000);
 }
}
```

# ESP32 – *Dual Core* Exemplo



```
void Thread_2Codigo(void * pvParameters){

 Serial.print("Core ");
 Serial.println(xPortGetCoreID());

 for(;;){
 digitalWrite(greenled, HIGH);
 delay(700);
 digitalWrite(greenled, LOW);
 delay(700);
 }
}

void loop() {
}
```

**Executar: Exemplo\_ESP32\_Multi\_Threads.ino, colocar os leds.**

# Exemplo - Mosquitto

## Publicando com *multithreads*

■ Publicando no Mosquitto, um tópico, usando *multithreads* :

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

const char* ssid = "ssssssssssssssss"; // WiFi SSID
const char* password = "pppppppppppppppp"; // WiFi password
const char* mqtt_server = "test.mosquitto.org"; // Endereço do Mosquitto - Broker
char msg[50];
char topico[50];

TaskHandle_t Thread_1;
TaskHandle_t Thread_2;

WiFiClient espClient;
PubSubClient client(espClient);

void setup(){
 Serial.begin(115200);
 Serial.print("Conectando a Rede WiFi ");
 WiFi.begin(ssid, password);
 while (WiFi.status() != WL_CONNECTED) {
 Serial.print(".");
 delay(500);
 }
 Serial.println("Conectado ");
 client.setServer(mqtt_server, 1883); // Atribui endereço do Broker Mosquitto e a porta
 xTaskCreatePinnedToCore(Thread_Sensor_01,"Thread_Sensor_01",10000,NULL,1,&Thread_1,0);
 delay(500);
 xTaskCreatePinnedToCore(Thread_Sensor_02,"Thread_Sensor_02",10000,NULL,1,&Thread_2,1);
```

# Exemplo - Mosquitto

## Publicando com *multithreads*



■ Publicando no Mosquitto, um tópico, usando *multithreads*:

```
void publica(char* msg, String local){
 if (!client.connected()){
 conecta();
 }
 local.toCharArray(topico,local.length()+1);
 client.publish(topico,msg,true);
 Serial.println("Mensagem publicada");
}

void conecta(){
 while (!client.connected()) {
 if (client.connect("ESP32_Deep_Sleep","","","", "Controle_Ambiente_01",1,true,"Fora do Ar")){
 Serial.println("Cliente Conectado ao Mosquitto");
 }
 else{
 Serial.println(client.state());
 delay(2000);
 }
 }
}
```

# Exemplo - Mosquitto Publicando com *multithreads*



- Publicando no Mosquitto, um tópico, usando *multithreads*:

```
void Thread_Sensor_01(void * pvParameters){

 while(true){

 String temperatura =(String) random(10,45); // Valor arbitrado randomicamente(É um exemplo!)
 temperatura.toCharArray(msg,temperatura.length()+1);
 publica(msg,"Temperatura");
 delay(5000);

 }
}

void Thread_Sensor_02(void * pvParameters){

 while(true){

 String umidade =(String) random(0,100); // Valor arbitrado randomicamente(É um exemplo!)
 umidade.toCharArray(msg,umidade.length()+1);
 publica(msg,"Umidade");
 delay(5000);

 }
}

void loop(){}

```

# OTA(*Over the Air*)



- Permite o *upload* de *sketchs* e arquivos pelo ar;
- Facilidade suprida pela biblioteca AsyncElegantOTA.h;
- Código do OTA fica incorporado ao *sketch* definitivamente;
- *Upload* pode ser protegido por senha;
- Passos para serem utilizados pela IDE do Arduino:
  - Abrir IDE do Arduino
  - Clicar em Sketch → Incluir Biblioteca → Gerenciar Bibliotecas
  - Procurar a entrada AsyncElegantOTA
  - Clicar sobre esta entrada
  - Selecionar a versão mais recente e instalar

# OTA(*Over the Air*) Instruções para o uso



- Incluir no código alvo:

```
#include <ESPAAsyncWebServer.h>
#include <AsyncElegantOTA.h>
```

```
AsyncWebServer server(80);
AsyncElegantOTA.begin(&server);
server.begin();
```

- Digitar na URL do navegador:

endereço-ip-obtido/[update](#)

- Após a digitação, aparecerá a seguinte tela:

# OTA(*Over the Air*) Instruções para o uso



Trabalhando com J...



<https://www.ib7.bra...>



Ro-Online Web



How to use LVGL li...



Java Tutorials Learn...



ElegantOTA

Firmware

Filesystem

[Escolher arquivo](#)

Nenhum arquivo escolhido

80E350CC

-  
ESP32

# OTA(*Over the Air*) Instruções para o uso



## ■ Para fazer *upload* de um código(*firmware*):

- Na IDE do Arduino:
  - Selecione Sketch → Exportar binário compilado.
  - O binário compilado ficará na mesma pasta do código da aplicação.
- No navegador, selecione *Firmware* e escolher arquivo:
  - Escolher o arquivo binário gerado.
- Aguardar o *upload*.

**Executar Exemplo\_ESP8266\_WiFi\_Mosquitto\_Publicando\_OTA\_1\_Topico.ino**

**Exemplo\_ESP32\_WiFi\_Mosquitto\_Assinando\_OTA\_1\_Topico.ino**

## ■ Para fazer *upload* de um arquivo do FileSystem:

- Atualizar o conteúdo do arquivo.
- Para descobrir o local do **spiffs.bin**:
  - Fazer a operação de ESP32 File Upload ou ESP8266 File Upload, **sem conectar o MCU**.
  - Procurar, na janela de debugging, a entrada **[SPIFFS] upload**.
    - Ver o *path* do arquivo de sufixo **.spiffs.bin**.
- No navegador, selecione *Filesystem* e escolher arquivo:
  - Escolher o arquivo de sufixo **.spiffs.bin**. (No Explorer do Windows, exibir itens ocultos e extensões)
- Aguardar o *upload*.

**Executar ESP32\_AsyncWebServer\_SPIFFS\_Ativo\_HTML\_CSS\_OTA.ino**

# OTA(*Over the Air*) Para executar



## ■ Executar para teste do *firmware*:

- **Exemplo\_ESP8266\_WiFi\_Mosquitto\_Publicando\_OTA\_1\_Topico.ino**
- **Exemplo\_ESP32\_WiFi\_Mosquitto\_Assinando\_OTA\_1\_Topico.ino**
- **Exemplo\_ESP32\_NOW\_OTA\_Mestre\_Broadcast\_01.ino**
- Executem cada um dos exemplos, vejam o resultado, desconectem o MCU do computador base e conectem em alguma outra fonte, alterem o código original e façam o *upload*. Importante anotar o endereço IP.

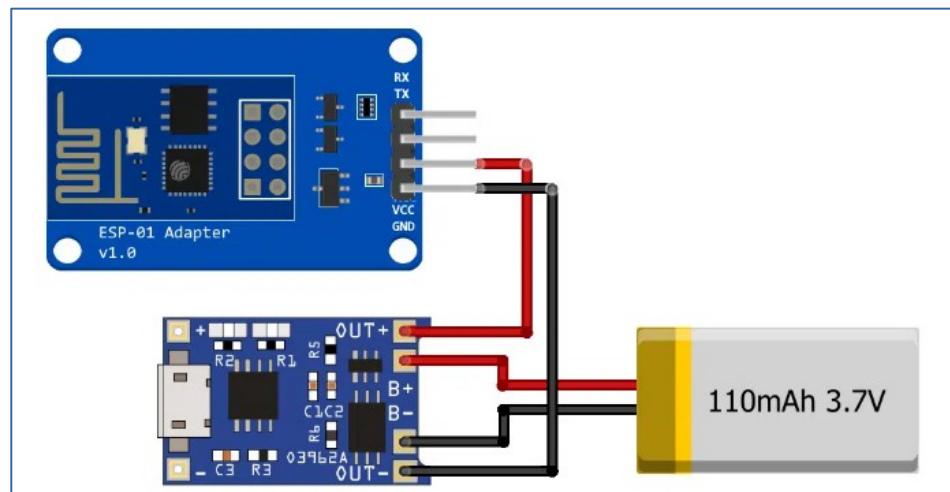
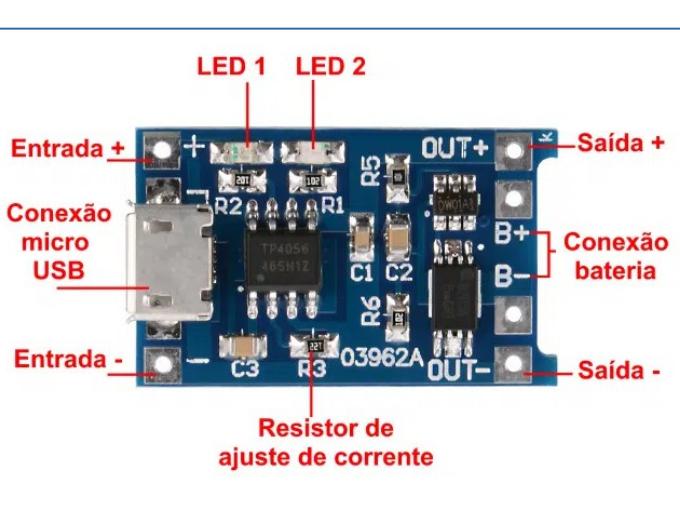
## ■ Executar para teste do FileSystem:

- Executar **ESP32\_AsyncWebServer\_SPIFFS\_Ativo\_HTML\_CSS\_OTA.ino**
- Observações idênticas.

# TP4056 Carregador bateria lítio



- Fundamental para dar autonomia energética ao projeto:



# Alexa



## ■ Passos para serem utilizados pela IDE do Arduino:

- Abrir IDE do Arduino.
- Clicar em Sketch → Incluir Biblioteca → Gerenciar Bibliotecas.
- Procurar a entrada **espalexa**.
- Clicar sobre esta entrada.
- Selecionar a versão mais recente e instalar.

# Alexa Biblioteca Espalexa



- Permite que se defina um valor variado, por exemplo, para brilho, temperatura e cor, além do controle liga/desliga;
- Permite dizer "Alexa, diminua a luz para 75%";
- Suporta cores com a API atual;
- Permite dizer "Alexa, torne a luz para azul";
- A temperatura da cor (tons brancos) também é suportada;
- Por padrão, é possível adicionar até um total de 10 dispositivos;
- Cada dispositivo tem uma faixa de brilho de 0 a 255, onde 0 está desligado e 255 está totalmente ligado;

# Alexa Funções



## ■ Algumas das funções:

- #include <Espalexa.h>
- espalexa.addDevice(nome-do-dispositivo,função-callback[,valor]);
  - nome-do-dispositivo – dispositivo que ouvirá a Alexa.
  - função-callback – nome da função que atenderá a solicitação da Alexa.
  - valor – valor inicial (0-255).
- espalexa.begin();
  - Para iniciar o serviço
- espalexa.loop();
  - Para o MCU ver se há ordens da Alexa.
- espalexa.toPercent(brightness);
  - Brightness - valor da intensidade do brilho.

# Alexa Declaração Alternativa



■ Outra forma :

```
EspalexaDevice* devicen;
```

```
devicen = new EspalexaDevice("Cafeteira",trataCafeteira);
```

```
espalexa.addDevice(devicen);
```

```
devicen->setValue(128);
```

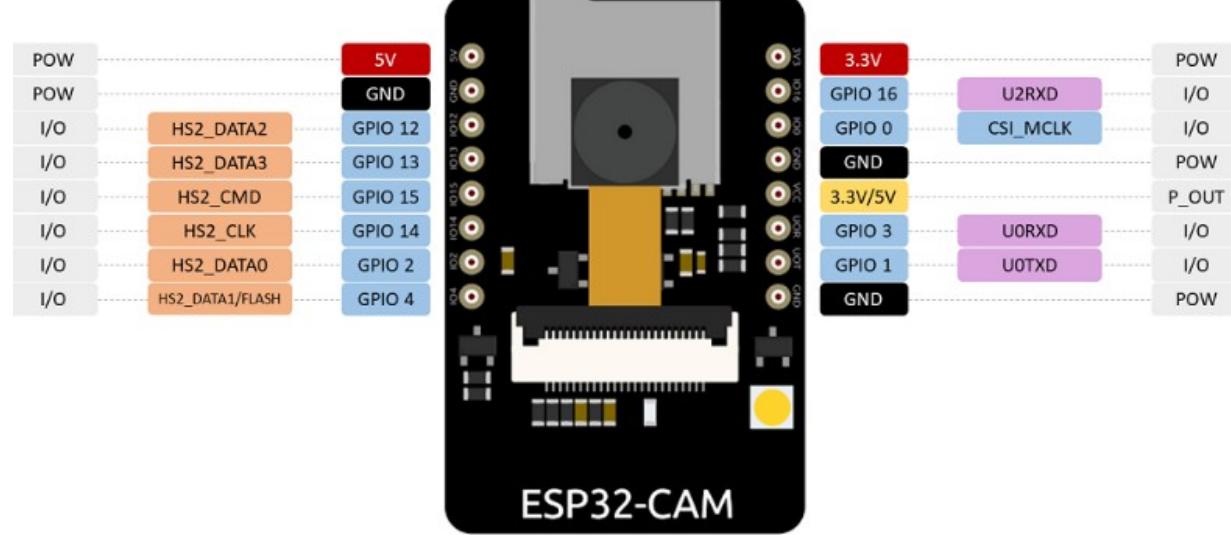
# ESP32-CAM



- ESP32-CAM une ESP32 com uma câmera OV2640 de 2MP;
- Possibilidade de filmar(*video streaming*) e fotografar;
- Também é possível armazenar as imagens no cartão MicroSD;
- Possui um LED de flash embutido na placa, para melhorar as imagens tiradas;
- Especificações Técnicas:

- Processador: Xtensa® Dual-Core 32-bit LX6
- Memória Flash programável: 4 MB (dos quais 0,9 Mb são usados pelo bootloader)
- Memória RAM: 520 Kbytes + **External 4M PSRAM**
- Memória ROM: 448 Kbytes
- Clock máximo: 240 MHz
- Pinos Digitais GPIO: 11 (todos com PWM)
- Resolução do PWM: até 16 bits (ajustável via código)
- Wireless 802.11 b/g/n - 2.4GHz (antena integrada)
- Modos de operação: Access Point / Estação / Access Point + Estação
- Bluetooth Low Energy padrão 4.2 integrado

# ESP32-CAM



ESP32-CAM AI-Thinker

# ESP32-CAM



## ■ Executar Exemplo:

- Na IDE do Arduino, selecionar a placa AI-Thinker ESP32 CAM.
- Em seguida, Arquivo → Exemplos → Esp32 → Camera → CameraWebServer.

## ■ Em seguida, executar :

Exemplo\_ESP32CAM\_Video\_Streaming\_01.ino.

Após executar, anotar o endereço IP obtido, desconectar a câmera, alimentá-la pelo *power bank*, andar com ela pelo ambiente, observar o resultado pelo navegador.

# ESP32/ESP8266 Email



- Biblioteca padrão, Esp-Mail-Client, e documentação para ESP32/ESP8266, em <https://github.com/mobitz/ESP-Mail-Client>;
- Baixar do Google Drive;
- Bons exemplos na IDE do Arduino;
- Instruções para uso(zoho mail):

```
#include "ESP32_MailClient.h"

#define emailSenderAccount "email-do-remetente"
#define emailSenderPassword "senha"
#define smtpServer "smtp.zoho.com" // Servidor de e-mail, ver o seu
#define smtpServerPort 465
#define emailSubject "Texto livre"
#define emailRecipient "email-do-remetente"

SMTPData smtpData;
```

# ESP32/ESP8266 Email



```
smtpData.setLogin(smtpServer, smtpServerPort, emailSenderAccount, emailSenderPassword);
smtpData.setSender("ESP32-CAM", emailSenderAccount);

smtpData.setPriority("High");

smtpData.setSubject(emailSubject);

smtpData.setMessage("<h2>Foto capturada e anexada a este Email.</h2>", true);

smtpData.addRecipient(emailRecipient);

// Add attach files from SPIFFS
smtpData.addAttachFile(FILE_PHOTO, "image/jpg"); // Anexa imagem recém capturada
// Set the storage type to attach files in your email (SPIFFS)
smtpData.setFileStorageType(MailClientStorageType::SPIFFS);

smtpData.setSendCallback(sendCallback);

MailClient.sendMail(smtpData)
```

# ESP32/ESP8266 Telegram



- Biblioteca padrão, UniversalTelegramBot, para ESP32/ESP8266 em <https://github.com/witnessmenow/Universal-Arduino-Telegram-Bot>;
- Documentação neste mesmo endereço;
- Baixar do Google Drive;
- O Telegram Messenger é um serviço de mensagens instantâneas e de voz, sobre IP, baseado em nuvem;
- O Telegram permite que a criação de *bots* com os quais pode-se interagir;
- *Bots* são aplicativos de terceiros que rodam dentro do Telegram. Os usuários podem interagir com *bots* enviando mensagens, comandos e solicitações. Os *bots* são controlados usando solicitações HTTPS para Telegram Bot API.

# ESP32/ESP8266

## Telegram



### Passos para usar:

- Instalar o Telegram no dispositivo móvel.
- Entrar no Telegram e procurar “*botfather*”.
- Clique nele e em seguida em Iniciar.
- Aparecerá um breve tutorial, clique em Menu e “*create new bot*”.
- Escolha um nome para o novo *bot* e em seguida um *username*(precisa terminar com *bot* e é **nome único**).
- Será gerado um *token*, anote-o.

# ESP32/ESP8266

## Telegram



- Qualquer pessoa que conheça o nome de usuário do *bot* pode interagir com ele;
- Para ignorarmos as mensagens que não são de nossa conta do Telegram, deve-se obter um ID de usuário do Telegram;
- Então, quando o *bot* receber uma mensagem, o ESP poderá verificar se o ID do remetente corresponde ao seu ID de usuário e processa ou ignora a mensagem;
- Em sua conta do Telegram, procure por “IDBot”;
- Após achá-lo, digite /getid, anote o retorno;

# ESP32/ESP8266 Telegram Funções



## ■ Principais funções:

- int getUpdates(long offset)
  - Obtém todas as mensagens pendentes do Telegram e as armazena em bot.messages.
- bool sendMessage(String chat\_id, String text, String parse\_mode = "")
  - Envia a mensagem para o chat\_id. Retorna se a mensagem foi enviada ou não.
- bool sendMessageWithReplyKeyboard(String chat\_id, String text, String parse\_mode, String keyboard, bool resize = false, bool oneTime = false, bool selective = false)
  - Para enviar teclados como resposta que podem ser usados como um tipo de menu. Retornará true se a mensagem for enviada com sucesso.
- bool sendMessageWithInlineKeyboard(String chat\_id, String text, String parse\_mode, String keyboard)
  - Para enviar teclados embutidos.
- bot.setMyCommands("[{\\"command\\":\"help\", \\"description\\\":\"get help\"}, {\\"command\\\":\"start\", \\"description\\\":\"start conversation\"}]")
  - Para definir comandos de bot programaticamente a partir do código. Os comandos serão mostrados em um local especial na área de entrada de texto

# ESP32/ESP8266 Telegram



## Executar :

- Exemplo\_ESP32\_Telegram\_Recebendo\_Mensagens\_01.ino
- Exemplo\_ESP8266\_Telegram\_Mandando\_Mensagens\_01.ino
- Exemplo\_ESP32\_Telegram\_ReplyKeyboardMarkup.ino
- Exemplo\_ESP32CAM\_Telegram\_Foto.ino

Obs.: Códigos precisam ser refinados.

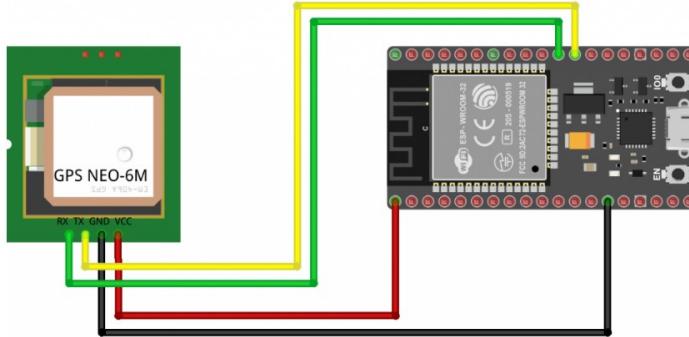
# GPS NEO-6M



- Tem por finalidade obter a geolocalização e fornecer os dados para o MCU;
- Conta com uma antena externa para melhorar a recepção de sinal;
- Comunicação é feita via serial (RX/TX);
- Especificações Técnicas:

- Tensão de operação: 2,7 a 5VDC
- Corrente de operação: 45mA
- Comunicação: serial
- Possui conector U.FL para antena
- Possui bateria para backup de informações
- Nível lógico: 3.3 e 5V
- Taxa de comunicação (padrão): 9600 bps
- Temperatura de operação: -40 a 85°Celsius
- Precisão: 5m

# GPS NEO-6M



# GPS NEO-6M



- O chip NEO-6M, fabricado pela u-Blox AG, é o *kernel* do módulo;
- Capacidade de rastrear até 22 satélites em 50 canais, enquanto consome apenas 45mA de corrente;
- Pode fazer até 5 atualizações de localização por segundo com precisão de posição horizontal de 2,5m;
- Possui o Power Save Mode (PSM), que permite uma redução no consumo de energia, ligando e desligando seletivamente partes do receptor;
- Reduz o consumo de energia do módulo (de 45 para 11mA), tornando-o adequado para aplicações sensíveis à energia;

# GPS NEO-6M



- A tensão de operação do chip NEO-6M varia de 2,7 a 3,6V;
- Possui regulador de 3.3V, já integrado ao módulo, tornando-o tolerante a 5V;
- São necessárias duas bibliotecas, EspSoftwareSerial e TinyGPSplus, baixá-las do Google Drive;
- A biblioteca EspSoftwareSerial é utilizada para obter os dados por qualquer pino da MCU;

# GPS NEO-6M TinyGPSPlus



## ■ Funcionalidades:

- TinyGPSPlus variável
  - Para definir o objeto gps.
- location – para obter a geolocalização atual.
  - location.lat() - para a latitude.
  - location.log() - para a longitude.
  - location.isValid() - para verificar se a informação é válida.
- date – para obter a data atual.
  - date.year() - para o ano.
  - date.month() - para o mês.
  - date.day() - para o dia.
- time – para obter o tempo atual.
  - time.hour() - para a hora.
  - time.minute() - para o minuto.
  - time.second() - para o segundo
  - time.centisecond() - para o centésimo do segundo.
- speed – para obter velocidade atual.
  - speed.mph() – velocidade em milhas por hora.
  - speed.mps() - velocidade em milhas por segundo.

# GPS NEO-6M TinyGPSPlus



## ■ Funcionalidades:

- altitude – para obter a altitude atual.
  - altitude.meters() - para a altitude em metros.
  - altitude.miles() - para a altitude em milhas.
  - altitude.kilometers() - para a altitude em quilômetros.
  - altitude.feet() - para a altitude em pés.
- satellites – para obter o número de satélites participantes visíveis.
  - satellites.values() - número de satélites.

**Analisar Exemplo\_ESP32\_Mosquitto\_GPS.ino**

# **Internet das Coisas com ESP8266/ESP32**

**FIM**