



BOSCH
Invented for life

Aula 2 - ESP 32

Docupedia Export

Author: Goncalves Donathan (CtP/ETS)
Date: 27-Feb-2023 14:09

Table of Contents

1	Especificações ESP 32	4
1.1	Mapeamento de pinos ESP 32	6
1.2	FreeRTOS	9
2	Programação em linguagem C	10
2.1	Linguagem C vs Python	10
2.1.1	Diferenças operadores C e Python	13
2.2	Estrutura de um programa em C	13
3	Arduino IDE	15
3.1	Conectando a placa	17
4	Protoboard	25
5	Iniciando a programação	27
5.1	Tipos de Sinais (Entrada x Saída)	27
5.2	Void setup()	27
5.3	Void loop()	28
5.4	pinMode()	29
5.5	digitalWrite()	29
5.6	Ligando um LED	30

Nesta aula iremos consultar as especificações fornecidas pelo fabricante da ESP 32, assim como suas funcionalidades e periféricos. Também será apresentada uma introdução à linguagem de programação C/C++, além da Arduino IDE, que será utilizada para construir nossos programas e gravá-los na ESP 32.

1 Especificações ESP 32

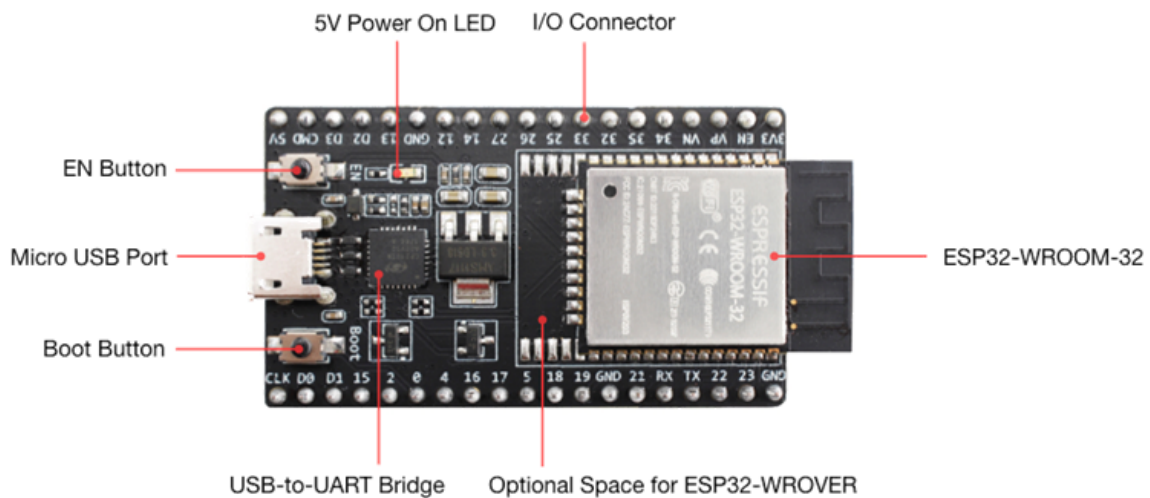
ESP32-WROOM-32 é um módulo microcontrolado Wi-Fi + BT + BLE genérico e poderoso que visa uma ampla variedade de aplicativos, que vão desde redes de sensores de baixa potência até as tarefas mais exigentes, como codificação de voz, streaming de música e decodificação de MP3.



No centro deste módulo está o chip ESP32-D0WDQ6. O chip embutido é projetado para ser escalável e adaptativo. Existem dois núcleos da CPU que podem ser controlados individualmente, e a frequência do clock da CPU é ajustável de 80 MHz a 240 MHz. O chip também possui um coprocessador de baixo consumo que pode ser usado no lugar da CPU para economizar energia ao executar tarefas que não requerem muito poder de computação, como o monitoramento de periféricos. ESP32 integra um rico conjunto de periféricos, variando de sensores de toque capacitivos, sensores Hall, SD interface de cartão, Ethernet, SPI de alta velocidade, UART, I²S e I²C. O chip ESP32-D0WDQ6 possui dois microprocessadores Xtensa® 32-bit LX6

A integração de Bluetooth®, Bluetooth LE e Wi-Fi garante que uma ampla gama de aplicativos possa ser direcionada, e que o módulo é versátil: o uso de Wi-Fi permite um grande alcance físico e conexão direta com a Internet através de um roteador Wi-Fi, enquanto usa o Bluetooth permite que o usuário se conecte convenientemente ao telefone ou transmita balizas de baixa energia para sua detecção.

Categories	Items	Specifications
Hardware	Module interfaces	SD card, UART, SPI, SDIO, I ² C, LED PWM, Motor PWM, I ² S, IR, pulse counter, GPIO, capacitive touch sensor, ADC, DAC, Two-Wire Automotive Interface (TWA [®] , compatible with ISO11898-1)
	On-chip sensor	Hall sensor
	Integrated crystal	40 MHz crystal
	Integrated SPI flash	4 MB
	Operating voltage/Power supply	3.0 V ~ 3.6 V
	Operating current	Average: 80 mA
	Minimum current delivered by power supply	500 mA
	Recommended operating temperature range	-40 °C ~ +85 °C
	Package size	(18.00±0.10) mm × (25.50±0.10) mm × (3.10±0.10) mm
	Moisture sensitivity level (MSL)	Level 3
Wi-Fi	Protocols	802.11 b/g/n (802.11n up to 150 Mbps) A-MPDU and A-MSDU aggregation and 0.4 μs guard interval support
	Frequency range	2.4 GHz ~ 2.5 GHz
Bluetooth	Protocols	Bluetooth v4.2 BR/EDR and BLE specification
	Radio	NZIF receiver with -97 dBm sensitivity
		Class-1, class-2 and class-3 transmitter
	Audio	AFH CVSD and SBC



Boot Button: Este botão é utilizado para gravar o programa no módulo, e durante o upload devemos manter o botão boot pressionado.

Micro USB: A placa é energizada utilizando um port interno Micro USB. Este port também pode ser utilizado para conectar a placa no computador e realizar upload do programa.

RESET Button: Este botão é utilizado para resetar a placa.

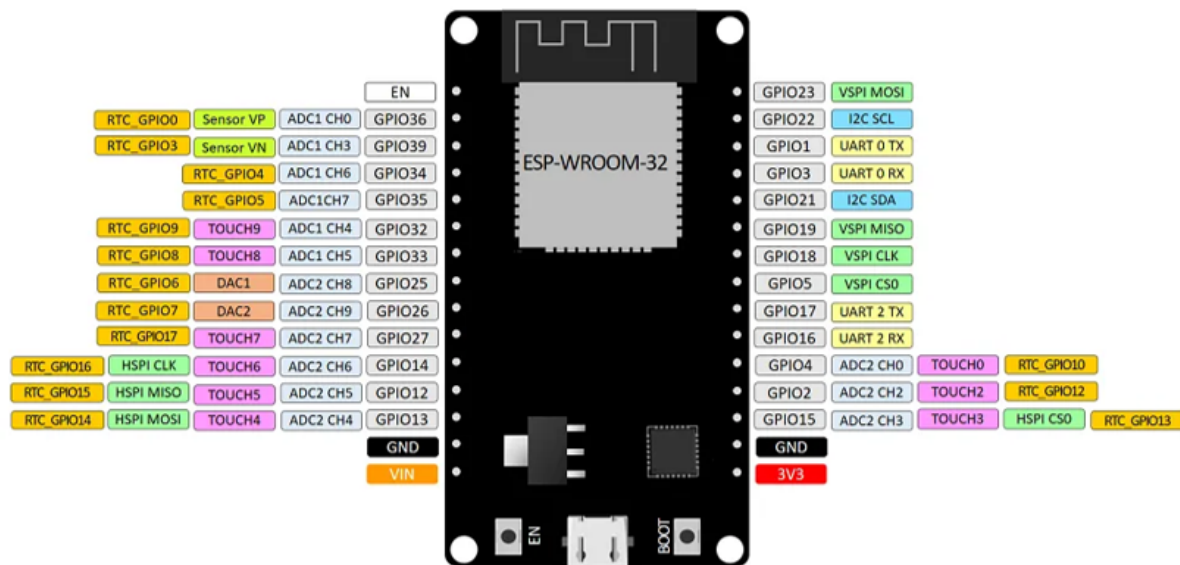
Touch Sensor: Este módulo possui 10 sensores capacitivos conectados a pinos GPIO:

- Touch0 (GPIO4)
- Touch1 (GPIO0)
- Touch2 (GPIO2)
- Touch3 (GPIO15)
- Touch4 (GPIO13)
- Touch5 (GPIO 12)
- Touch6 (GPIO14)
- Touch7 (GPIO27)
- Touch8 (GPIO33)
- Touch9 (GPIO32)

Estes pinos GPIO podem sentir variações se tocados por qualquer coisa que conduz energia elétrica, como a pele humana por exemplo. Portanto, quando tocamos o pino GPIO com nosso dedo, uma variação é gerada e é lida pelo sensor. Podemos utilizar a função **touchRead(GPIO)** para ler o valor do sensor Touch.

Sensor Hall Effect: Esta placa possui um sensor integrado Hall Effect, que é utilizado para medir mudanças no campo magnético ao seu redor.

1.1 Mapeamento de pinos ESP 32



- ADC2_CH5 (GPIO 12)
- ADC2_CH6 (GPIO 14)
- ADC2_CH7 (GPIO 27)
- ADC2_CH8 (GPIO 25)
- ADC2_CH9 (GPIO 26)

DAC (Digital to Analog Converter) Channels:

A placa possui dois DAC de 8 bits que podem ser utilizados para converter um sinal digital em um sinal analógico:

- DAC_1 (GPIO25)
- DAC_2 (GPIO26)

UART (*Universal Asynchronous Receiver-Transmitter*) Pins:

A placa de desenvolvimento ESP32 tem três interfaces UART, - UART0, UART1 e UART2, que fornecem comunicação assíncrona entre os dispositivos habilitados para UART até uma velocidade de 5 Mbps. A interface UART na placa de desenvolvimento também consiste em dois pinos extras que permitem que o receptor e o remetente alertem um ao outro sobre seu estado atual - pinos de sinais CTS e RTS.

UART0 Pins:

- U0 TXD (GPIO1)
- U0 RXD (GPIO3)
- U0 CTS (GPIO19)
- U0 RTS (GPIO22)

UART1 Pins:

- U1 TXD (GPIO10)
- U1 RXD (GPIO9)
- U1 CTS (GPIO6)
- U1 RTS (GPIO11)

UART2 Pins:

- U2 TXD (GPIO17)
- U2 RXD (GPIO16)
- U2 CTS (GPIO8)
- U2 RTS (GPIO7)

Pinos SPI:

A placa possui três SPIs (SPI, HSPI e VSPI) nos modos escravo e mestre. Esses pinos são usados para conectar os dispositivos habilitados para SPI externos.

SPI Pins on board:

- SPI_D (GPIO8)
- SPI_WP (GPIO10)
- SPI_HD (GPIO9)
- SPI_Q (GPIO7)
- SPI_CLK (GPIO6)
- SPI_CS0 (GPIO11)

HSPI Pins:

- V_SPI_ID (GPIO23)
- V_SPI_WP (GPIO22)

- V_SPI_HD (GPIO21)
- V_SPI_Q (GPIO19)
- V_SPI_CLK (GPIO18)
- V_SPI_CS0 (GPIO5)

VSPI Pins:

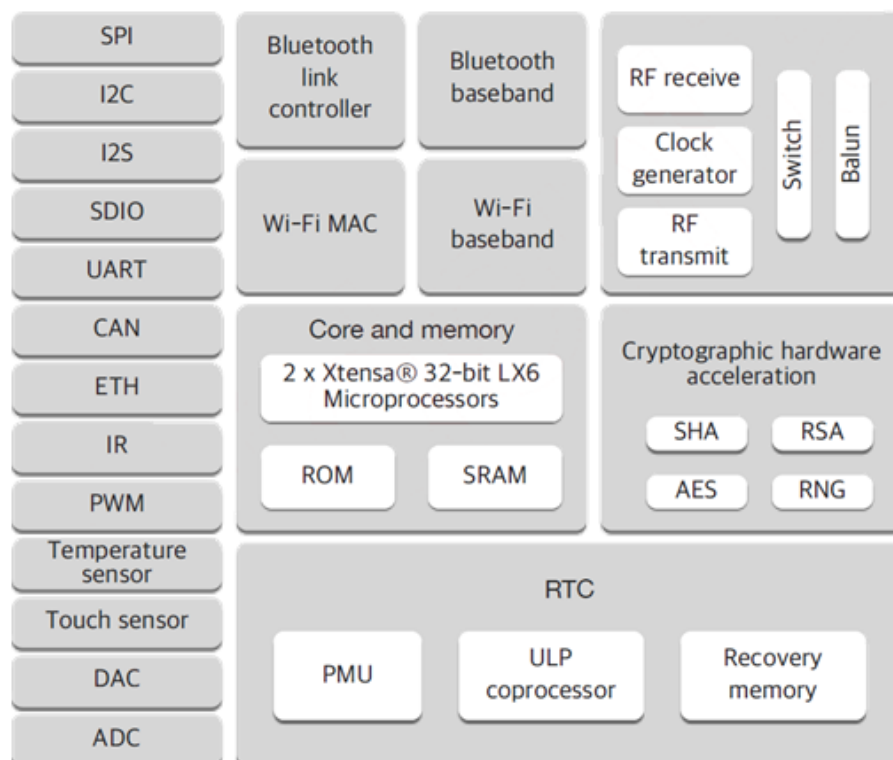
- HSPI_ID (GPIO13)
- HSPI_WP (GPIO2)
- HSPI_HD (GPIO4)
- HSPI_Q (GPIO12)
- HSPI_CLK (GPIO14)
- HSPI_CS0 (GPIO15)

Pinos PWM:

A placa vem com 25 pinos habilitados para PWM (quase todos os pinos GPIO). A saída PWM pode ser usada para acionar motores digitais e LEDs.

Pino EN ou ENABLE:

En é uma abreviação de Enable. Este pino habilita o regulador de 3,3 V. Quando ele é colocado em nível lógico baixo, o microcontrolador é resetado.



O sistema operacional escolhido para ESP32 é freeRTOS com LwIP; TLS 1.2 com aceleração de hardware é integrado no módulo também. A atualização segura (criptografada) pelo ar (OTA) também é suportada, para que os usuários possam atualizar seus produtos mesmo após seu lançamento, com custo e esforço mínimos.

1.2 FreeRTOS

FreeRTOS é um sistema operacional criado para operar em sistemas embarcados e sistemas IoT. Este é um sistema operacional em tempo real e de código aberto, tendo como objetivo facilitar a programação, a implantação, a segurança, a conexão e o gerenciamento de dispositivos de borda pequenos com baixo consumo de energia. Distribuído gratuitamente sob a licença de código aberto do MIT, o FreeRTOS inclui um kernel e um conjunto crescente de bibliotecas de software adequados para o uso em vários setores e aplicações. Isso inclui conectar em segurança seus dispositivos pequenos de baixo consumo de energia a Serviços de Nuvem da AWS, como o AWS IoT Core, ou a dispositivos de borda mais potentes que executam o AWS IoT Greengrass.

Você poderá conectar os dispositivos do FreeRTOS com segurança a serviços na nuvem como o AWS IoT Core, a um dispositivo de borda local ou a um dispositivo móvel via Bluetooth Low Energy e atualizá-los remotamente usando o recurso de atualização OTA disponibilizado pelo AWS IoT Device Management.



2 Programação em linguagem C

A programação na ESP-32 é feita em linguagem C e C++, e iremos utilizar a Arduino IDE, que contém bibliotecas para manipulação dos periféricos do microcontrolador.

Agora veremos pequenas diferenças entre a linguagem Python e a linguagem C:

2.1 Linguagem C vs Python

Para declarar nossas variáveis no Arduino é preciso fazer um casting, ou seja, declarar o tipo da variável.

Casting

```
int x;  
int y = 2;  
float z = 4.2;  
char w = "abc";
```

Note que no final de cada linha foi colocado um ';', o uso desse símbolo é obrigatório ao final de toda instrução do código. Além disso, podemos declarar a variável com um valor predefinido ou não.

Para importar bibliotecas é utilizado o #include

Include

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

Podemos definir um valor fixo para um nome, utilizando #define

Define

```
#define ZERO 0
```

Em Python os blocos de instrução eram definidos pela indentação.

Python instruction block

```
if x == 0  
    print('x = 0')
```

Em linguagem C os blocos de instrução são definidos por chaves.

C instruction block

```
if (x == 0) {  
    print("x = 0");  
}
```

- A condição **if** deve estar entre parênteses
- Toda string deve estar entre aspas duplas, a não ser que seja apenas um caractere.
- Em C++ não existe *elif*, mas sim *else if*.

Veja como é a notação do comando **for** em C++:

For command

```
for(i = 0; i < 10; i++) {  
    print(i);  
}
```

Dentro dos parênteses o comando **for** recebe três argumentos:

- **i=0**: a variável começa o loop valendo 0;
- **i<10**: o loop vai se repetir enquanto **i** for menor que 10;
- **i++**: é o mesmo que escrever "**i = i+1**", significa que a cada repetição a variável será incrementada de 1.

Exemplo de comando **While**:

While command

```
while(x > 10) {  
    print("x é maior que 10");  
}
```

Para gerar um loop infinito (**while True**):

While true

```
while(1) {  
    print("loop infinito");  
}
```

Em linguagem C também existe o comando **break** para sair de um loop.

Break command

```
int i = 0;

while(1) {
    i++;
    if(i == 5) {
        break;
    }
}
```

Declaração de funções:

Function declare

```
int somavar(int x, int y, int z) {
    return x+y+z;
}
```

O exemplo acima mostra uma declaração de função, vamos ver do que ela é composta:

- **int:** o tipo de dado do valor que a função retorna, se ela não retornar nada, se escreve void;
- **somaVar:** o nome da função, é a partir dele que ela é chamada no decorrer do programa;
- **(int x, int y, int z):** os parâmetros da função, note que deve ser feito o casting neste caso;
- **return:** o valor retornado pela função.

Comando **switch**:

Switch command

```
switch(x) {
    case 1:
        print("x = 1");
        break;
    case 2:
        print("x = 2");
        break;
    case 3:
        print("x = 3");
    default:
        break;
}
```

- O exemplo acima é do comando switch. Ele não existe em Python, mas é muito útil em linguagem C;

- Ele irá analisar o valor da variável `x`, e executará a instrução caso seja algum dos valores declarados;
- Se o valor for diferente dos declarados, ele executa a instrução *default*.

2.1.1 Diferenças operadores C e Python

C = Python

Operadores de comparação

`!=` (diferente de)

`<` (menor que)

`<=` (menor ou igual a)

`==` (igual a)

`>` (maior que)

`>=` (maior ou igual a)

C:

Operadores Booleanos

`!` (NÃO Lógico)

`&&` (E lógico)

`||` (OU lógico)

Python: No python nós utilizávamos as palavras para referir a esses operadores booleanos:

- **NOT**
- **AND**
- **OR**

C:

Números aleatórios

`random()`

Python: usamos o `import random → random.randint()`

2.2 Estrutura de um programa em C

Structure of C Program

<i>Header</i>	<code>#include <stdio.h></code>
<i>main()</i>	<code>int main() {</code>
<i>Variable declaration</i>	<code>int a = 10;</code>
<i>Body</i>	<code>printf("%d ", a);</code>
<i>Return</i>	<code>return 0; }</code>



3 Arduino IDE

Uma **IDE (Integrated Development Environment)** é um recurso extremamente útil ao criar aplicações que envolvem programação e desenvolvimento, e também quando há integração com outras plataformas. Nela, vários processos são unificados em uma única interface, sendo alguns deles:

- Editor de texto para códigos-fonte;
- Ferramentas de compilação, que transforma o código escrito pelo usuário em um código compreensível para a máquina;
- Auto-preenchimento de comandos, prevendo o trecho que o desenvolvedor irá digitar;
- Ferramentas de *debug*, que realiza testes em trechos de códigos, a fim de se encontrar erros com facilidade;
- Otimização do código por meio do compilador;

Neste módulo, iremos utilizar a IDE do Arduino para a programação da ESP 32. Nela existem diversas características e benefícios, sendo alguns deles:

- Diversidade de bibliotecas para recursos específicos;
- Interface simples e intuitiva;
- Baixa complexidade no editor de códigos e compilação;
- Compatibilidade com um grande número de placas diferentes;
- Linguagem própria baseada na linguagem C e C++;
- Recomendado para aplicações com fins educacionais.

O **ciclo de programação** na IDE Arduino pode ser dividido da seguinte maneira:

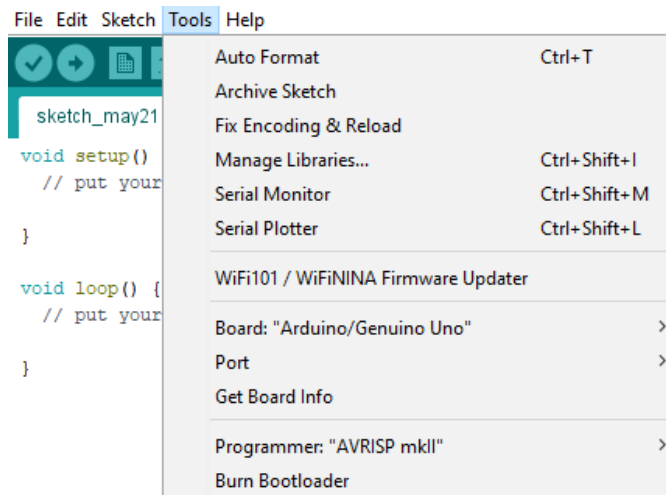
1. Conexão da placa a uma porta USB do computador;
2. Desenvolvimento de um sketch com comandos para a placa;
3. Upload do sketch, utilizando a comunicação USB;
4. Aguardar reinicialização, após ocorrerá a execução do sketch criado;
5. A partir do upload para o Arduino, o computador não é mais necessário. O Arduino executará o sketch criado, desde que seja ligado a uma fonte de energia.

O IDE é dividido em três partes:

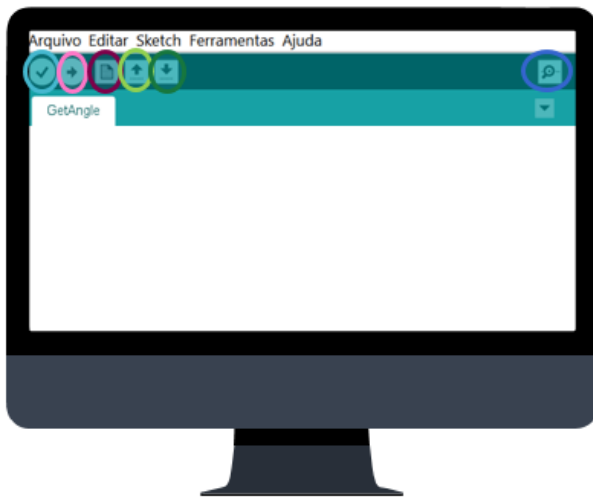
1. Toolbar no topo;
2. Código ou a Sketch Window no centro;
3. Janela de mensagens na base.



Os botões na Toolbar fornecem acesso rápido às funções mais utilizadas dentro desses menus:



Abaixo estão descritos os atalhos da IDE:



01

Verify

- ▶ Verifica se existe erro no código digitado.

02

Upload

- ▶ Compila o código e grava na placa Arduino se corretamente conectada;

03

New

- ▶ Cria um novo sketch em branco.

04

Open

- ▶ Abre um sketch, presente no sketchbook.

05

Save

- ▶ Salva o sketch ativo

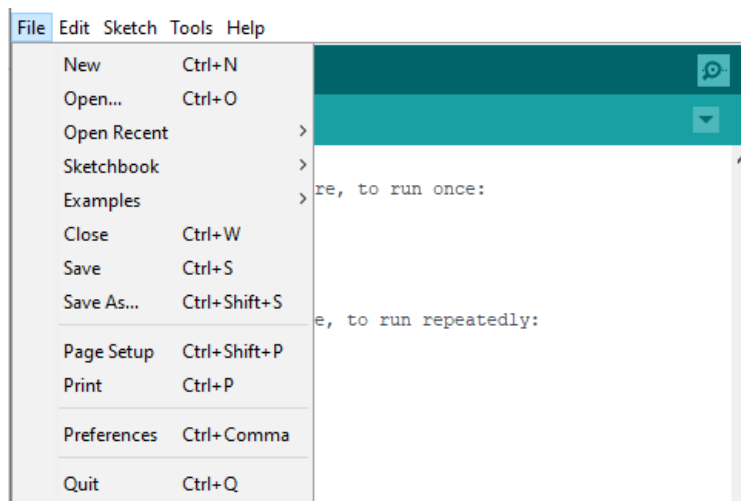
06

Serial monitor

- ▶ Abre o monitor serial.

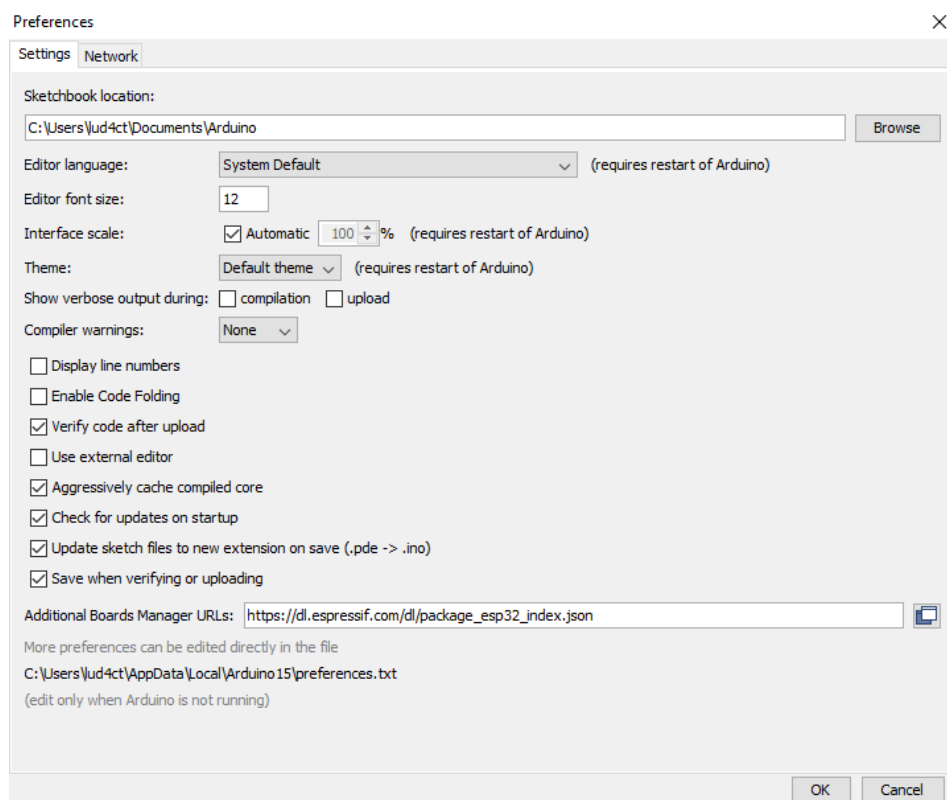
3.1 Conectando a placa

Para conectarmos a ESP 32 na Arduino IDE, devemos primeiramente baixar o board respectivo. Para isso devemos ir em File → Preferences

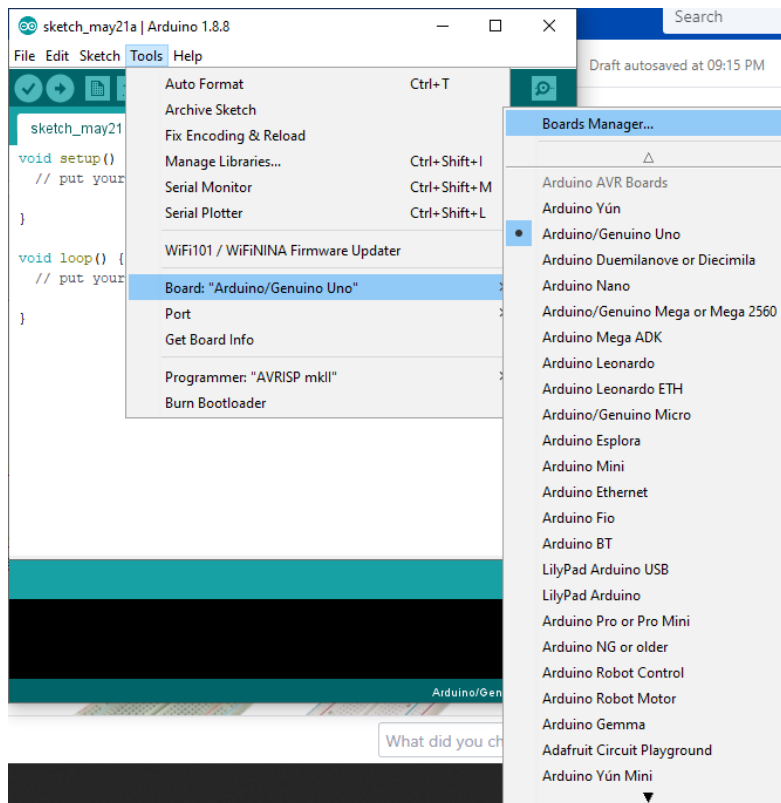


Na tela que abrir, adicione o seguinte link em "Additional Boards Manager URLs" e clique em OK:

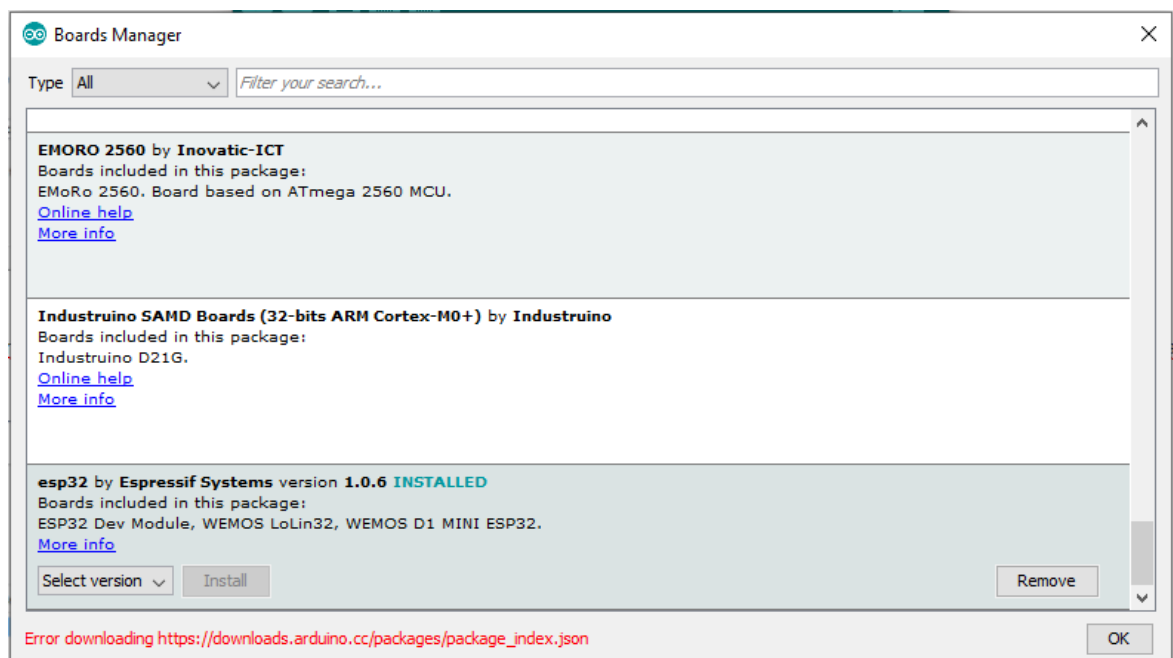
https://dl.espressif.com/dl/package_esp32_index.json



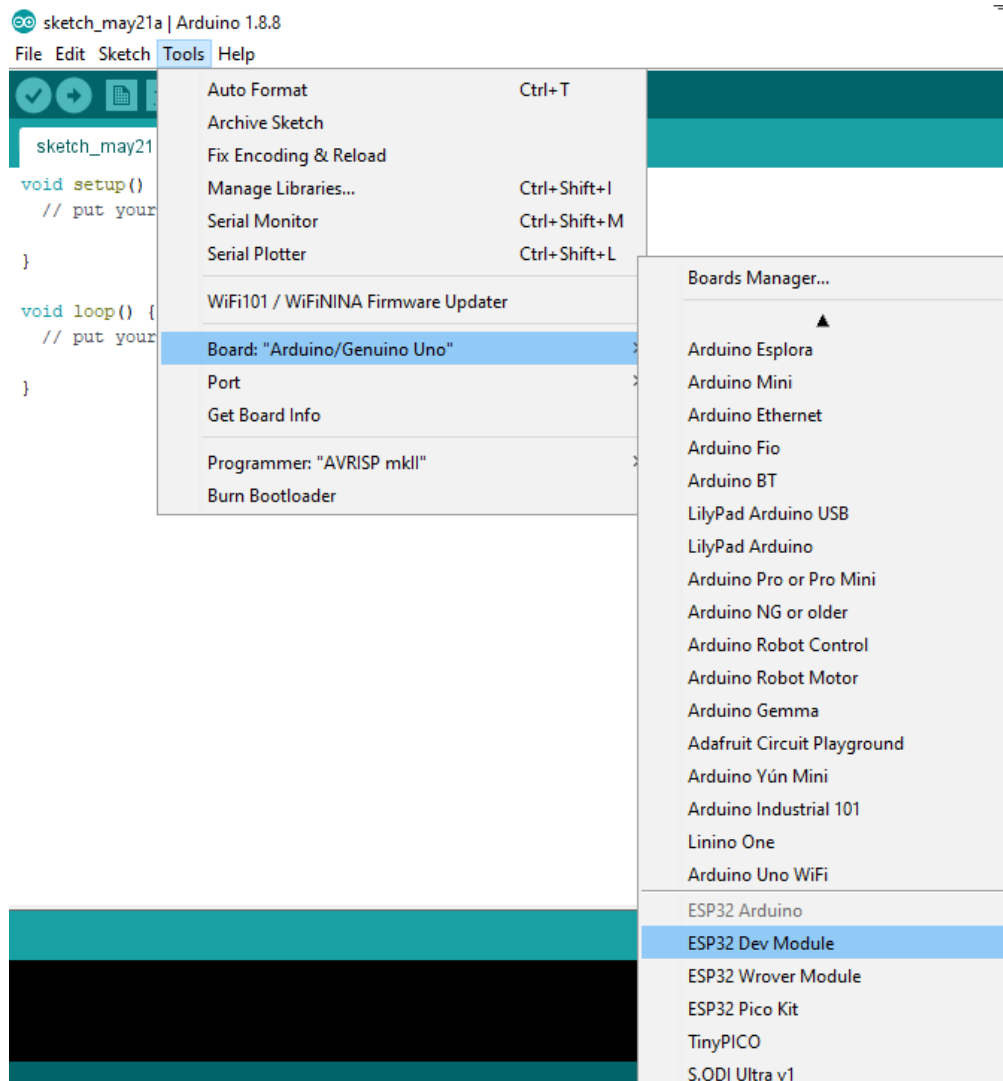
Após este procedimento, retorne a tela inicial da IDE e navegue até Tools → Board → Boards Manager, e então aguarde até que ele realize o download da listagem completa de placas.



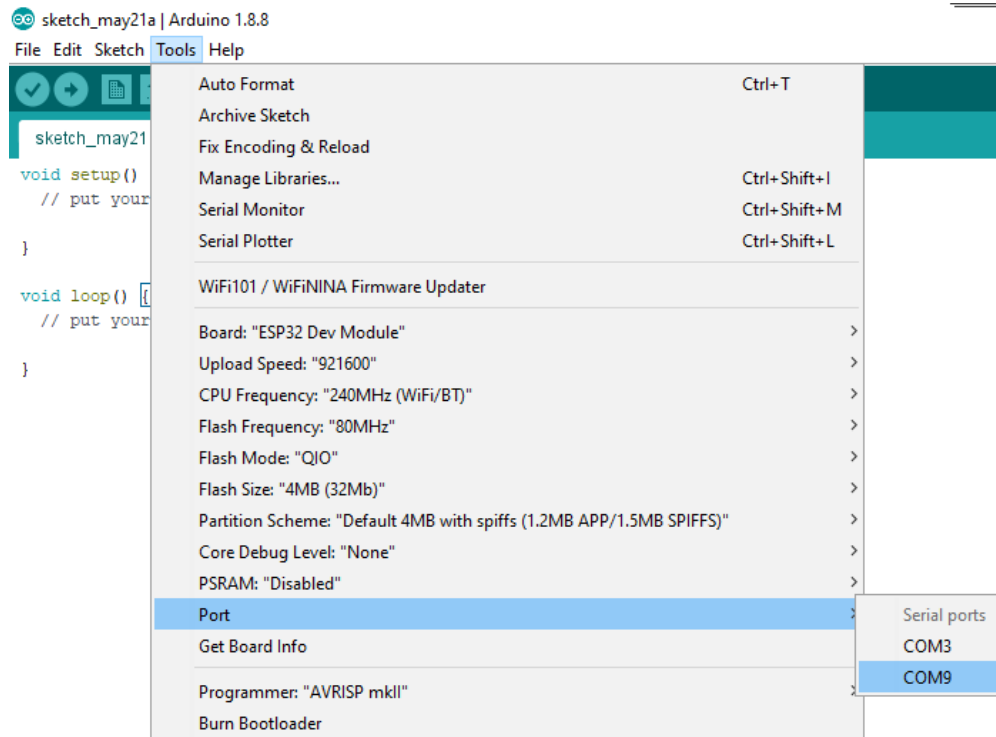
E então, encontre a board "esp32 by Espressif Systems" e clique em Install e aguarde até que seja instalado por completo.



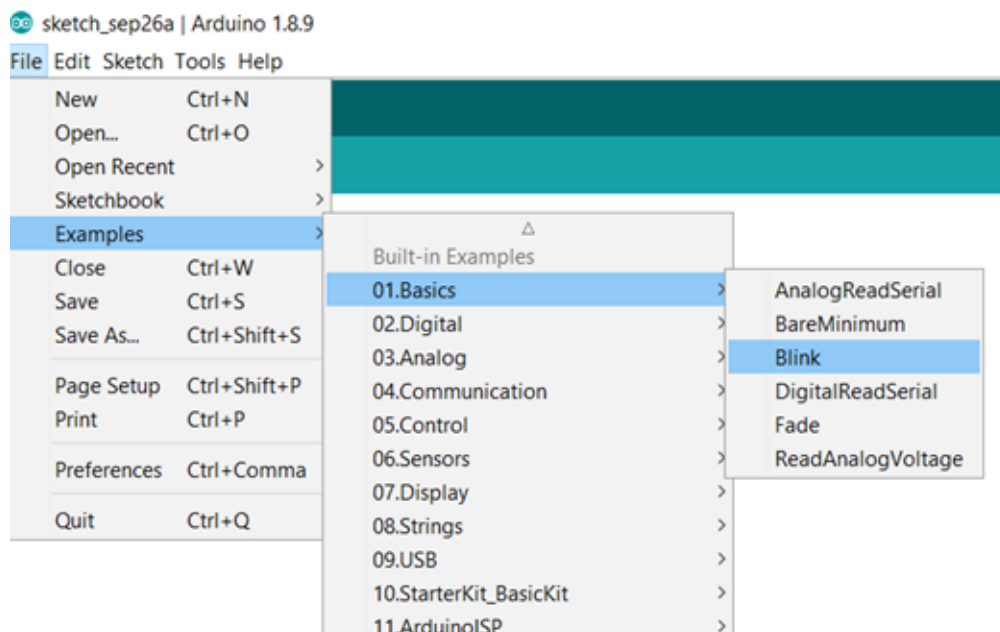
Para o próximo passo, conecte a placa no computador e selecione "ESP32 Dev Module" em Tools → Board



Selecione a porta serial em que a placa está conectada. (Nesse caso, a ESP 32 foi conectada na porta COM9)



- Para iniciar os estudos com a IDE do Arduino, o programa “Blink” será utilizado.
- Siga o passo a passo da imagem abaixo.
- Após selecionar “Blink” uma nova página se abrirá.
- Nesse momento, a janela anterior pode ser fechada, e então apenas a janela com o programa deverá ficar aberta.



O seguinte programa será aberto:

Blink

```

/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to
  the correct LED pin independent of which board is used.
  If you want to know what pin the on-board LED is connected to on your Arduino
  model, check the Technical Specs of your board at:
  https://www.arduino.cc/en/Main/Products

  modified 8 May 2014
  by Scott Fitzgerald
  modified 2 Sep 2016
  by Arturo Guadalupi
  modified 8 Sep 2016
  by Colby Newman

  This example code is in the public domain.

  http://www.arduino.cc/en/Tutorial/Blink
*/

// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin LED_BUILTIN as an output.
  pinMode(LED_BUILTIN, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(LED_BUILTIN, HIGH); // turn the LED on (HIGH is the voltage level)
  delay(1000);                     // wait for a second
  digitalWrite(LED_BUILTIN, LOW);  // turn the LED off by making the voltage LOW
  delay(1000);                     // wait for a second
}

```

Antes de compilar e gravar, altere a constante "LED_BUILTIN" para "2", pois assim iremos atuar no pino 2 da ESP, que é o pino com o led onboard. Assim, o código ficará dessa forma:

Blink

```

/*
  Blink

  Turns an LED on for one second, then off for one second, repeatedly.

  Most Arduinos have an on-board LED you can control. On the UNO, MEGA and ZERO
  it is attached to digital pin 13, on MKR1000 on pin 6. LED_BUILTIN is set to

```

the correct LED pin independent of which board is used.
 If you want to know what pin the on-board LED is connected to on your Arduino model, check the Technical Specs of your board at:
<https://www.arduino.cc/en/Main/Products>

modified 8 May 2014
 by Scott Fitzgerald
 modified 2 Sep 2016
 by Arturo Guadalupi
 modified 8 Sep 2016
 by Colby Newman

This example code is in the public domain.

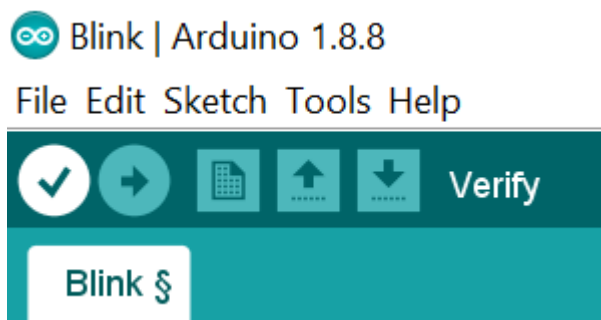
<http://www.arduino.cc/en/Tutorial/Blink>

```
*/

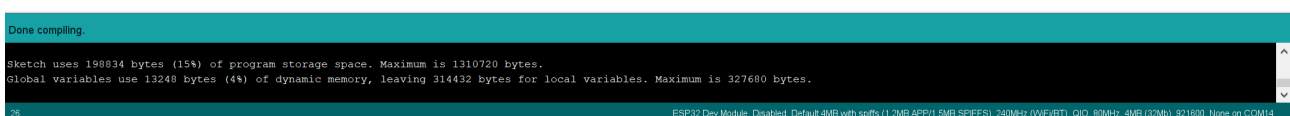
// the setup function runs once when you press reset or power the board
void setup() {
  // initialize digital pin 2 as an output.
  pinMode(2, OUTPUT);
}

// the loop function runs over and over again forever
void loop() {
  digitalWrite(2, HIGH);   // turn the LED on (HIGH is the voltage level)
  delay(1000);             // wait for a second
  digitalWrite(2, LOW);    // turn the LED off by making the voltage LOW
  delay(1000);             // wait for a second
}
```

Para verificar se o código está correto deve-se clicar no ícone "Verify".

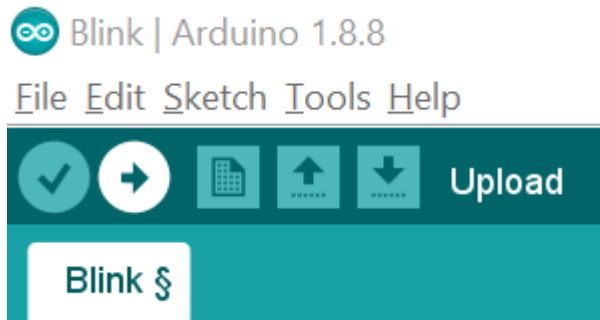


Após a compilação é exibida uma mensagem de status da operação, e caso esteja tudo certo será exibida a quantidade de bytes gerados pelo programa:



Para gravar o código na memória flash do micro-controlador é necessário clicar no ícone "Upload".

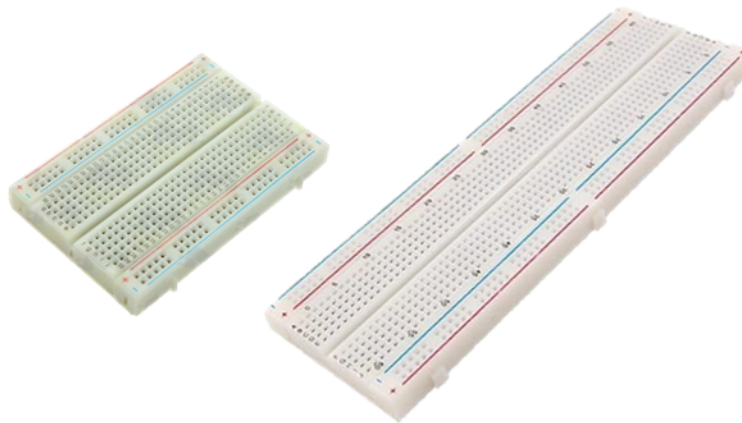
O código será transferido para a placa e após algum tempo o LED ligado ao pino 2 começará a piscar em intervalos de 1 segundo.



4 Protoboard

Protoboard = Matriz de contato = Breadboard = Placa de ensaio

Com ela é possível montar circuitos sem a necessidade de soldar qualquer componente. Então se você não tem certeza de como um determinado circuito irá se comportar durante seu funcionamento, a protoboard é o lugar mais recomendado para montar este circuito e efetuar todos os testes necessários. Uma utilização muito comum é interligar dispositivos com o Arduino.



01 Barramentos de Alimentação

- ▶ São os barramentos verticais nas extremidades da protoboard, que geralmente são usados para alimentação dos circuitos. É comum encontrar duas linhas em cima e duas em baixo. Com isso podemos observar que as linhas de alimentação estão separadas no meio. Para os modelos menores de protoboard não existe esta separação.

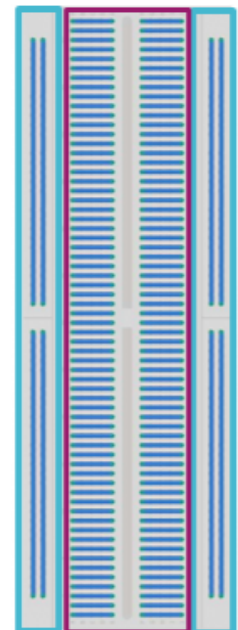
02

Barramentos de Prototipagem (área de trabalho)

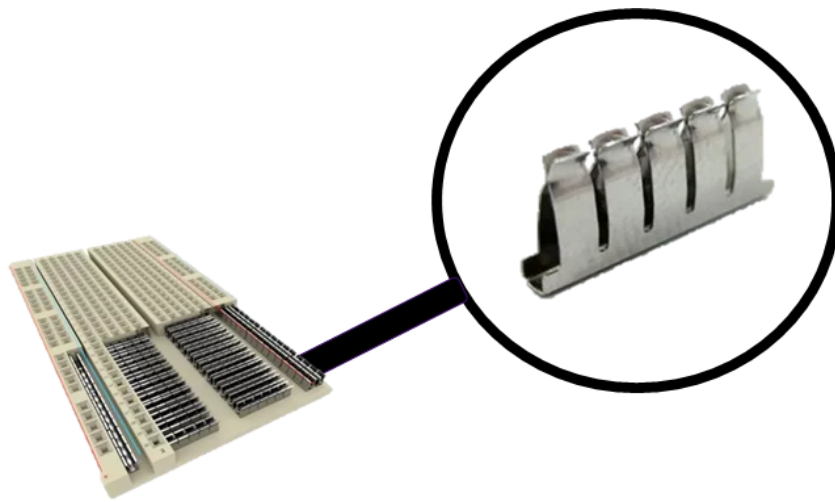
- ▶ Barramentos horizontais usados para a montagem do circuito. As colunas formam agrupamentos de 5 furos em 5 furos, ou seja, assim que inserirmos um componente em um dos furos, ele estará eletricamente conectado a todos os outros furos daquela coluna.

É importante você saber que os barramentos de cima não são conectados com os de baixo;

Existe um espaçamento central entre os barramentos horizontais que os separa em barramento horizontais esquerda e direita.



Por dentro do protoboard existe uma matriz de contatos. Vários barramentos metálicos em paralelo tal como podemos ver nessa imagem em que cada linha desenhada representa um barramento.

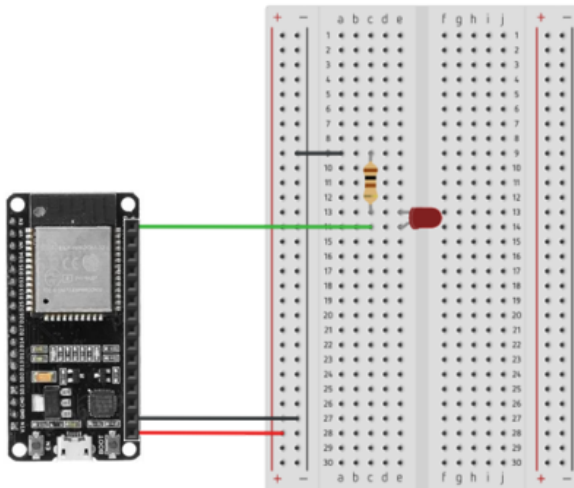


5 Iniciando a programação

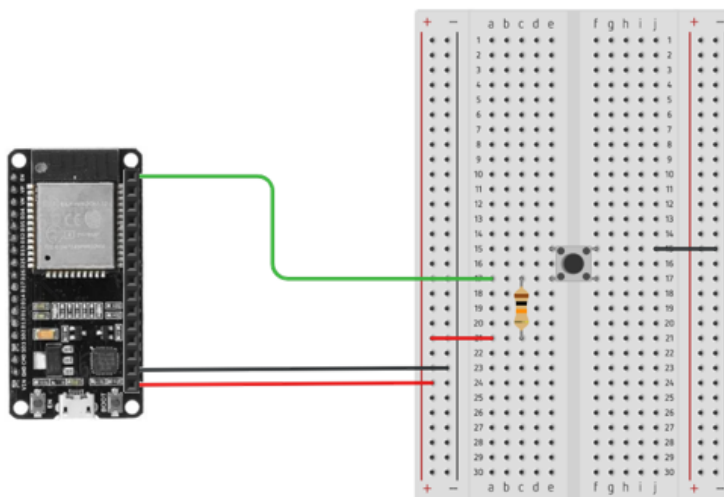
Vamos entender com mais detalhes alguns pontos importantes para nossas aplicações e programas:

5.1 Tipos de Sinais (Entrada x Saída)

Saída (OUTPUT): São os sinais emitidos pela ESP 32, que resultam em alguma execução externa, como o sinal emitido para acender um led.



Entrada (INPUT): São os sinais emitidos pelo meio externo e que são captados pela ESP 32. Por exemplo, o sinal emitido quando você aciona um botão.



5.2 Void setup()

A função `void setup()` é chamada no início da programação. Usamos ela para inicializar variáveis, e configurar o modo dos pinos (INPUT ou OUTPUT).



5.3 Void loop()

Depois do void setup, a programação entra no void loop(), na qual, realiza exatamente que seu nome sugere, repete o que estiver dentro dele até que a placa seja desligada.



5.4 pinMode()

- Configura o pino para que ele funcione como o especificado, se for uma entrada u uma saída.
- Ele também pode ativar os resistores internos de pull-up com o modo INPUT_PULLUP.
- Sintaxe: pinMode(pino, modo)
- Pino – é o numero do pino em que se quer configurar.
- Modo – INPUT, OUTPUT OU INPUT_PULLUP.



5.5 digitalWrite()

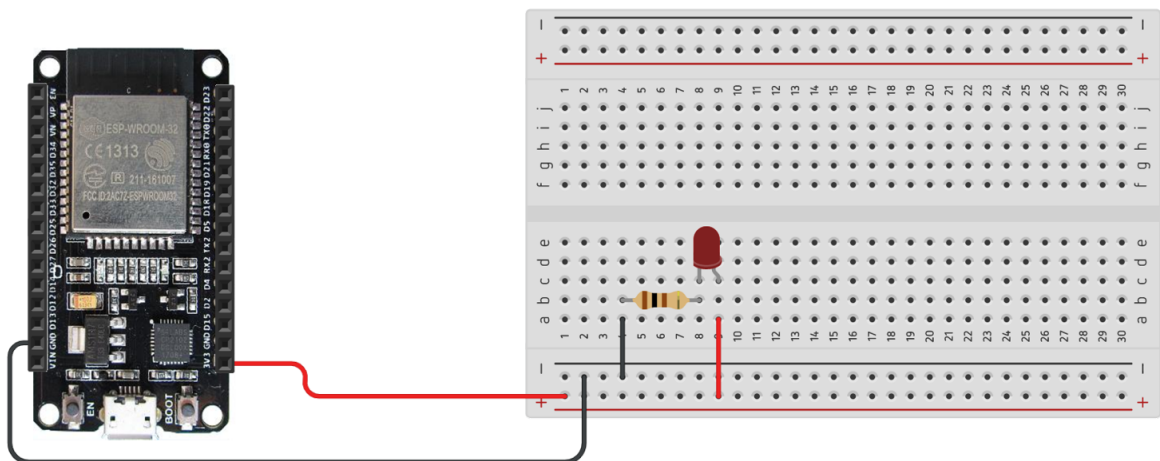
- Aciona um valor HIGH ou LOW em um pino.
- Se o pino for configurado como saída (OUTPUT) com a função pinMode(), sua tensão será acionada para o valor correspondente: 5V (ou 3.3V em placas alimentadas com 3.3V como o DUE) para o valor HIGH, 0V (ou ground) para LOW.
- Se o pino for configurado como entrada (INPUT), a função digitalWrite() irá ativar (HIGH) ou desativar (LOW) o resistor interno de pull-up no pino de entrada. É recomendado configurar pinMode() com INPUT_PULLUP para ativar o resistor interno de pull-up. Veja o tutorial sobre pinos digitais para mais informações.
- Se você não configurar o pino com pinMode() e OUTPUT, e conectar um LED ao pino, quando chamar digitalWrite(HIGH), o LED pode aparecer um pouco apagado.
- Sintaxe: digitalWrite(pino, valor)
- Valor - HIGH ou LOW

digitalWrite

```
void setup() {  
    pinMode(13,OUTPUT);  
}  
  
void loop() {  
    digitalWrite(13,HIGH);  
    delay(1000);  
    digitalWrite(13,LOW);  
    delay(1000);  
}
```

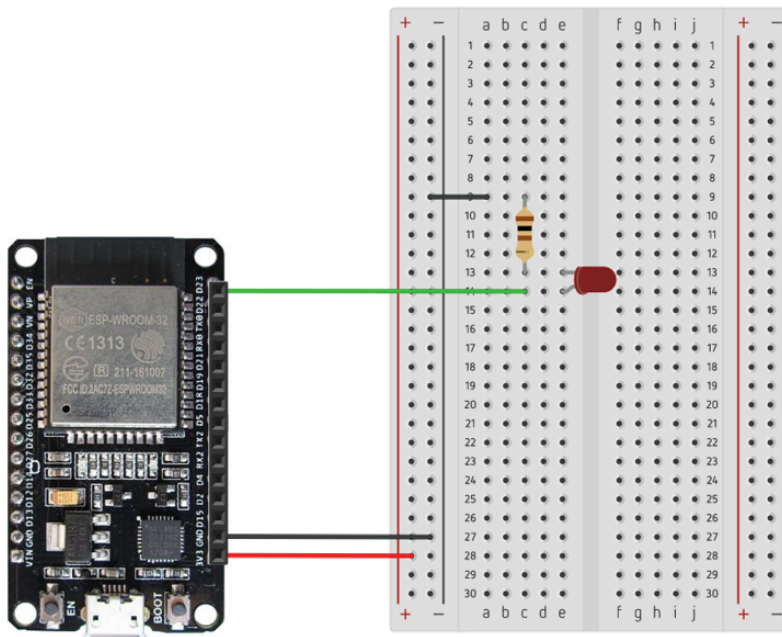
5.6 Ligando um LED

Existe mais de uma maneira de se montar, neste caso ligaremos o led direto no 3,3 V e no GND com o resistor limitando a corrente. Neste caso o led se manterá acesso sem mudanças até que a ESP não esteja energizada.



Exercício 1: Piscar

- Tarefa: Piscar um LED em uma frequência de 2 Hz.
- Tempo estimado: 10 minutos.
- Dica: use HIGH para acender o LED, e LOW para apagá-lo.
- Ex.: digitalWrite(led, HIGH);



Exercício 2: Piscar 3 leds

- Tarefa: Piscar led vermelho, amarelo e verde nesta sequência um de cada vez. Ao piscar o led verde, iniciar novamente a sequência.
- Tempo estimado: 15 minutos.
- Dica: utilize For, e outra função (void x ();) para praticar mais de uma forma de programação.

