Sinal Digital

O sinal digital, possui um número finito de valores. Esse sinal está presente no nosso dia a dia e, o conceito adotado não precisa ser aplicado diretamente a um eletrônico. Por exemplo, uma porta pode estar aberta ou fechada, uma pessoa pode estar acordada ou dormindo e assim por diante. Pegou o conceito?

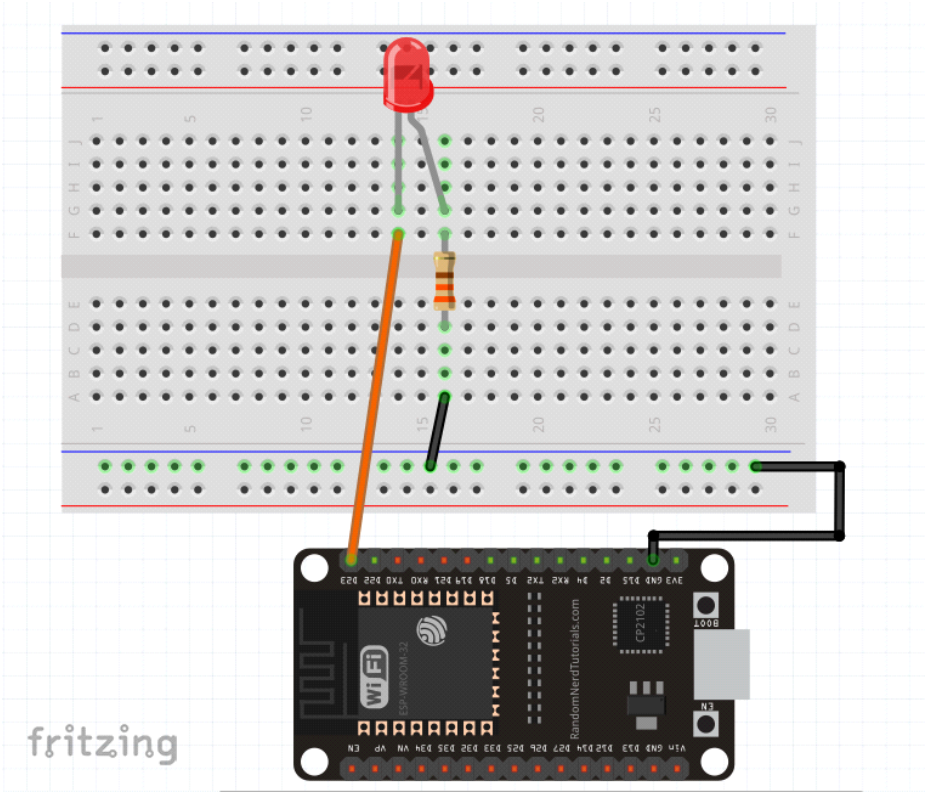
O sinal digital, é aquele que pode assumir apenas dois valores, sendo eles 0 e 1. Para aqueles que já pertencem ao mundo da eletrônica ou computação conhecem esse conceito com número binário.

Aqui vamos aprender como podemos aplicar esse conceito através da programação utilizando o ESP32 com a linguagem de programação MicroPython.

Para iniciarmos a aula, você vai precisar de:

- ESP32;
- LED (qualquer cor);
- Resistor 220 Ohm ou 330 ohms e 10 ohms;
- Push Button;
- Jumpers.

Agora, é só realizar a montagem abaixo:



Código – Saída Digital

No código abaixo, vamos fazer o nosso ‘Hello World’ das plaquinhas microcontroladas, mais conhecido como Blink (pisca-pisca), onde vamos visualizar o conceito de uma saída digital. Copie o código abaixo no IDE do Thonny e execute na sua placa:

```
from machine import Pin #acessar os pinos da placa
from time import sleep#intervalos de temporização
#import time

led = Pin(23, Pin.OUT) #Configurar o pino 2 como uma saída digital

while True:
    led.value(1) #liga o led
    sleep(1) #aguarda 1 segundo
    led.value(0) #desliga o led
    sleep(1)
```

Explicação

Na primeira linha, vamos importar a função Pin do módulo machine, que nos possibilita ter acesso aos pinos da placa:

```
from machine import Pin #acessar os pinos da placa
```

Agora, vamos importar a função sleep da biblioteca time, utilizada para intervalos de temporização. Ou, você pode importar diretamente a biblioteca time, porém, antes de chamarmos as funções será necessário usar time.sleep, por exemplo.

```
2 from time import sleep#intervalos de temporização
3 #import time
```

Vamos configurar o nosso pino como uma saída digital. Para isso, vamos chamar a função Pin (ou, machine.Pin, caso você tenha importado apenas o módulo). Entre parênteses, primeiro vamos identificar qual o pino estamos utilizando e depois dizer se uma saída ou uma entrada digital (OUT / IN). Todas essas informações devem estar atribuídas ao objeto led, como vemos abaixo:

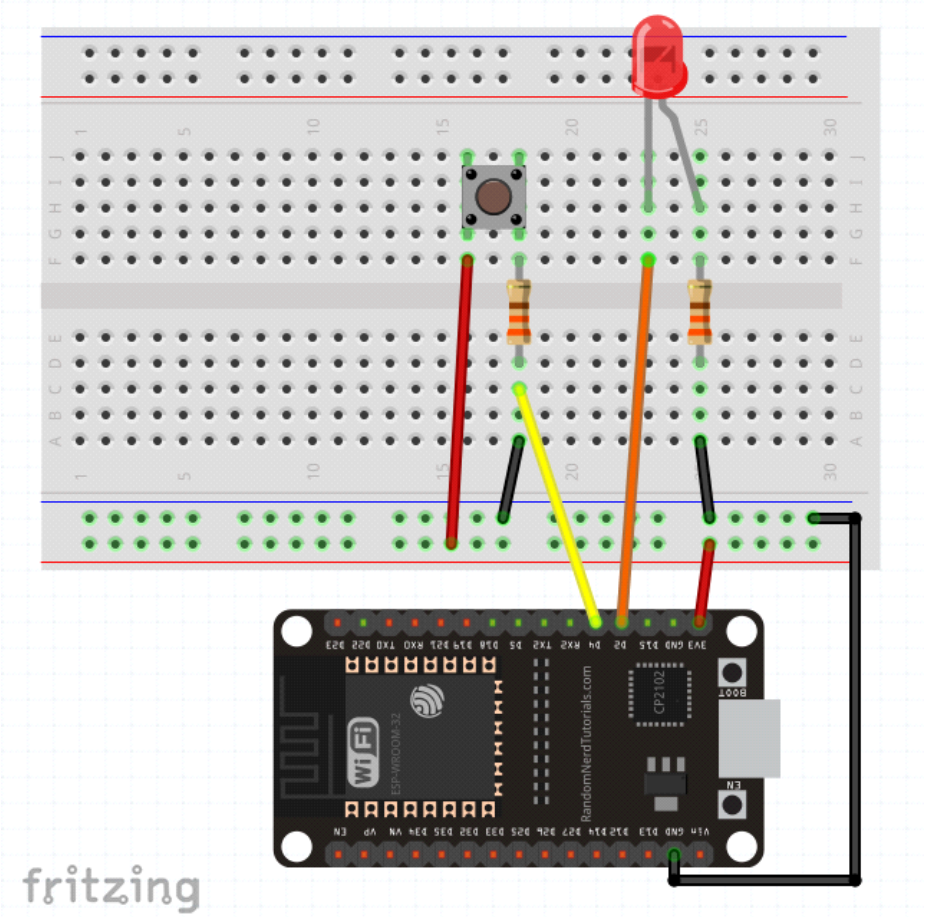
```
5 led = Pin(23, Pin.OUT) #Configurar o pino 2 como uma saída digital
```

Por fim, criarmos o laço de repetição infinita, mais conhecido como loop e para esse objetivo utilizamos o while True. Dentro do laço, começaremos ligando o nosse led (colocando ele no estado 1) para isso vamos chamar a função led.value( ). Por conseguinte, vamos aguardar um segundo, desligar o led por mais um segundo e isso se repetirá até que haja uma interferência:

```
7  while True:
8      led.value(1) #liga o led
9      sleep(1) #aguarda 1 segundo
10     led.value(0) #desliga o led
11     sleep(1)
```

Código 2 – Entrada Digital

Realizar a montagem abaixo:



No segundo código, aprenderemos a configurar o pino como uma entrada digital, que é muito similar ao que já vimos, apenas com algumas mudanças.

```
#Aula 2: Sinal Digital

#Módulos Necessários
from machine import Pin           #acessar os pinos da placa
from time import sleep  #intervalos de temporização

#Configurando o pino como saída digital
led = Pin(2, Pin.OUT)    #Associar o pino 2 da placa a variável led
                         #e identificá-lo como um pino digital

#Configurando o pino como entrada digital
botao = Pin(4, Pin.IN) #mapea para pino 3 e configura como entrada

estado = 0

while True: #laço de repetição infinito (loop)

    if botao.value():         #Se houver alteração no valor do botão,
                              #ou seja, se o botão for pressionado
        estado = not estado   #troca o estado

    led.value(estado)        #led se comporta de acordo com a variável estado
    print(estado)
    sleep(1)
```

Copie o código acime. Execute-o em sua placa

Explicação

Aqui, importaremos os mesmos módulos usados no exemplo anterior e configuraremos um pino como uma saída digital:

```
4  from machine import Pin           #acessar os pinos da placa
5  from time import sleep  #intervalos de temporização
6
7  #Configurando o pino como saída digital
8  led = Pin(2, Pin.OUT)    #Associar o pino 2 da placa a variável led
9                           #e identificá-lo como um pino digital
```

Agora, vamos configurar o pino como uma entrada digital. Repetiremos o mesmo procedimento seguido para uma saída digital, a única diferença é que usaremos o IN invés de OUT:

```
12 botao = Pin(4, Pin.IN) #mapea para pino 3 e configura como entrada
```

Em sequência, vamos criar uma variável chamada estado que será iniciada com o valor 0:

```
14 estado = 0
```

Por fim, criaremos o nosso loop. Dentro do laço, utilizaremos a condição if, onde caso o botão seja pressionado, o valor armazenado na variável estado é trocado e, consequentemente, o estado do led também é trocado:

```
16 while True: #laço de repetição infinito (loop)
17
18     if botao.value():         #Se houver alteração no valor do botão,
19                               #ou seja, se o botão for pressionado
20         estado = not estado   #troca o estado
21
22     led.value(estado)        #led se comporta de acordo com a variável estado
23     print(estado)
24     sleep(1)
```

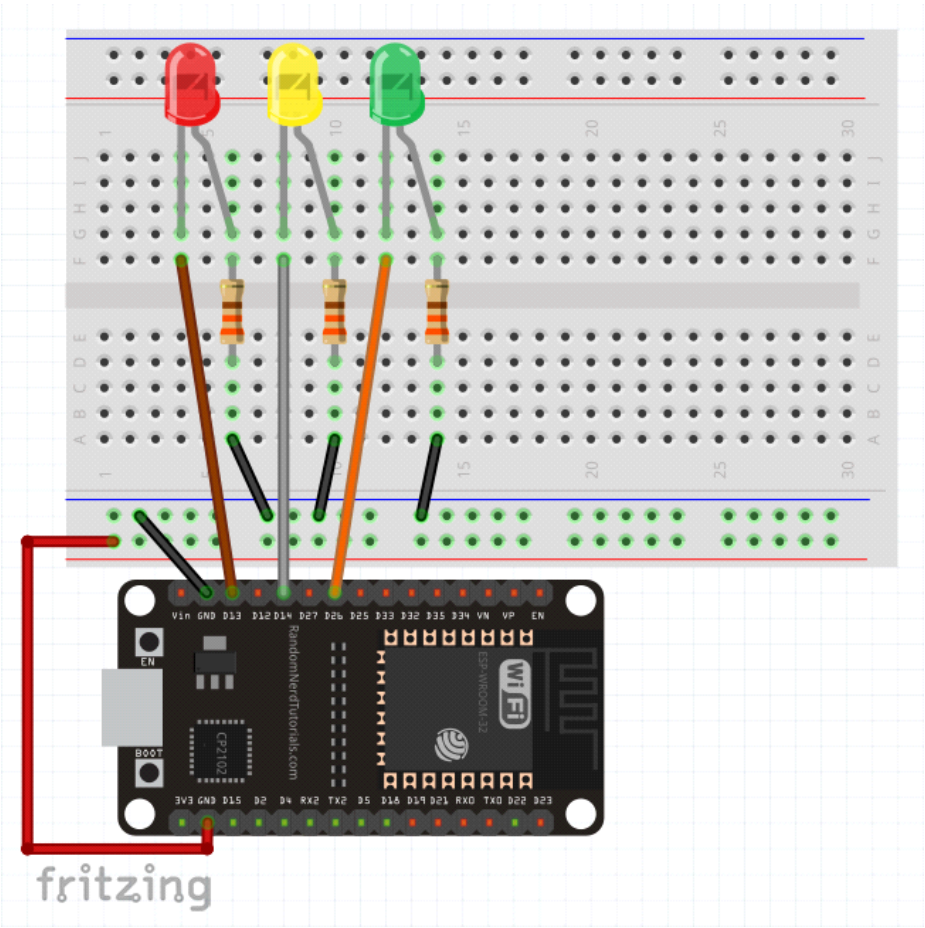
Exercícios

- Ao pressionar um botão, faça um led piscar. Para esse exercício, você vai utilizar a montagem do exemplo 2.

  Resolução:

```python
#Aula 2: Sinal Digital

#Módulos Necessários
from machine import Pin #acessar os pinos da placa
from time import sleep  #intervalos de temporização

#Configurando o pino como saída digital
led = Pin(2, Pin.OUT)    #Associar o pino 2 da placa a variável led
                         #e identificá-lo como um pino digital

#Configurando o pino como entrada digital
botao = Pin(4, Pin.IN) #mapea para pino 3 e configura como entrada

estado = 0
while True: #laço de repetição infinito (loop)

    if botao.value() == 1: #se o botão estiver presionado
        led.value(1)
        sleep(0.2)
        led.value(0)
        sleep(0.2)

    else:
        led.value(0)
```

2) Utilizando três LEDs, vamos simular um semáforo de trânsito três leds (verde, vermelho e amarelo). Siga o seguinte algoritmo:

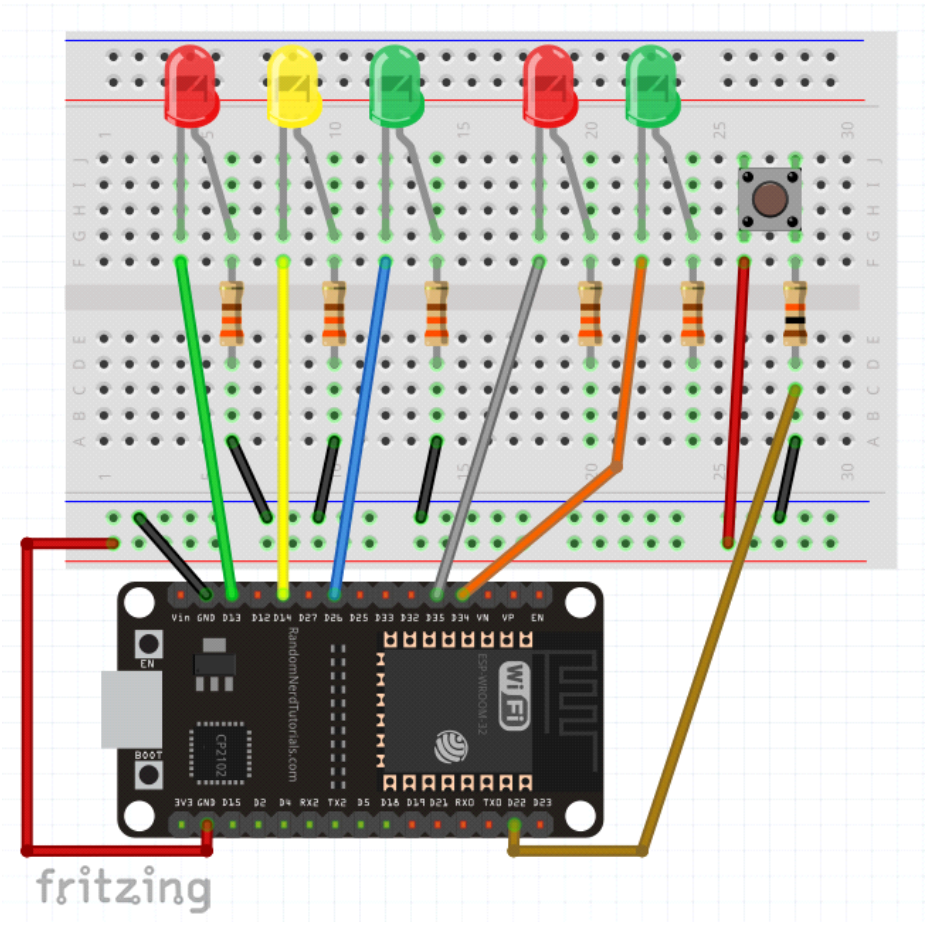


- LED verde aceso por 5 segundos, restante apagado;

- LED verde apaga;
- LED amarelo acende por 2 segundos, LED vermelho continua apagado;
- LED amarelo apaga;
- LED vermelho acende por 5 segundos, LED verde continua apagado;
- LED vermelho apaga, LED verde acende por 5 segundos, LED amarelo continua apagado;
- Repetir sequência

DESAFIO

3) Simule um semáforo de trânsito completo, para isso vamos utilizar 5 LEDs (2 verdes, 2 vermelhos 1 amarelo) e um botão. Siga o algoritmo a seguir:



- LED vermelho pedestre e LED verde carro começam acesos. Demais apagados;
- Caso o botão seja pressionado:
    - LED verde carro apaga, LED vermelho pedestre continua aceso;

        - LED amarelo pisca em 2 segundos, em um intervalo de 0,5 segundos;
        - LED amarelo e vermelho pedestre apagam;
        - LED verde pedestre e LED vermelho carro acendem por 5 segundos;
        - LED verde pedestre e LED vermelho carro apagam;
        - LED verde carro e LED vermelho pedestre acendem;
- Caso o botão na seja pressionado, não ocorre nada;