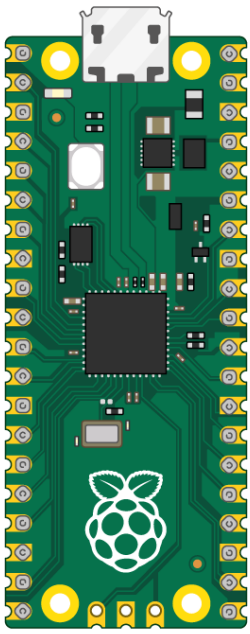
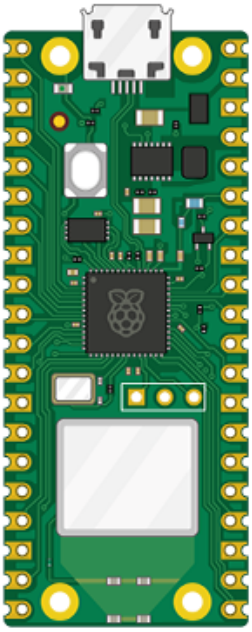
Eletrônica & programação para Raspberry Pi Pico & Pico W





Engenharia entendida

Por Eng. Dhanuzio A. A.

INICIANTE

Módulo 1 - Introdução ao Raspberry Pi Pico

Módulo 2 - Projetos com Pi Pico e W

Módulo 3 – Projetos intermediários com pico

Aula 01 - O que é o Raspberry Pi Pico?

Raspberry Pi Pico é um microcontrolador de baixo custo e de alto desempenho desenvolvido pela Raspberry Foundation. Ele foi lançado em janeiro de 2021 e é baseado no chip RP2040, que foi projetado pela própria Raspberry. A versão com Wi-fi e bluetooth foi lançada logo depois, que é a Raspberry Pi Pico W.

As placas são programadas com a linguagem de programação MicroPython, que é uma versão simplificada da linguagem do Python, ou usando linguagens de programação mais avançadas, como C ou C++. Ele também pode ser programado usando o software oficial de desenvolvimento da Raspberry Pi, chamado Raspberry Pi Pico C/C++ SDK, e pelo programa Thonny IDE, muito simples e fácil de usar.

Se trata um dispositivo muito versátil que pode ser usado para uma ampla gama de projetos de eletrônica, como controle de motores, sensores, dispositivos de Internet das Coisas (IoT), robótica, e muitos outros. Ele tem 26 pinos de entrada/saída (GPIO) que podem ser programados para executar uma variedade de funções e serão explicados detalhadamente mais à frente. O Pico W, que é uma versão com Wi-Fi e Bluetooth 5.2, pode ser configurado para usar os perfis Bluetooth Classic e Bluetooth LE (Low Energy). Possui as mesmas funções e pinos, com algumas mínimas diferenças. A presença do Wi-Fi o torna uma opção interessante para projetos de IoT devido à sua capacidade de processamento e baixo custo. Além disso, as linguagens, MicroPython, C e C++, facilita sua integração com outros dispositivos e sistemas. Na Figura 1 temos as duas placas.

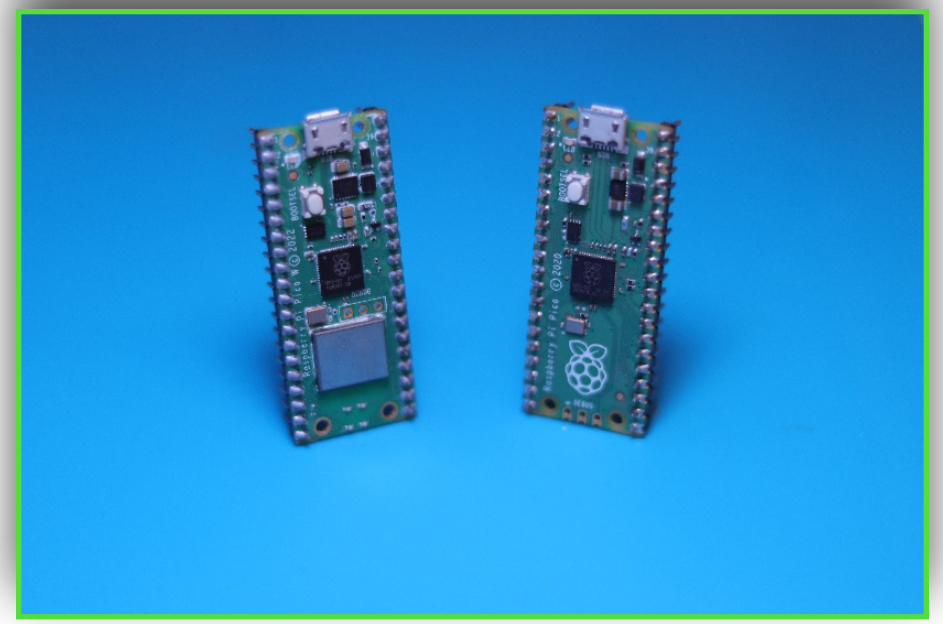


Figura 1: Raspberry Pi Pico W e Raspberry Pi Pico.

Com o Raspberry Pi Pico W, é possível criar projetos IoT que envolvam desde a coleta de dados de sensores até ações automatizadas, como ligar ou desligar dispositivos de acordo com determinadas condições. Por exemplo, é possível criar sistemas de monitoramento de temperatura em um ambiente, que envie notificações para o celular do usuário caso a temperatura ultrapasse um limite pré-determinado.

Aula 02 - Conhecendo o hardware

Para iniciamos na programação da placa, precisamos entender cada detalhe do seu hardware. Vamos iniciar a explicação detalhada dos pinos, processador, memória, conexão etc. Na Figura 2 temos alguns componentes, pinos e conexões importantes, na qual iremos explanar cada uma delas.

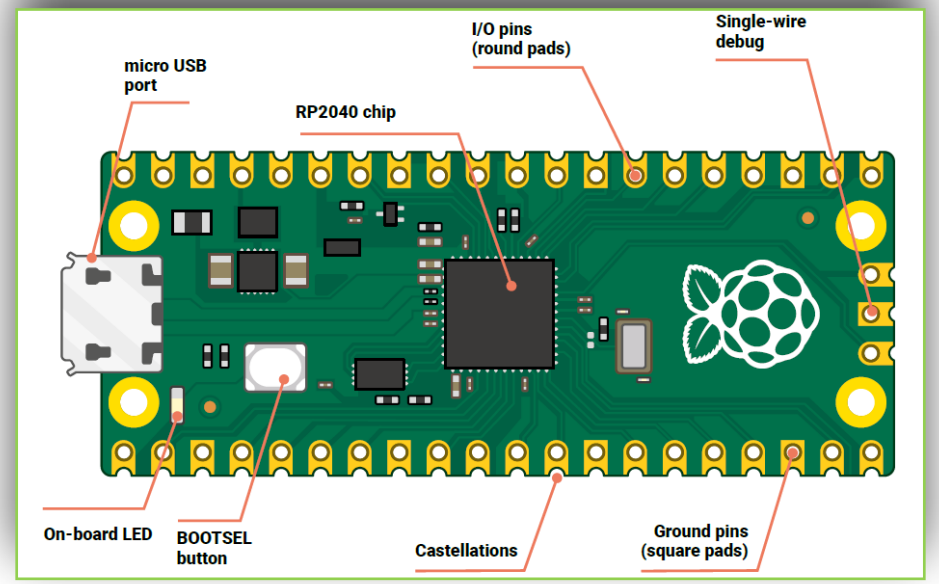


Figura 2: componentes, pinos e conexões importantes.

Micro USB port: Conexão usb para realizar a comunicação e receber o código que foi compilado no IDE e alimentação.
RP2040: Microcontrolador utilizado na placa. O RP2040 é o primeiro microcontrolador da Raspberry Pi, ARM Cortex-M0+ de 32 bits de duplo núcleo com Clock de até 133 MHz, 264 KB de RAM e até 16 MB de memória flash. Além disso, ele tem uma grande variedade de periféricos integrados, como 30 pinos de entrada/saída (GPIO), interfaces de comunicação serial (SPI, I2C, UART), PWM, ADC etc. Temos as especificações:

- RP: Raspberry Pi.
- 2: Número de núcleos no processador presente no microcontrolador.
- 0: Indica o tipo do núcleo do processador, no caso, Cortex-M0+.
- 4: Indica quanto de memória RAM o chip possui, baseado na fórmula matemática especial $\log_2\left(\frac{\mathrm{RAM}}{16}\right)$, logo, 4 significa que temos 264kB de RAM.

0: Indica quanto de memória não volátil (ROM) o chip possui, baseado na fórmula $\log_2\left(\frac{\text{ROM}}{16}\right)$, 0 significa que o chip não tem armazenamento não volátil.

I/O Pins: Pinos de entrada e saída, que serão explorados mais a frente juntamente com os demais pinos.

Single wire debug: A depuração de fio único (Single Wire Debug - SWD) é um protocolo de depuração de hardware comumente usados em microcontroladores e outros dispositivos de sistema embarcado. Ele permite que um depurador externo se comunique com o microcontrolador de destino por meio de um único fio, que geralmente é compartilhado com outros sinais do sistema.

BOOTSEL button: BOOTSEL é a abreviação de "seleção de inicialização", que alterna seu Pico entre dois modos de inicialização, usaremos ele mais à frente.

Castellations: São os pequenos terminais em forma de dentes de serra localizados na borda de um chip ou placa de circuito impresso (PCB). Eles são usados para fornecer uma conexão elétrica entre o chip ou PCB e outras partes do sistema.

Ground pins: Em eletrônica, GND é um ponto de referência de potencial elétrico comum para todos os componentes em um circuito.

Na placa Raspberry Pi Pico W temos o módulo Wi-Fi Infineon CYW43439, que oferece conectividade sem fio (Wi-Fi 4, 802.11n) ao microcontrolador com antena embutida para o Wi-Fi, integrada diretamente à placa além de um chip de Gerenciamento de Energia que também controla o módulo Wi-Fi de maneira eficiente. A seguir, na Figura 3, temos o esquemático disponibilizado pela Raspberry Foundation.

Aula 03 - Desvendando os pinos do Pi Pico

O Raspberry Pi Pico tem 40 pinos (GPIOs) no total, dos quais 26 estão disponíveis para o usuário. Na Figura 4 temos todas as conexões da placa Raspberry Pi Pico e na Figura 5 as conexões da Raspberry Pi Pico W, disponibilizado pela própria Raspberry. A seguir teremos a análise de cada pino de forma detalhada e objetiva. Como podemos notar, a principal diferença entre o Raspberry Pi Pico e o Pico W é a conectividade Wi-Fi integrada, além disso, o Raspberry Pi Pico W possui um conjunto ligeiramente diferente de pinos, como por exemplo, a conexão com o LED onboard, que no Pi Pico é conectado no pino GP25 e no Pi Pico W está no WL_GPOI0. As duas placas são bastante semelhantes. Ambas são baseadas no microcontrolador RP2040 e possuem 264 KB de RAM e 2 MB de memória flash, bem como uma variedade de periféricos integrados, como UART, I2C e SPI, como já vimos acima. Então, vamos explorar os pinos dessas placas.

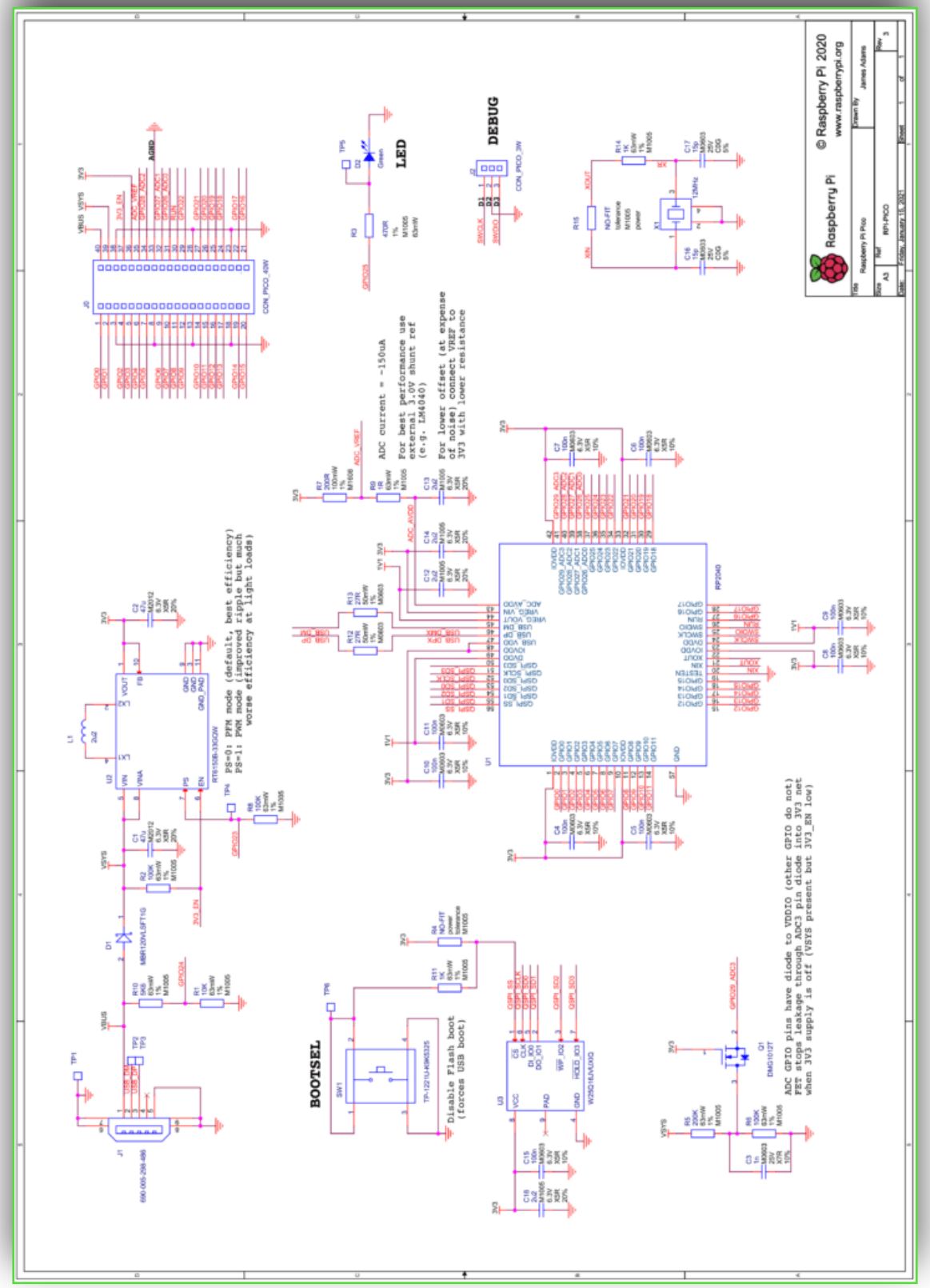


Figura 3: Esquemático Raspberry Pi Pico.

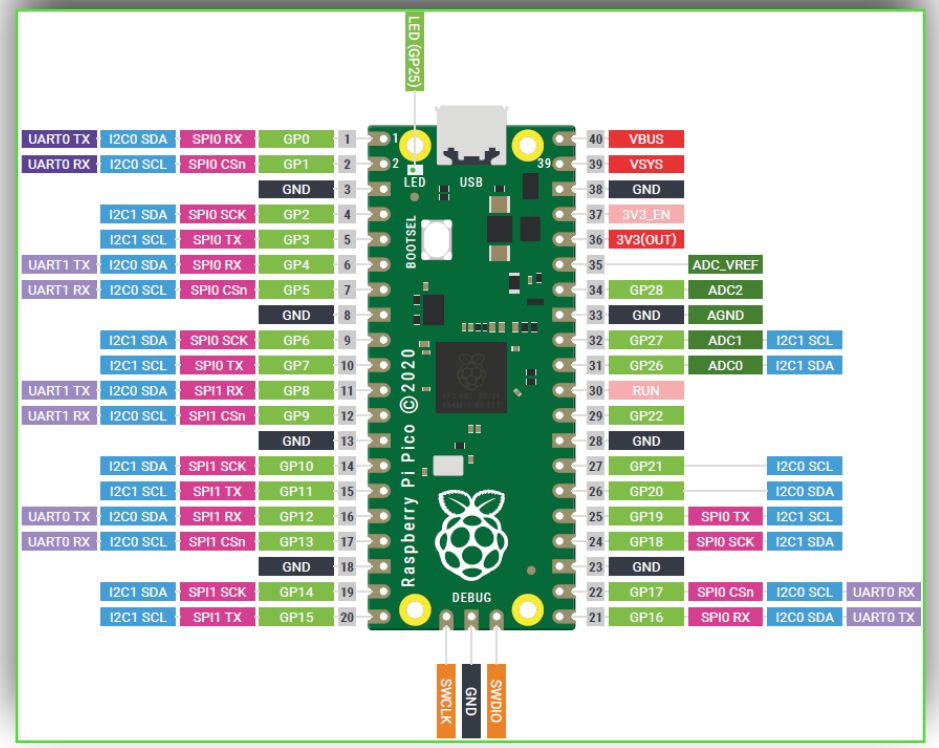


Figura 4: Todas as conexões da placa Pi Pico.

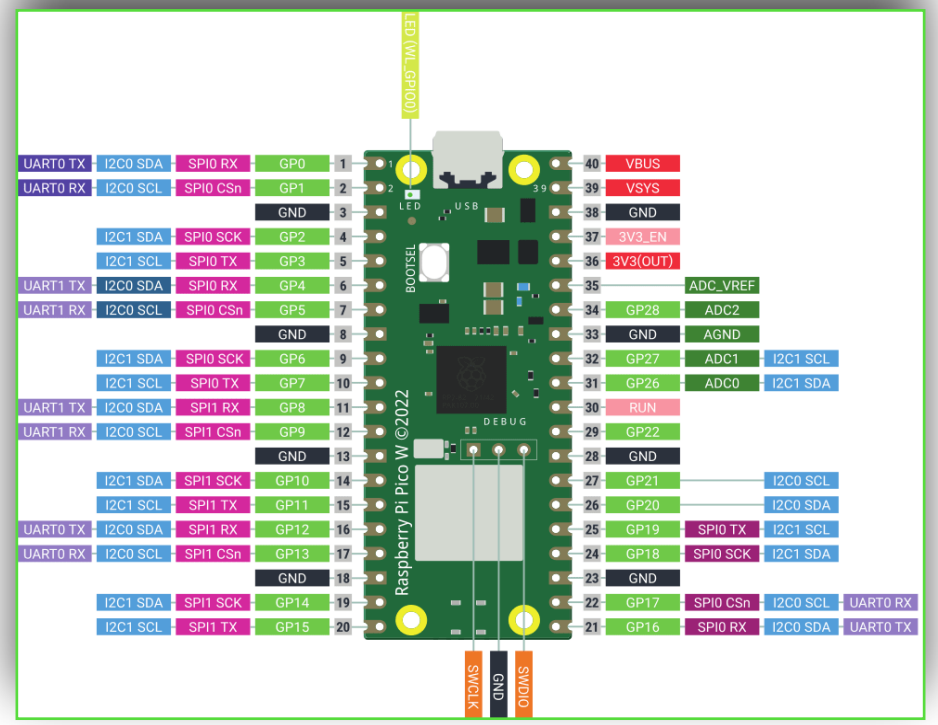


Figura 5: Todas as conexões da placa Pi Pico W.

VBUS: É a tensão de entrada da conexão micro-USB, nominalmente 5V ou 0V se o USB não estiver conectado.

VSYS: É a tensão de alimentação principal da placa, que pode variar de 1.8V a 5.5V, graças ao conversor DC-DC na placa gerando 3.3V para RP2040 e os GPIOs.

3V3_EN: Este pino é responsável por habilitar o conversor DC-DC onboard e consequentemente, a alimentação da placa. Está em nível alto por um resistor de pull-up de 100 kΩ. Para desativá-lo, basta enviar nível baixo (0V) para esse pino.
3V3(OUT): É a saída de 3.3V do RP2040 e suas entradas e saídas, gerada pelo conversor DC-DC onboard. Pode ser usado para alimentar circuitos externos, recomenda-se manter a carga neste pino inferior a 300mA.
ADC_VREF: É a tensão de referência do conversor ADC vindo da fonte de alimentação de 3.3V. Este pino pode ser usado com uma referência externa caso seja necessária uma referência diferente.
AGND: É a referência para do GPIO26 ao 29, que tem o ADC0, ADC1 E ADC2, na qual poderá ser usado para uma tensão de referência do conversor ADC diferente do GND padrão. Caso o uso do ADC não seja tão crítico, ele poderá ser conectado ao GND normalmente.
RUN: É o pino que habilita o RP2040. Ele possui um resistor pull-up de 50 kΩ interno conectado nos 3.3V, pois ele é habilitado em nível alto. Para redefinir ou desligar o RP2040, basta colocá-lo em nível baixo(0V).
GP0 à GP26: Os GPIOs (General Purpose Input Output) são pinos de entrada e saída para uso geral da placa. Por padrão e segurança, ao iniciar a placa, eles estarão em nível baixo (0V).
I2C0 E I2C1: No Raspberry Pi Pico, você pode usar a comunicação com dispositivos via I2C. A comunicação I2C (Inter-Integrated Circuit) é um protocolo de comunicação serial síncrono que permite a comunicação entre dispositivos usando apenas dois fios: SDA (Serial Data) e SCL (Serial Clock).
SPI0 E SPI1: A comunicação SPI (Serial Peripheral Interface) é um protocolo de comunicação serial síncrono que permite a comunicação entre dispositivos usando quatro fios: MOSI (Master Output Slave Input), MISO (Master Input Slave Output), SCK (Serial Clock) e SS (Slave Select).
UART0 E UART1: O RP2040 inclui dois UART (universal asynchronous receiver-transmitter) ou receptores-transmissores assíncronos universais em português. Trata-se de um protocolo de comunicação serial assíncrono que permite a comunicação entre dispositivos usando dois fios: TX (transmissão) e RX (recepção).

Há muitos recursos disponíveis para começar a programar o Raspberry Pi Pico e criar projeto com essa placa. Temos a documentação oficial da Raspberry Foundation para nos ajudar com algumas dúvidas e truques, além de ser importante um conhecimento básico de programação e eletrônica antes de começar a trabalhar com o Raspberry Pi Pico. Mas não se preocupe, vamos iniciar nossos estudos desde o básico da eletrônica e da programação no decorrer desse livro. Para programar o Raspberry Pi Pico, precisamos usar a linguagem de programação C ou MicroPython e uma IDE (integrated development environment) ou em outras palavras, ambiente de

desenvolvimento integrado, como o Thonny IDE, Visual Studio Code, Pycharm ou qualquer outro editor de código de sua escolha. Vamos trabalhar com MicroPython e usaremos o Thonny IDE a princípio. Vamos nessa!!!

Aula 04 - O que é MicroPython?

MicroPython é uma implementação da linguagem de programação Python 3 que é otimizada para ser executada em microcontroladores e sistemas embarcados com recursos limitados de memória e processamento. O MicroPython permite que os desenvolvedores escrevam códigos em Python para controlar dispositivos eletrônicos, como sensores, motores e atuadores. Temos então, uma alternativa mais leve e eficiente em termos de recursos completo para sistemas embarcados, extremamente eficiente e adequado para projetos de Internet das Coisas (IoT- Internet of Things) e outras aplicações de eletrônica e automação.

É uma linguagem compatível com ampla variedade de microcontroladores e placas, que além do Raspberry Pi Pico e Pico W, a popular placa ESP32. Podemos também usar a linguagem C para programar a placa, porém, vamos usar o MicroPython pela simplicidade e facilidade de utilização, facilitando nossa compreensão do código. Vamos analisar o código mostrado na Figura 6, observe que temos o famoso pisca led para o Raspberry Pi Pico, não se preocupe ainda em entender o código, vamos explicar tudo isso mais à frente. A esquerda temos o código em linguagem C, na direita o código em linguagem MicroPython, observe que precisamos de menos bibliotecas, menos linhas, e não precisamos de ponto e vírgula, temos expressões mais simples, a ponto de, se lermos um pouco atentamente as linhas do código, notemos que ele é autoexplicativo!

```c
#include <stdio.h>
#include "pico/stdlib.h"
#include "pico/time.h"

#define LED_PIN 25

int main() {
    stdio_init_all();
    gpio_init(LED_PIN);
    gpio_set_dir(LED_PIN, GPIO_OUT);

    while (true) {
        gpio_put(LED_PIN, 1);
        sleep_ms(500);
        gpio_put(LED_PIN, 0);
        sleep_ms(500);
    }

    return 0;
}
```

```python
import machine
import utime

LED_PIN = 25

led = machine.Pin(LED_PIN, machine.Pin.OUT)

while True:
    led.value(1)
    utime.sleep(0.5)
    led.value(0)
    utime.sleep(0.5)
```

Figura 6: Comparação do pisca led escrito em C e em MicroPython.

Logo, MicroPython e C são duas linguagens de programação diferentes que podem ser usadas para programar o Raspberry Pi Pico. C é uma linguagem de

baixo nível e mais complexa, que oferece maior controle e desempenho em operações de baixo nível. Por outro lado, o MicroPython é uma linguagem de alto nível baseada em Python que é mais fácil de aprender e usar, além de ter um processo de desenvolvimento mais rápido e flexível.

Se estamos trabalhando em um projeto que envolve operações de baixo nível e queremos um alto desempenho, a linguagem C pode ser a melhor opção. Por exemplo, se estamos desenvolvendo drivers de dispositivos ou precisamos de um controle preciso do hardware, a linguagem C pode ser mais adequada. Por outro lado, se estamos desenvolvendo projetos que envolvem programação de alto nível ou de aplicativos de IoT, o MicroPython pode ser uma escolha melhor. Se você é novo em programação, o MicroPython pode ser mais fácil de aprender do que a linguagem C. Em resumo, a escolha entre MicroPython e C depende do tipo de projeto que você está trabalhando, de suas habilidades e preferências pessoais. Ambas as linguagens têm seus próprios benefícios e desvantagens, e é importante escolher a que melhor se adapta às suas necessidades de desenvolvimento.

Aula 05 - Instalando o Thonny IDE

Thonny é uma IDE para a linguagem de programação Python. É uma ferramenta de código aberto e gratuita que oferece uma interface gráfica amigável para ajudar os programadores a escrever, depurar e executar código para microcontroladores como Raspberry Pi Pico, Raspberry Pi Pico W, Esp32, entre outros. Para instalar a IDE, basta acessarmos o site oficial em https://thonny.org e encontraremos no site, a área de download mostrado na Figura 7.

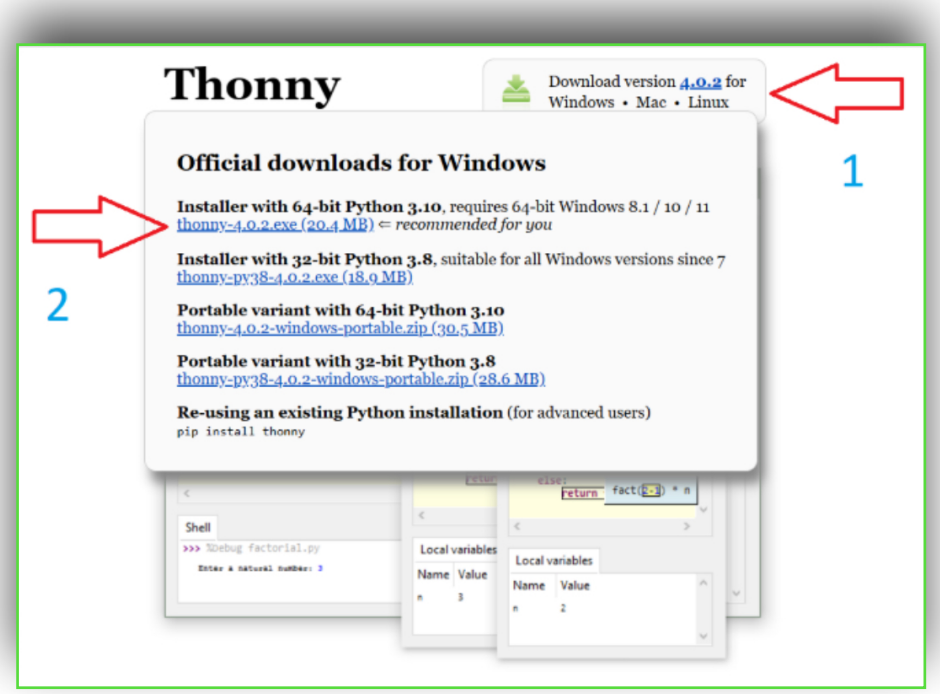
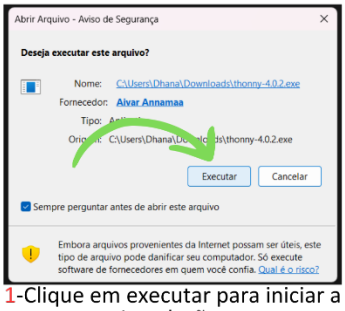
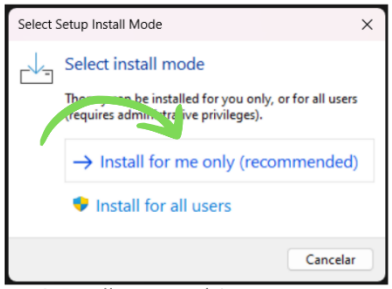


Figura 7: Site oficial do Thonny IDE.

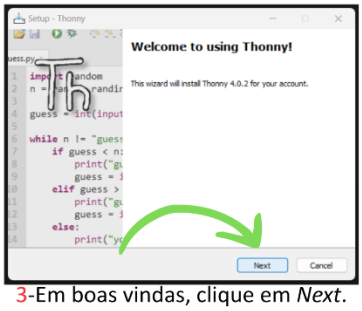
Em Download version, teremos a versão, que até a data deste livro, está na 4.1.4 e as versões para cada sistema operacional. Escolha seu sistema e baixe o instalador. Vamos seguir as etapas de instalação:



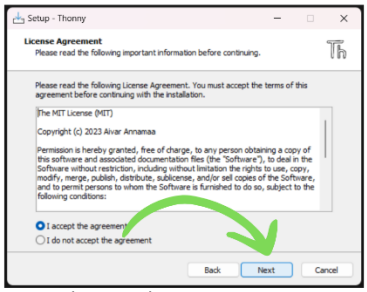
1-Clique em executar para iniciar a instalação.



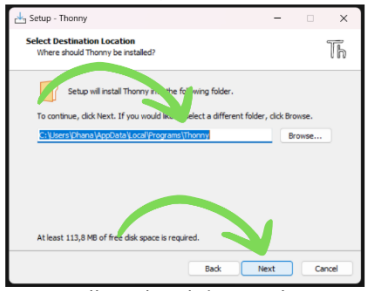
2-Escolha os usuários para o seu computador.



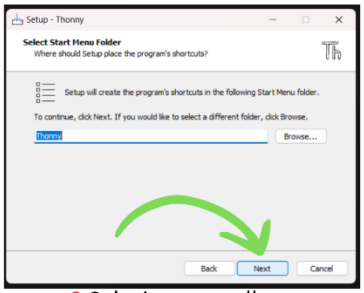
3-Em boas vindas, clique em *Next*.



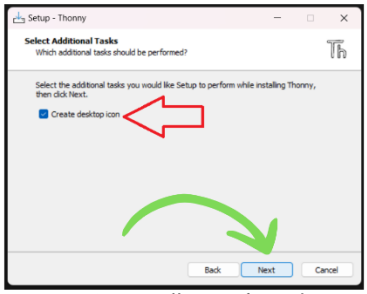
4-Se de acordo, aceite os termos.



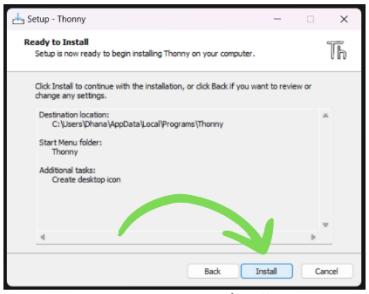
5-Escolha o local de instalação.



6-Selecione o atalho.



7-Crie o atalho na área de trabalho.



8-Inicie a instalação.

9-Clique em *Finish* para finalizar.

Com a IDE instalada, vamos clicar no atalho e iniciar na programação em MicroPython. Na Figura 8 temos a tela inicial do Thonny IDE.

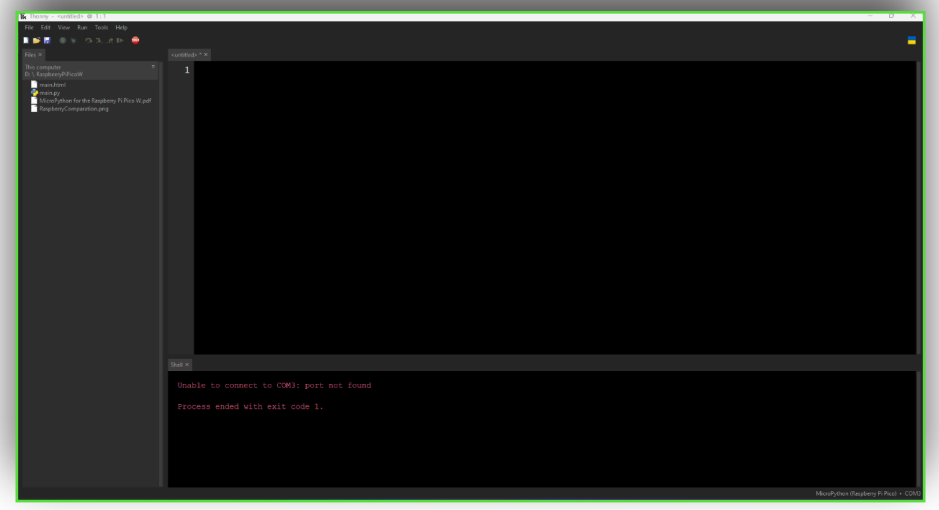


Figura 8: Tela inicial do Thonny IDE.

O Thonny oferece várias funcionalidades úteis para programadores iniciantes e experientes, incluindo destaque de sintaxe, depuração interativa, visualização de valores de variáveis, complemento automático de código e um ambiente de execução integrado. Além disso, o Thonny possui um recurso exclusivo chamado "modo de depuração", que ajuda os iniciantes a entender melhor como o código funciona, fornecendo explicações passo a passo do código enquanto é executado.

Aula 06 - Configurando a placa

Para o nosso primeiro código, devemos entender como funciona o compilador que acabamos de instalar, como conectar e gravar a placa. Vale lembrar que não vamos explicar como funciona o compilador e o código de forma separada, vamos mostrar de forma simultânea tudo que devemos saber para entender as funcionalidades, código, eletrônica e montagem no decorrer desse livro.

Para nossa primeira configuração, precisamos instalar o firmware do MicroPython na placa. O firmware do Raspberry Pi Pico é um conjunto de instruções que é gravado na memória flash do microcontrolador da placa, e é responsável por controlar o hardware, permitindo que ele execute as tarefas necessárias. O firmware do Raspberry Pi Pico é responsável por gerenciar a execução dos programas que são carregados na placa. Ele fornece uma API (Application Programming Interface) para a programação de periféricos e comunicação com outros dispositivos, além de incluir drivers para os principais componentes da placa, como os pinos GPIO, as interfaces de comunicação e USB.

O firmware do Raspberry Pi Pico é atualizável, o que significa que a Raspberry Pi Foundation pode corrigir bugs, adicionar novos recursos e melhorar o desempenho ao longo do tempo. Além disso, os usuários avançados podem modificar o firmware para personalizar o comportamento do Pico de acordo com suas necessidades específicas. Vamos iniciar: Pressione e seguro o botão BOOTSEL, conecte o cabo usb na placa no seu computador. Com o Thonny IDE aberto, irá abrir dois arquivos localizados dentro da placa. Um irá levar ao site com a documentação oficial da placa e o segundo arquivo mostra os dados da placa, como modelo e Bootloader, como mostra a Figura 9.

Para instalar o firmware, vá no canto inferior direito, e você terá a opção de instalar, como mostra a Figura 10. Na janela que irá abrir, seleciona o volume RPI-RP2, que estará, provavelmente, selecionado. Em Variação do MicroPython, selecione Raspberry Pi Pico/Pico H. Para finalizar, selecione a última versão, que até essa publicação, é a 1.2. No canto inferior direito escolha a placa Raspberry Pi Pico, como visto na Figura 11. Tudo pronto para iniciarmos.

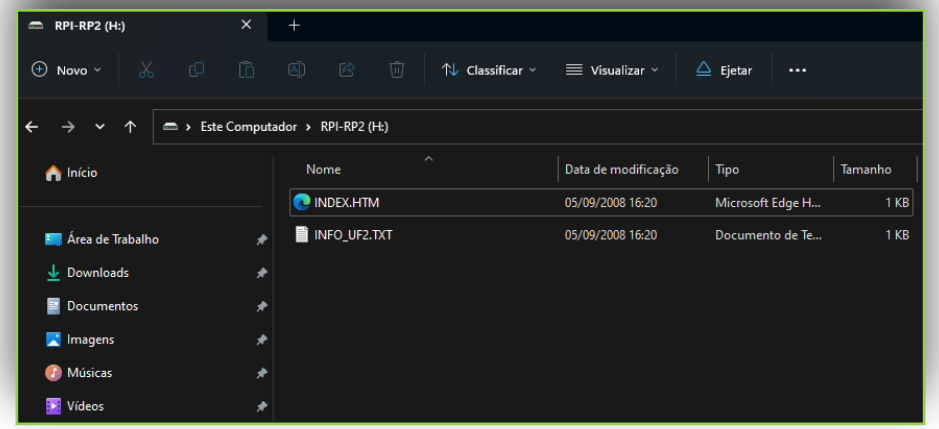


Figura 9: Arquivos presente na placa Raspberry Pi Pico.

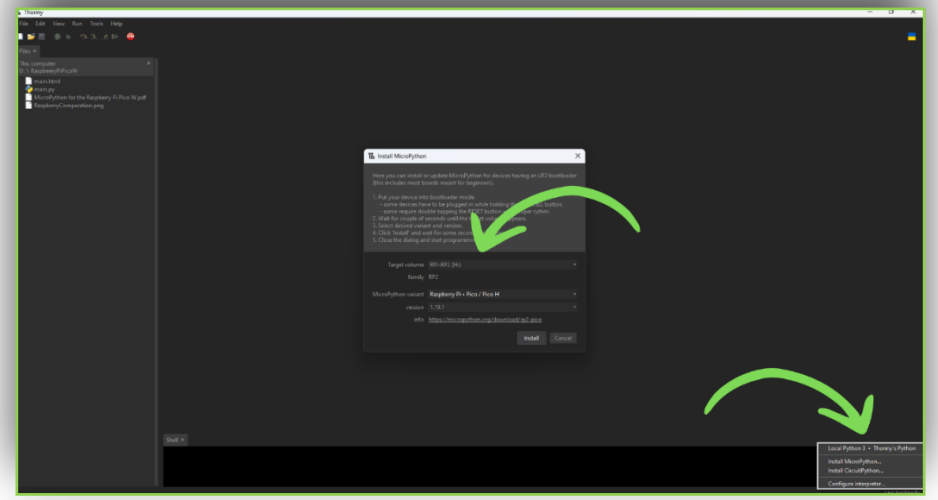


Figura 10: Passos para instalar o MicroPython na placa.

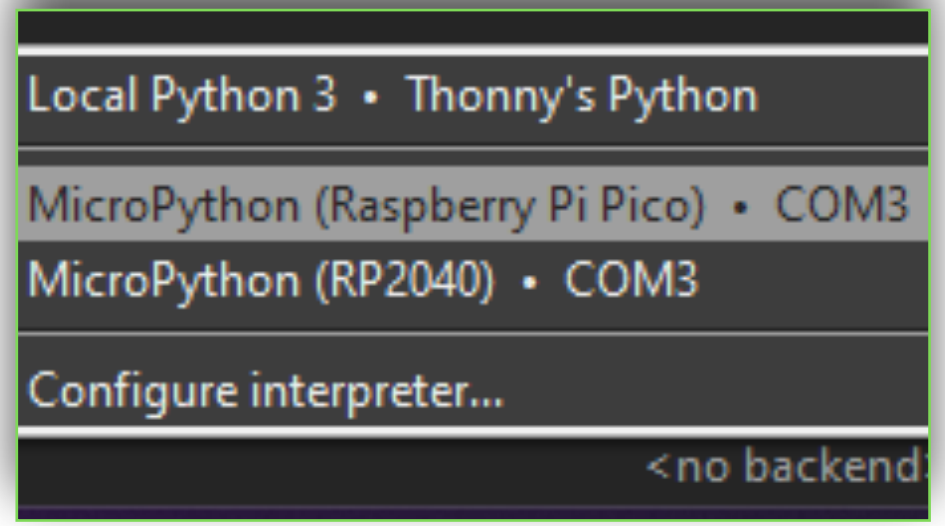


Figura 11: A placa como opção para intérprete da IDE.

Aula 07 - Hello World com Pi Pico

Vamos iniciar o nosso primeiro código, o famoso Hello World, ou, para os microcontroladores, o pisca led. Com o Thonny IDE aberto, clique em File e New no canto superior esquerdo. Com a placa conectada e selecionada no

intérprete, como foi visto na Figura 11, podemos notar na lateral esquerda, a aba arquivos na qual mostra que não temos nada gravado na placa. Caso não consiga visualizar a aba, vá na barra de menus do Thonny IDE, clique em View e Files, como visto na Figura 12.

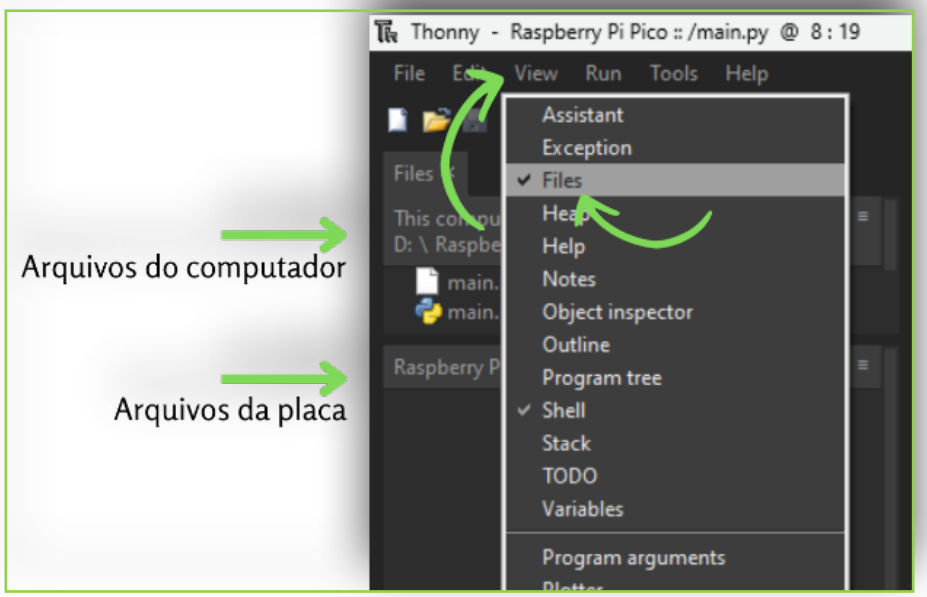


Figura 12: Visualização dos arquivos presentes na placa.

Note que nossa IDE está totalmente em inglês, sim, é importante ter familiaridade com o idioma para programar. Isso irá facilitar o entendimento do código e trará novos caminhos para sua vida profissional. Por isso, um conselho particular: Inglês hoje é fundamental, principalmente para a área da tecnologia, por isso, tenha o máximo contato com o idioma. Vamos digitar nosso primeiro código:

```
import machine
import utime
led=machine.Pin(25,machine.Pin.OUT)
while(True):
    led.value(1)
    utime.sleep(1)
    led.value(0)
    utime.sleep(1)
```

Não se preocupe, vamos explicar linha a linha. Após isso, vamos na barra de menus, no canto superior esquerdo, clique em File e Save as..., veremos uma tela como a vista na Figura 13, na qual iremos selecionar onde salvar e executar o código. Na opção This Computer usaremos o computador para executar o código, porém, não é o que queremos. O "computador" que vamos usar é a nossa placa. Por isso, e obviamente, precisamos enviar o código para ela. Na próxima tela, salve o arquivo com o nome Main.py, isso fará com que a placa entenda que esse é o código principal (Main). Note na lateral

esquerda, na aba Files, que agora temos o arquivo na placa. Clique em Run na barra de menus, e pronto, você verá o led onboard da placa piscando a cada 1 segundo. Lembre-se de fazer todos os procedimentos vistos até aqui, caso contrário, poderemos ter problemas e erros de salvamento e execução do código.

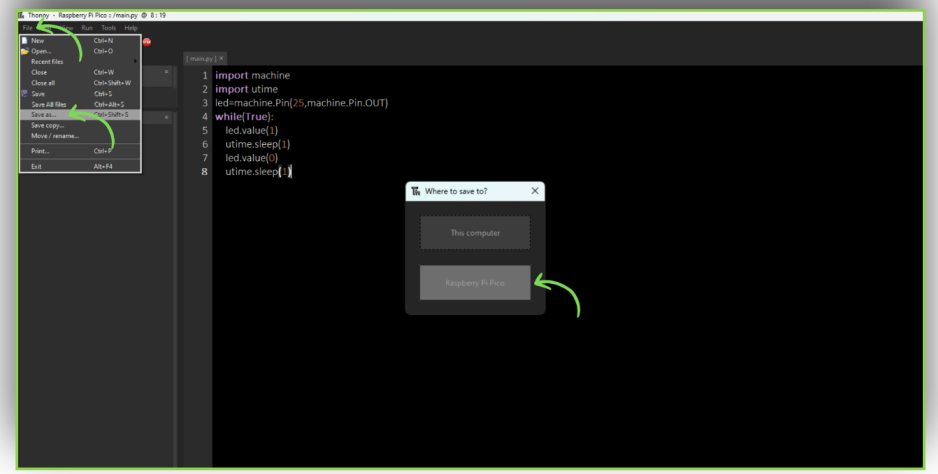


Figura 13: Salvando o código na placa Raspberry Pi Pico.

Aula 08 - Explicando o código

Para que possamos controlar nossa placa via programação, precisamos ter acesso ao hardware, como pinos GPIO, conversores analógico-digitais e temporizadores. Esse acesso nos permite controlar vários componentes conectados à placa, como LEDs, motores, sensores e telas. Podemos criar instâncias de várias classes fornecidas pelo módulo, como Pin, ADC, PWM, I2CI, SPI e UART, para interagir com o hardware. Para isso, precisamos de uma biblioteca que realize essa interação, temos então nossa primeira linha do código:

```
import machine
```

Aqui, estamos importando a biblioteca machine. Uma biblioteca em MicroPython é um conjunto de módulos ou arquivos de código que fornecem funções e classes que podem ser usadas para interagir com componentes ou dispositivos específicos em um microcontrolador ou outro sistema embarcado. Uma biblioteca pode conter uma ampla variedade de componentes, como funções matemáticas, algoritmos de ordenação, manipulação de arquivos, operações de rede, interfaces de usuário e muito mais. Elas podem ser escritas em diferentes linguagens de programação e são distribuídas em forma de arquivos ou pacotes que podem ser incluídos nos projetos.

Essas bibliotecas são geralmente criadas para simplificar o desenvolvimento de projetos em MicroPython, fornecendo uma camada de abstração sobre o hardware. As bibliotecas em MicroPython podem ser criadas por

desenvolvedores individuais ou por empresas que fabricam dispositivos embarcados. Vamos imaginar que estamos em uma cidade chamada MicroPython City. Nessa cidade, temos várias "casas" especiais, onde cada uma tem suas próprias regras e ferramentas.

Biblioteca: Imagine que você vai à Biblioteca Central da cidade. Lá, você encontra livros que ensinam a fazer diferentes tarefas, como cozinhar, construir, programar, etc. Quando você pega um livro, pode usar as instruções dele para realizar uma tarefa. Assim, uma biblioteca em MicroPython é um conjunto de códigos prontos que você pode usar para facilitar o seu trabalho.
Classe: Agora, pense em uma dessas casas na MicroPython City, que é uma fábrica de robôs. A classe é como o molde para criar diferentes robôs. Esse molde define como todos os robôs da fábrica serão construídos. Então, uma classe em MicroPython é um plano ou um modelo a partir do qual você cria objetos.
Atributos: Dentro da fábrica de robôs, cada robô tem características diferentes, como cor, tamanho, ou capacidade. Esses são os atributos. Em MicroPython, os atributos são as características ou informações que um objeto da classe pode ter, como o nome, idade, ou outras propriedades.
Métodos: Agora, pense nas coisas que os robôs podem fazer, como andar, falar ou dançar. Essas ações são os métodos. Em MicroPython, os métodos são as funções que pertencem a uma classe e definem o que os objetos dessa classe podem fazer.

Então, na MicroPython City, você pode ir à biblioteca pegar instruções prontas, criar objetos a partir de uma classe (como robôs), definir seus atributos (como cor e tamanho) e programar o que eles podem fazer usando métodos (como andar ou dançar).

```
import utime
```

Aqui importamos a biblioteca utime, que fornece funções para medir e gerenciar o tempo. Ele é usado para trabalhar com o tempo em microcontroladores e outros sistemas embarcados, onde o controle preciso do tempo é fundamental. Ela fornece funções para medir o tempo em microssegundos e milissegundos, bem como para atrasar a execução de um programa por um período específico.

Ela também inclui funções para obter a hora atual do relógio em segundos, permitindo que tenhamos o controle do tempo decorrido no nosso programa e outras funções interessantes. Como vamos precisar de um tempo em que o led ficará acesso e apagado, precisamos dela para configurar esse tempo de forma precisa. Vamos definir agora um nome para o pino que está conectado ao led na placa onboard, como foi visto, está no GPIO25. O nome que iremos usar, ou melhor dizendo, a variável, será led. Agora vamos dizer qual pino da

máquina estará conectado a variável led. Traduzindo isso em linhas de código:

```
led=machine.Pin(25).
```

Explicando mais uma vez para deixar bem claro: estamos dizendo que a variável led será o pino da placa GPIO25. Como sabemos, o Pi Pico tem vários pinos de entrada e saída, logo, precisamos informar se vamos usar esse pino como entrada ou saída. Traduzindo para MicroPython, temos machine.Pin.OUT. Estamos dizendo que esse pino da máquina, será usada como uma saída, tento em vista que vamos enviar um sinal para acender o led. Juntando tudo isso na mesma linha, temos a próxima linha de código:

```
led=machine.Pin(25,machine.Pin.OUT)
```

Com isso, finalizamos as configurações iniciais, essas linhas de código serão executadas apenas uma vez quando o microcontrolador for iniciado. Porém, queremos que nosso led fique piscando indefinidamente ou até ser interrompida a alimentação, ou melhor falando, em loop.

```
while(True):
```

A instrução while(True): cria um loop infinito que executa o bloco de código dentro dele repetidamente até que seja interrompido por uma instrução de break, uma exceção ou por interrupção via código ou via alimentação da placa. O True é uma constante booleana que sempre é avaliada como verdadeira. Portanto, o loop nunca é interrompido a menos que uma instrução de break seja usada explicitamente para sair dele. Guarde essa informação, vamos usar ela futuramente. Se você tem familiaridade com alguma outra linguagem de programação, como C, vai notar que no MicroPython, não vimos vírgulas ou parênteses. Isso porque o Python funciona por indentação. Em Python, a indentação é usada para indicar o bloco de código que deve ser executado junto. A indentação é uma forma de organização e estruturação do código e é feita usando espaços ou tabulações, e é importante que a indentação seja consistente em todo o código. Isso ajuda a tornar o código mais legível e fácil de entender. A maioria dos editores de texto modernos tem suporte para a indentação automática, tornando mais fácil para os programadores manterem-na consistente em todo o código. É importante notar que o Python usa a indentação para definir a estrutura do código, e não usa chaves ou palavras-chave como begin e end, como em outras linguagens de programação. Por exemplo, a estrutura de controle if e else em Python é escrita com a seguinte sintaxe:

```
if condição:
    #bloco de código a ser executado no if.
else:
    #bloco de código a ser executado no else.
```

Observe que o bloco azul abaixo do if e else é indentado com quatro espaços ou tab. Essa indentação indica que o código dentro desses blocos pertence ao respectivo bloco de controle de fluxo. Não se preocupe, veremos if e else em breve. É importante ter em mente que a indentação em Python é significativa. A falta de indentação adequada pode levar a erros de sintaxe e comportamentos inesperados do programa. Por isso as 4 linhas restantes estão com indentação, para configurá-las dentro do While.

O que precisamos fazer agora é mudar o estado do led, ligá-lo e desligá-lo. Vamos então definir o valor para ele na próxima linha:

```
    led.value(1)
```

Com isso, como as máquinas entendem apenas dois estados, ligado e desligado, 0 ou 1, estamos definindo o led com o valor 1, ligado. Na próxima etapa precisamos deixar o led 1 segundo ligado. Por isso, vamos então deixar nossa placa "dormindo" por 1 segundo com a linha de código:

```
    utime.sleep(1)
```

Como dito, a biblioteca utime nos permite trabalhar com delays no nosso código, precisamos apenas informar o tempo em segundos. Portando, com essa linha estamos usando a função de "dormir" por 1 segundo. Após 1 segundo, vamos desligar o led, com a mesma função na linha 5, porém vamos definir o led como desligado, ou seja, 0, com a linha de código:

```
    led.value(0)
```

E pronto, basta esperar 1 segundo novamente, como visto na linha 6, e ligar o led novamente, em loop. Assim chamamos mais uma vez a função utime.sleep(1), e encerramos o código e nossa análise. Acabamos de realizar a primeira programação da placa Raspberry Pi Pico.

A programação na placa Raspberry Pi Pico W é exatamente igual, com uma única diferença: para configurar o led onboard, como temos o módulo Wi-fi, há algumas mudanças simples do hardware, como no nosso caso. Logo, usamos o nome do pino como 'WL_GPIO0' e não 25 como vimos na placa Pi Pico. Lembrando que temos as aspas. Temos apenas que modificar a linha 3, sendo assim, o código para a Pi Pico W fica:

```
import machine
import utime
led=machine.Pin('WL_GPIO0',machine.Pin.OUT)
while(True):
    led.value(1)
    utime.sleep(1)
    led.value(0)
    utime.sleep(1)
```

Aula 09 - Semáforo

A placa Pi Pico é bem completa e simples de usar para alguns projetos, porém, bem limitada em relação a sensores e atuadores. Por exemplo, temos o sensor de temperatura no processador RP2040 que podemos usar em nossos códigos. Mas e se no projeto quiséssemos acionar uma ventilação de acordo com a temperatura do ambiente? A placa não tem ventoinha nem corrente suficiente nos pinos para acionar algumas cargas diretamente. A placa pode fornecer 3.3V com apenas 50mA no máximo nos seus pinos. Iremos então usar componentes externos para projetos daqui em diante. E não se preocupe, aqui você vai aprender eletrônica também. Nosso primeiro projeto será um semáforo para veículos, com as 3 cores: vermelho, amarelo e verde, que como sabemos, são as cores de indicação de um semáforo. Vamos usar 3 leds, um de cada cor. Mas afinal...

9.1 – O que são leds?

LED (Light Emitting Diode) ou diodo emissor de luz, são componentes eletrônicos que convertem energia elétrica diretamente em luz. Eles são usados em diversos dispositivos eletrônicos, como lâmpadas, painéis de TV, monitores de computador, displays, luzes de sinalização e iluminação de ambientes. Quando uma tensão é aplicada ao material semicondutor do LED, os elétrons do material adquirem energia e saltam para um nível de energia mais elevado. Quando esses elétrons retornam ao seu estado original, eles liberam a energia na forma de luz. A cor da luz emitida pelo LED depende do tipo de material semicondutor utilizado. Por exemplo, LEDs com material semicondutor de arseneto de gálio emitem luz na cor vermelha, enquanto os com material semicondutor de fosfeto de gálio emitem luz verde. Veja na Figura 14 suas características.

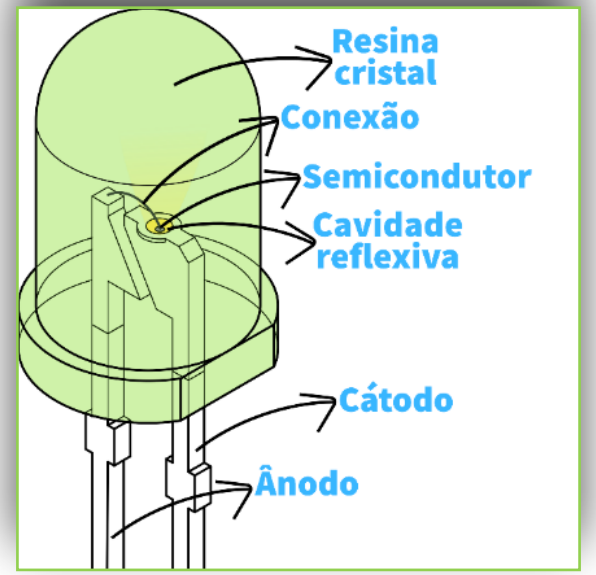


Figura 14: Led e suas partes internas.

Os LEDs são muito eficientes, pois consomem menos energia elétrica, são compactos e duram mais tempo que lâmpadas fluorescentes, por exemplo. São mais versáteis em termos de design, com grande variedade de formas e tamanhos para diferentes aplicações.

É importante observar a polaridade, conectando o terminal positivo do LED (ânodo) ao positivo da fonte e o negativo do LED (cátodo) ao negativo da fonte. Note que o terminal negativo do led tem um chanfro de indicação. Se a polaridade for invertida, ele não acenderá. Outro pronto importante é a tensão e a corrente que o led trabalha, pois, conectá-lo com polaridade invertida não irá queimá-lo, mas tensão e/ou corrente acima dos limites dele, sim. A Tabela 01 mostra as tensões de acordo com as cores usadas.

Cor	Tensão típica (V)	Corrente recomendada (mA)
Vermelho	1,8 - 2,2	20 - 30
Verde	2,0 - 3,4	20 - 30
Amarelo	2,0 - 2,2	20 - 30
Azul	2,8 - 3,4	20 - 30
Branco	3,0 - 3,4	20 - 30
Rosa	3,0 - 3,4	20 - 30
UV	3,0 - 3,4	20 - 30
Infravermelho	1,2 - 1,8	20 - 30

Tabela 01: Tensão e corrente para LEDs.

Para controle da corrente do LED, precisamos de um componente que controle o fluxo dela. Um componente que ofereça resistência para que a corrente não ultrapasse o limite de 10 a 30mA. Então vamos conhecer o resistor.

9.2 – O que são Resistores?

Resistores são componentes eletrônicos que têm como função principal de oferecer uma resistência à passagem de corrente elétrica. Usamos em circuitos eletrônicos para controlar o fluxo e limitar a quantidade de corrente que passa através de componentes. São construídos a partir de materiais que possuem alta resistividade elétrica, como carbono, cerâmica ou filme de metal. A unidade de resistência elétrica é o ohm (Ω). Com resistores podemos definir os valores de corrente que queremos em nossos projetos, no caso, a corrente para os leds do semáforo. Precisamos então relacionar a tensão e a corrente dos leds através do uso do resistor, mas

como juntar essas três coisas? A lei de Ohm é um princípio fundamental da eletricidade que estabelece a relação entre a tensão (V), a corrente elétrica (I) e a resistência elétrica (R) em um circuito elétrico. Segundo a lei de Ohm, a corrente elétrica que flui através de um material condutor é diretamente proporcional à tensão aplicada e inversamente proporcional à resistência elétrica do material. Matematicamente, a lei de Ohm pode ser expressa pela equação:

$$V = I*R$$

Onde V é a tensão aplicada em volts (V), I é a corrente elétrica em amperes (A) e R é a resistência elétrica em ohms (Ω). Essa relação simples é muito útil para calcular a corrente elétrica que passa através de um componente, a tensão que deve ser aplicada para produzir uma determinada corrente elétrica e a resistência elétrica de um material condutor.

Usaremos os 3.3V (Vfonte) da placa e queremos 2V(Vled) no led com uma corrente de 20mA, qual resistor precisamos para os 1.3V restantes sobre ele? Vamos então subtrair o valor que queremos no led do valor da fonte e dividirmos pela corrente, devemos transformar para ampere, logo, 20mA é o mesmo que 0.020A. Aplicando na fórmula e isolando o R, temos:

$$R = \frac{Vfonte - Vled}{I}$$

$$R = \frac{3.3 - 2}{0.020}$$

$$R = 65\Omega$$

Podemos conferir nossos cálculos no simulador mostrado na Figura 15.

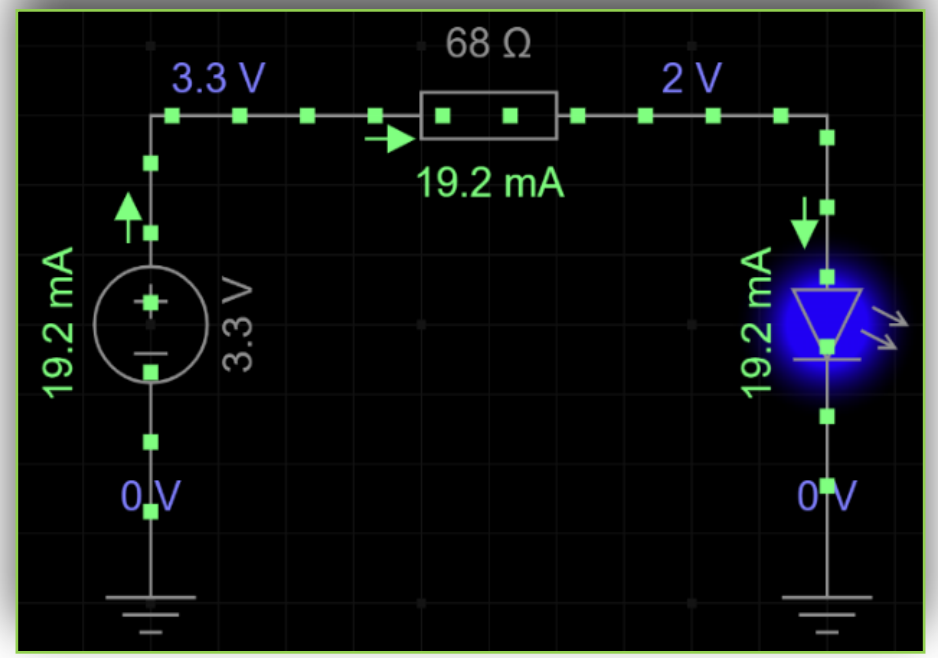


Figura 15: Simulação com valores no everycircuit.com.

Com um resistor de 65Ω teremos os valores desejados, porém, esse não é um valor de resistor comercial, vamos então utilizar o de 68Ω que dará uma corrente de 19.11mA. E como saber o valor de um resistor? Observe na Tabela 02 os valores de alguns resistores comerciais:

Valor	Código da Cor	Tolerância
10 Ω	Marrom-Preto-Preto	±1%
12 Ω	Marrom-Vermelho-Preto	±1%
15 Ω	Marrom-Verde-Preto	±1%
22 Ω	Vermelho-Verde-Preto	±1%
33 Ω	Laranja-Laranja-Preto	±1%
47 Ω	Amarelo-Violeta-Preto	±1%
56 Ω	Verde- Azul-Preto	±1%
68 Ω	Azul-Cinza-Preto	±1%
100 Ω	Marrom-Preto-Marrom	±1%
120 Ω	Marrom-Vermelho-Marrom	±1%
150 Ω	Marrom-Verde-Marrom	±1%
220 Ω	Vermelho-Verde-Marrom	±1%
330 Ω	Laranja-Laranja-Marrom	±1%
470 Ω	Amarelo-Violeta-Marrom	±1%
560 Ω	Verde-Azul-Marrom	±1%
680 Ω	Azul-Cinza-Marrom	±1%
1 kΩ	Marrom-Preto-Vermelho	±1%
1.2 kΩ	Marrom-Vermelho-Vermelho	±1%
1.5 kΩ	Marrom-Verde-Vermelho	±1%
2.2 kΩ	Vermelho-Verde-Vermelho	±1%
3.3 kΩ	Laranja-Laranja-Vermelho	±1%
4.7 kΩ	Amarelo-Violeta-Vermelho	±1%
5.6 kΩ	Verde-Azul-Vermelho	±1%
6.8 kΩ	Azul-Cinza-Vermelho	±1%
10 kΩ	Marrom-Preto-Laranja	±1%
12 kΩ	Marrom-Vermelho-Laranja	±1%
15 kΩ	Marrom-Verde-Laranja	±1%

Valor	Código da Cor	Tolerância
22 kΩ	Vermelho-Verde-Laranja	±1%
33 kΩ	Laranja-Laranja-Laranja	±1%
47 kΩ	Amarelo-Violeta-Laranja	±1%
56 kΩ	Verde-Azul-Laranja	±1%
68 kΩ	Azul-Cinza-Laranja	±1%
100 kΩ	Marrom-Preto-Amarelo	±1%
120 kΩ	Marrom-Vermelho-Amarelo	±1%
150 kΩ	Marrom-Verde-Amarelo	±1%
220 kΩ	Vermelho-Verde-Amarelo	±1%
330 kΩ	Laranja-Laranja-Amarelo	±1%
470 kΩ	Amarelo-Violeta-Amarelo	±1%
560 kΩ	Verde-Azul-Amarelo	±1%
680 kΩ	Azul-Cinza-Amarelo	±1%
1 MΩ	Marrom-Preto-Verde	±1%
1.2 MΩ	Marrom-Vermelho-Verde	±1%
1.5 MΩ	Marrom-Verde-Verde	±1%

Tabela 02: Valores de resistores comerciais.

Podemos notar que para cada valor, temos cores diferentes e essa é a forma que os resistores são marcados para indicação da sua resistência. Temos na Figura 16 alguns modelos de resistores.

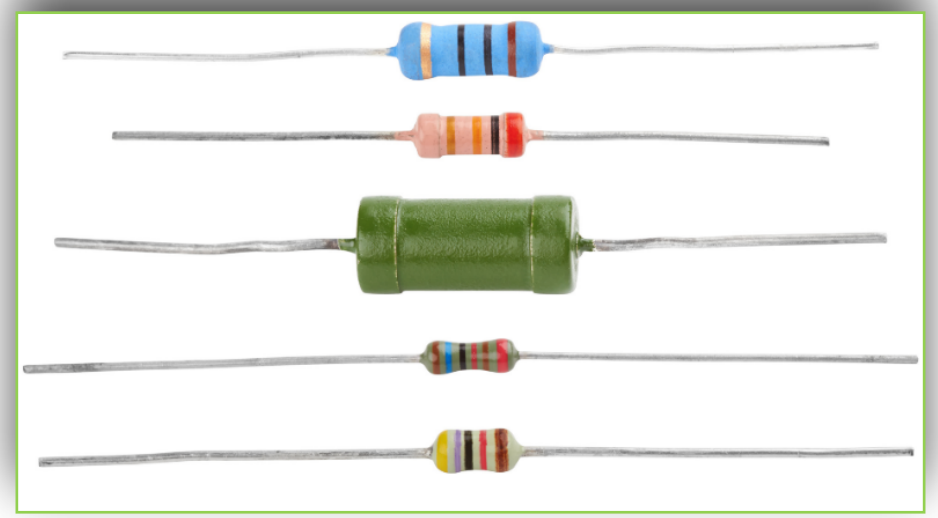


Figura 16: Alguns modelos de resistores.

Temos atualmente resistores de potência, para maiores corrente, resistores de 5 linhas, que são de precisão e resistores ajustáveis, que veremos mais a frente, entre outros. Para lermos os valores, basta usarmos o código de cores dos resistores, existem diversas versões que você encontra na internet. Temos um modelo mostrado na Figura 17. O código de cores dos resistores é uma convenção utilizada para indicar o valor nominal e a tolerância dos resistores. Ele é composto por quatro ou cinco faixas coloridas que representam números e multiplicadores, além de uma faixa que indica a tolerância do valor nominal do resistor. Para resistores de 4 faixas, temos as duas primeiras para valores, a terceira para o fator de multiplicação e a quarta para a tolerância.

Por exemplo, um resistor com as faixas de cores marrom, preto, vermelho e dourado indica um valor nominal de 1,0 kΩ com uma tolerância de ±5%. A primeira faixa marrom indica o dígito 1, a segunda faixa preta indica o dígito 0, a terceira faixa vermelha indica o fator multiplicativo de 100, resultando em um valor nominal de 1,0 kΩ (ou 1.000 Ω). A faixa dourada indica a tolerância de ±5%, o que significa que o valor real do resistor pode variar em até 5% em relação ao valor nominal.

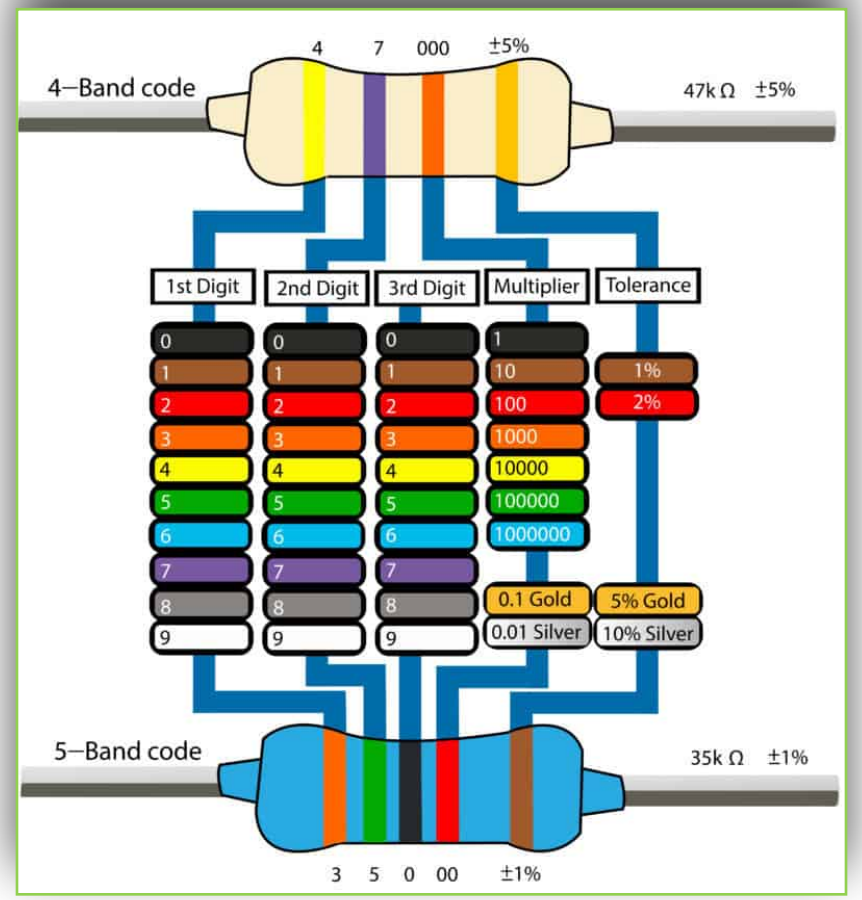


Figura 17: Código de cores dos resistores.

Para resistores de 5 faixas, o terceiro indica o digito 3, o quarto indica o fator de multiplicação e a quinta faixa a tolerância, que geralmente é 1% ou 2%. Os resistores de 5 faixas são mais precisos e usados em projetos na qual os resistores precisam de valores exatos.

9.3 – Código para o semáforo

Vamos entender algumas novas linhas e comandos nesse código, o MicroPython é bem intuitivo e irá deixar o código bem autoexplicativo. Para o nosso projeto do semáforo, vamos digitar o código a seguir:

```
from machine import Pin
from utime import sleep
LedRed=Pin(2,Pin.OUT)
LedYellow=Pin(3,Pin.OUT)
LedGreen=Pin(4,Pin.OUT)

while(True):
    LedRed.value(1)
    LedYellow.value(0)
    LedGreen.value(0)
    sleep(3)

    LedRed.toggle()
    LedGreen.toggle()
    sleep(1)

    LedYellow.toggle()
    LedGreen.toggle()
    sleep(3)
```

Em MicroPython, machine é um módulo que fornece acesso a hardware específico da placa em que o código está sendo executado. O pin é uma classe dentro do módulo machine que permite controlar as entradas e saídas digitais da placa. Como vimos no início. Ao usar from machine import Pin estamos importando a classe Pin do módulo machine. Isso permite que você crie instâncias da classe Pin e use os métodos e atributos da classe para controlar o estado do pino. Podemos usar apenas a classe Pin do módulo machine, que traduzindo fica: "De máquina(módulo) importe Pin(classe)". Logo LedRed=machine.Pin(2,machine.Pin.OUT) será substituído por uma linha mais simples: LedRed=Pin(2,Pin.OUT). Assim também para Sleep, importando ela do módulo utime, usamos somente sleep().

No primeiro código, tínhamos apenas 1 variável: o LED onboard. Aqui vamos usar LEDs externos, portanto, vamos usar 3 variáveis. A lógica é a mesma do primeiro código, o que muda é o pino que vamos usar, no nosso caso, os pinos 2, 3 e 4 para os LEds Red, Yellow e Green respectivamente. Como foi visto

anteriormente, precisamos definir o pino e configurá-lo como saída, portanto, faremos isso nas próximas linhas:

```
LedRed=Pin(2,Pin.OUT)
LedYellow=Pin(3,Pin.OUT)
LedGreen=Pin(4,Pin.OUT)
```

No nosso loop while(True): vamos simplesmente alterar os valores dos LEDs como acontece em um semáforo. Como temos uma simulação, vamos trabalhar com 3 segundos para o vermelho e verde, e 1 segundo para o amarelo. Portando, vamos começar com o semáforo em vermelho, temos então:

```
    LedRed.value(1)
    LedYellow.value(0)
    LedGreen.value(0)
    sleep(3)
```

Nas próximas linhas, temos um novo comando: toggle. Em uma tradução literal, significa alternar, já que, quando mudamos do vermelho para o verde, por exemplo, não precisamos indicar o valor do amarelo, pois seu estado não irá mudar, ele permanece apagado. O que nos interessa é alternar o LEDs que devem mudar naquele momento. Vamos então dar um toggle depois de 3 segundos no led vermelho, alternando o seu estado, se ele estava desligado, obviamente, vamos ligá-lo. O verde deve ser ligado quando o vermelho apagar, logo, como estava desligado, um toggle irá ligá-lo, e assim para os 3 estados de um semáforo.

9.4 – Simulador online

O wokwi.com permite a simulação e programação da nossa placa. Na Figura 18 temos a página inicial. Vamos clicar nos 3 pontos e colocar em full screen. Podemos agora, iniciar a montagem do circuito. Adicione componentes, clicando no "+" e adicione resistores, capacitores, LEDs etc. Para conectá-los, basta clicar nos seus terminais e arrastar até o outro ponto de conexão, visto na Figura 19.

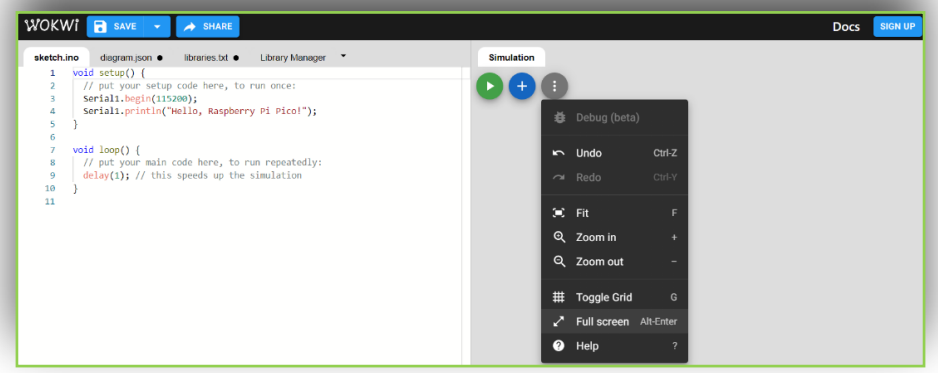


Figura 18: Página inicial do simulador online.

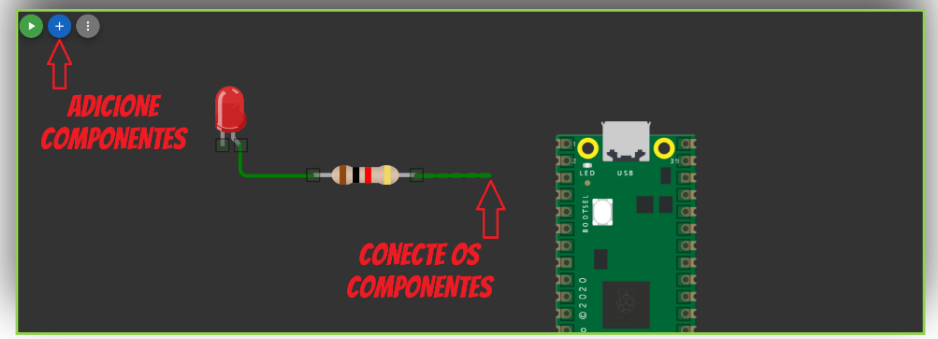


Figura 19: Adicionando e conectando componentes.

Caso a placa Raspberry Pi Pico não apareça no simulador, saia o modo full screen, clique no canto superior esquerdo, no ícone do wokwi e adicione a placa, como mostra a Figura 20.

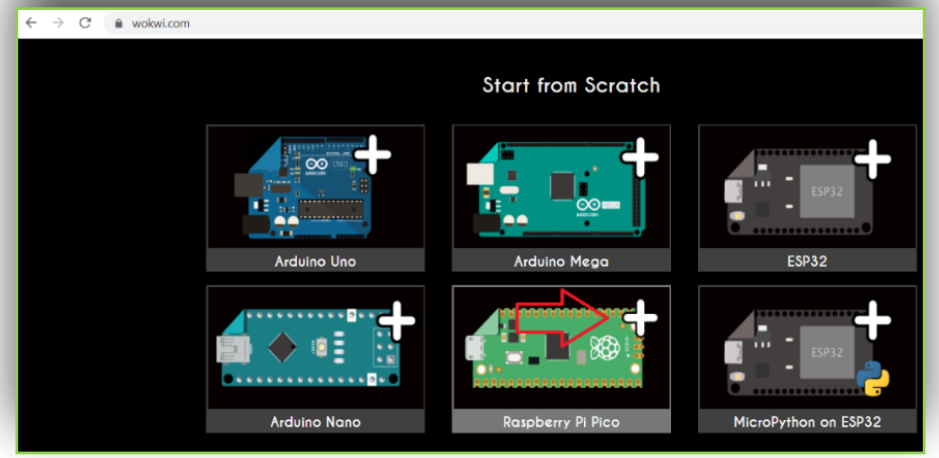


Figura 20: Adicionando a placa Raspberry Pi Pico.

9.5 – O que é Protoboard?

Uma protoboard é uma placa de circuito que permite a montagem e conexão temporária de componentes eletrônicos. Possui uma sequência de furos e trilhas condutoras em sua superfície, onde os componentes podem ser inseridos e conectados através de fios ou jumpers. Os furos são interligados em linhas e colunas, permitindo a criação de circuitos. Cada furo é conectado a um ponto de contato metálico internamente, possibilitando a conexão elétrica entre os componentes. Isso facilita a montagem rápida de circuitos sem a necessidade de solda.

Os componentes, como resistores, capacitores, transistores, chips integrados e outros, são inseridos nos furos. A protoboard é uma ferramenta para prototipagem e teste de circuitos, mas não é adequado para aplicações finais devido à sua natureza temporária e à possibilidade de conexões instáveis, além dos materiais das conexões não serem de um bom condutor. Na Figura 21 temos os detalhes de uma protoboard.

Agora, podemos montar o circuito da Figura 22, tanto no simulador ou na bancada eletrônica, teremos o mesmo resultado. Analise as conexões, os

pontos de alimentação e o código em MicroPython. Caso não consiga escrever seu código em Python, clique na seta ao lado no menu do lado esquerdo e crie o arquivo main.py, como mostra a Figura 23. Com Raspberry Pi Pico W, as conexões são exatamente iguais. Vamos lembrar de conectar as linhas horizontais da protoboard, pois elas serão a alimentação do nosso circuito. Esse projeto não tem interação com usuários, como por exemplo, passagem para pedestre. E se quisermos adicionar um botão para que um pedestre consiga solicitar a passagem? Precisamos trabalhar com input, para dizer a placa que vamos receber um comando externo e que iremos realizar alguma ação depois disso. Vamos agora aprender a usar esse botão e a fazer a verificação dele de forma paralela do funcionamento do semáforo, através de threads. Vamos nessa!!!

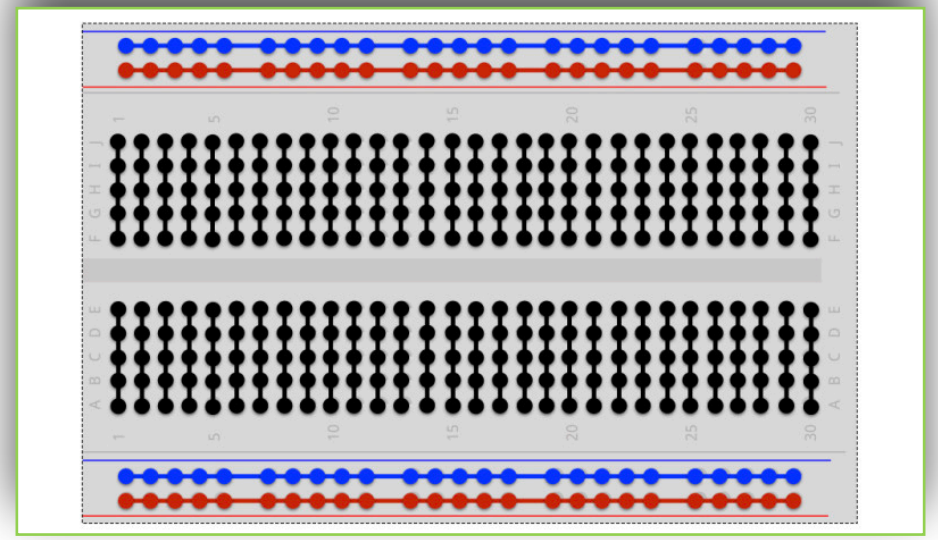


Figura 21: Protoboard e suas conexões.

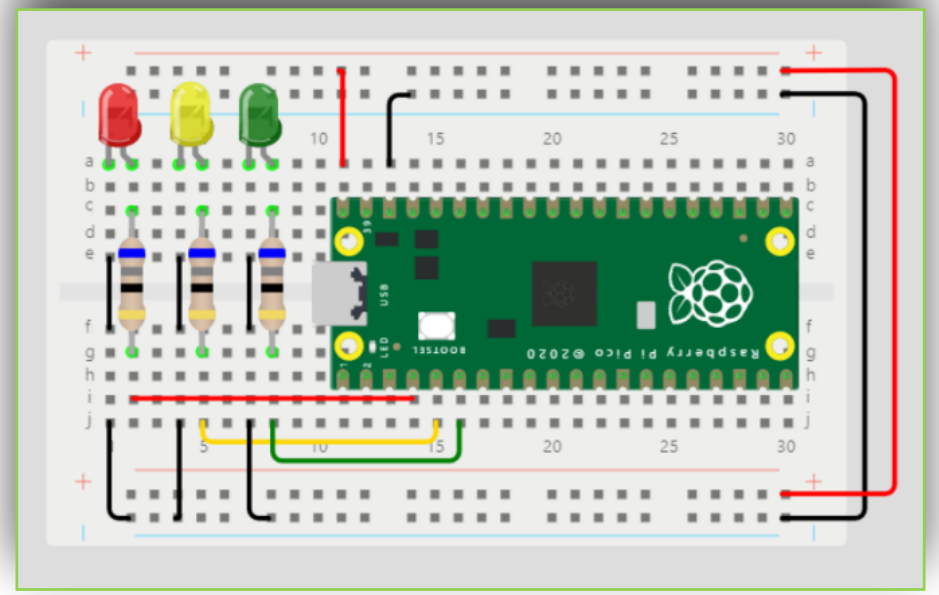


Figura 22: Circuito do semáforo montado em Protoboard.

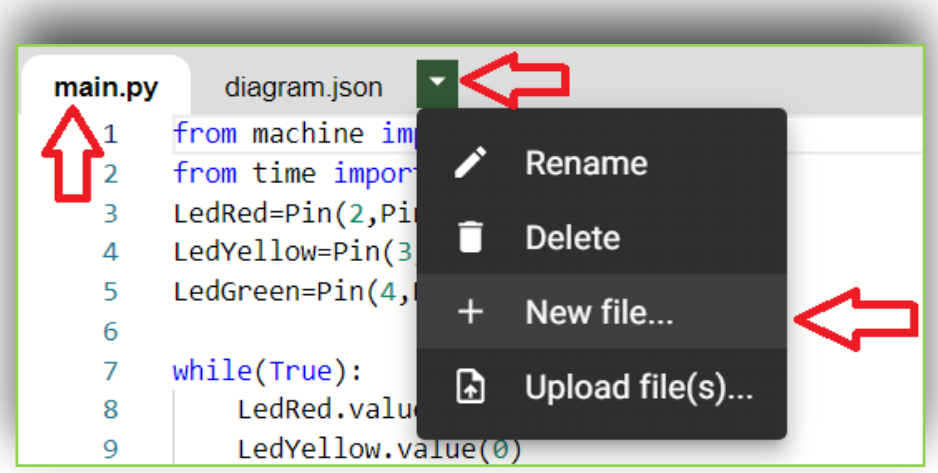


Figura 23: Adicionando arquivo main.py no simulador.

Aula 10 - Semáforo interativo

Um semáforo com passagem para pedestres funciona como um dispositivo de controle de tráfego que permite que os pedestres atravessem com segurança uma via onde há fluxo de veículos. Ele é projetado para equilibrar o fluxo de veículos e a segurança dos pedestres. Normalmente, um semáforo com passagem para pedestres é composto por três aspectos principais:

Sinal para veículos: Essa parte do semáforo controla o fluxo de veículos na via. Geralmente, é composta por luzes vermelhas, amarelas e verdes. A luz vermelha indica que os veículos devem parar, a luz amarela alerta para a parada iminente e a luz verde permite que os veículos prossigam.
Sinal para pedestres: Essa parte do semáforo é dedicada aos pedestres. Geralmente, é representada por uma figura de um pedestre em verde e um em vermelho. Quando a figura de pedestre está verde, os pedestres podem atravessar com segurança. Quando o símbolo de "mão" está vermelho, indica que os pedestres devem esperar.
Botão de solicitação: Em muitos semáforos com passagem para pedestres, há um botão de solicitação localizado próximo à calçada. Os pedestres precisam pressionar esse botão para solicitar a passagem segura. Quando o botão é acionado, ele envia um sinal para o semáforo, ativando o ciclo que permite a travessia.

Inicialmente, o sinal para veículos está verde e o sinal para pedestres está vermelho. Os veículos têm prioridade. Quando um pedestre pressiona o botão de solicitação, o semáforo entra em um ciclo para permitir a passagem segura. Após um certo intervalo de tempo, o sinal para veículos muda de verde para amarelo e, em seguida, para vermelho, indicando que eles devem parar. Ao mesmo tempo, o sinal para pedestres muda de vermelho para verde, permitindo que os pedestres atravessem a via com segurança. Após um período determinado, o sinal para pedestres muda de verde para vermelho, indicando que os pedestres devem parar. O sinal para veículos volta a ficar verde, permitindo que o tráfego de veículos prossiga.

10.1 – O que são thread?

Antes de irmos para nosso código, vamos entender as threads. O thread, em português: fio de execução ou encadeamento de execução, é uma forma como um processo/tarefa do programa de computador ser dividido em duas ou mais tarefas que podem ser executadas simultaneamente. Como vimos no começo desse livro, o RP2040 possui dois núcleos, ou seja, ele pode processar duas tarefas de forma simultânea, uma em cada núcleo, sem que uma interfira no tempo de execução da outra, fazendo uso das threads. Em MicroPython, as threads são unidades independentes de execução que permitem a execução simultânea de diferentes partes do código. Uma thread pode ser vista como uma "sub-rotina" que pode ser executada em paralelo com outras threads, permitindo que várias tarefas sejam executadas simultaneamente. a responsabilidade de gerenciar a execução das threads recai sobre o programador, chamando-a dentro do código e usando bibliotecas, como veremos. Podemos executar tarefas como leitura de sensores, controle de atuadores e comunicação em rede de forma assíncrona, tornando o código mais eficiente e responsivo. Vamos digitar o código a seguir. Não se preocupe se não entender algumas linhas agora, como dito, vamos entender cada detalhe.

```
from machine import Pin
from utime import sleep
import _thread
#=====================Configuration=========================#
LedRed=Pin(0,Pin.OUT)
LedYellow=Pin(1,Pin.OUT)
LedGreen=Pin(2,Pin.OUT)
Button=Pin(3,Pin.IN,Pin.PULL_DOWN)
Buzzer=Pin(4,Pin.OUT)
#=======================Variable============================#
global ButtonPressed
ButtonPressed=False
#=======================thread==============================#
def Pressed():
    global ButtonPressed
    while True:
        if Button.value()==1:
            ButtonPressed=True
        sleep(0.1)
_thread.start_new_thread(Pressed,())
#=======================Main================================#
while True:
    LedYellow.value(0)
    LedRed.value(1)
```

```
    if ButtonPressed==True:
        for i in range(10):
            Buzzer.value(1)
            sleep(0.25)
            Buzzer.value(0)
            sleep(0.25)
        global ButtonPressed
        ButtonPressed = False
    else:
        sleep(5)
    LedRed.value(0)
    LedGreen.value(1)
    sleep(5)
    LedGreen.value(0)
    LedYellow.value(1)
    sleep(2)
```

No início do nosso código, temos as bibliotecas que vamos utilizar, na qual já temos duas bem conhecidas. O que temos de novidade é a import _thread. Essa nos permite usar um processamento em paralelo com o código principal. Imagine que o semáforo está em funcionamento, e em um momento exato ele irá verificar se tem alguém pedindo passagem na faixa de pedestre. Se o pedestre pressiona o botão quando o código estiver na linha que faz essa verificação, tudo beleza. Mas, qual a probabilidade de isso acontecer? Qual a chance de o usuário pressionar o botão justamente no momento que o código está na linha que verifica isso? Exatamente por isso que precisamos estar o tempo todo verificando se o botão foi pressionado sem que essa verificação pause ou interfira no funcionamento do semáforo. O processador do Raspberry Pi Pico e Raspberry Pi Pico W possuem dois núcleos, podemos então rodar o código principal em um núcleo e a verificação do botão no outro, e, quando houver a solicitação, o núcleo avisará ao outro e irá alterar o fluxo do código principal, para isso que usamos essa biblioteca.

Em Configuration temos o cabeçalho, que está entre "#", portando, ele não é compilado no código, serve apenas como anotação para o nosso código. Isso irá nos ajudar a entender cada etapa, principalmente quando precisarmos fazer alguma alteração futuramente. Com as anotações, teremos sempre um lembrete do que se trata aquele código. Vamos definir então as configurações do nosso projeto. Temos o nossos led, como no primeiro projeto, porém, agora temos os dois componentes novos: um buzzer e um botão.

Um buzzer é um dispositivo eletrônico que produz um som contínuo e característico quando ativado. Eles são usados para fins de sinalização ou

aviso em uma variedade de dispositivos eletrônicos, como alarmes, relógios, eletrodomésticos, brinquedos, sistemas de segurança e muitos outros. Podem variar em termos de formato, tamanho e recursos. Alguns buzzers têm apenas uma frequência fixa e emitem um som simples, enquanto outros podem ser programáveis e reproduzir uma variedade de tons e melodias. Além disso, alguns buzzers podem ter componentes adicionais, como um oscilador interno para gerar diferentes frequências de som. O modelo que vamos usar no nosso projeto, é visto na Figura 24.

Figura 24: Buzzer simples.

Agora, vamos entender o que são resistores de pull-up e pull-down antes de darmos continuidade do nosso código. Como dito, vamos aprender eletrônica e programação de forma detalhada.

10.2 – Resistores de pull-up e pull-down

Resistores de pull-up e pull-down são componentes usados em circuitos eletrônicos para ajudar a manter os níveis de tensão das linhas de sinal estáveis. Esses resistores são usados em circuitos digitais para evitar problemas causados por flutuações de tensão e fornecer um valor de referência definido para os pinos de um microcontrolador, por exemplo, quando nenhum outro dispositivo estiver atuando sobre elas. A escolha entre resistor de pull-up ou pull-down depende do projeto específico e dos componentes elétricos envolvidos no circuito. Na Figura 25 temos o exemplo com as duas configurações. No lado esquerdo temos o resistor de pull-up com nosso botão e no lado direito temos o resistor de pull-down com nosso botão.

Quando você conecta um botão a um microcontrolador, geralmente você precisa garantir que o pino de entrada do botão esteja sempre em um estado definido, seja alto (nível "1") ou baixo (nível "0"). Sem os resistores de pull-up ou pull-down, o pino ficaria flutuante quando o botão não estivesse pressionado, o que poderia levar a resultados imprevisíveis. Isso quer dizer que poderíamos ter valores como 3.1V, 2V, 1.8V etc. Isso acontece por ruídos ou sinais mal interpretados.

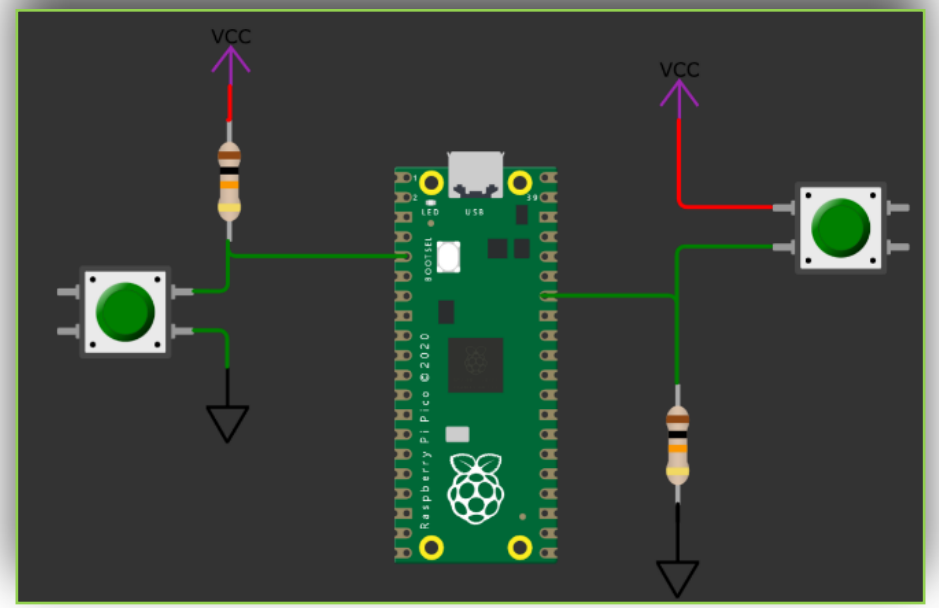


Figura 25: Configuração de pull-up e pull-down.

Vamos considerar o exemplo de um botão conectado a um microcontrolador usando um resistor de pull-up, como visto acima. Nesse caso, uma extremidade do botão é conectada à linha de sinal ou no GPIO da placa e a outra extremidade é conectada ao GND. Um resistor de pull-up é conectado entre o pino da placa e a tensão de alimentação (Vcc).

O resistor de pull-up "puxa" o pino da placa para um estado lógico alto (Vcc) quando o botão não está pressionado. Quando o botão é pressionado, ele conecta o pino diretamente ao GND, resultando em um estado lógico baixo (GND). Assim, o microcontrolador pode detectar quando o botão é pressionado, pois o pino é puxado para o estado lógico baixo.

Isso acontece porque a corrente elétrica busca sempre o caminho de menor resistência. Quando o botão não está pressionado, temos uma resistência infinita, portando, o resistor "ganha" por ter uma resistência menor e mantém o pino em nível alto, pois é onde ele está conectado. Quando o botão é pressionado, sua resistência agora vai para quase zero, "ganhando" do resistor.

No caso de um resistor de pull-down, a lógica é invertida. O resistor de pull-down é conectado ao pino da placa e o GND puxando o pino para um estado lógico baixo, quando o botão não está pressionado. Quando o botão é pressionado, ele conecta o pino diretamente à tensão de alimentação (Vcc), e como resultado, em um estado lógico alto (Vcc). Esses resistores garantem que a linha de sinal esteja sempre em um estado definido, mesmo quando o botão não está pressionado, evitando flutuações e fornecendo uma referência clara para a leitura do microcontrolador. Dessa forma, o microcontrolador pode identificar quando o botão foi pressionado ou liberado com confiabilidade.

10.3 – Analisando o código

Como vimos acima, temos as bibliotecas já conhecidas e a mais nova que acabamos de aprender. Agora, iremos partir para as próximas linhas. Vamos lá!

```
#====================Configuration=========================#
LedRed=Pin(0,Pin.OUT)
LedYellow=Pin(1,Pin.OUT)
LedGreen=Pin(2,Pin.OUT)
Button=Pin(3,Pin.IN,Pin.PULL_DOWN)
Buzzer=Pin(4,Pin.OUT)
```

Aqui temos as configurações de hardware dos pinos que vamos usar. 0, 1 e 2 são usados como saída e serão nossos LEDs RED, YELLOW e GREEN respectivamente. Observe que o pino do button é uma entrada e devemos definir isso para a placa, temos Button=Pin(3,Pin.IN,Pin.PULL_DOWN), na qual estamos dizendo que o botão está conectado no pino 3, como uma entrada e usará o resistor de pull-down interno. Como vimos no exemplo como funciona os resistores, poderíamos adicioná-los a montagem na protoboard, porém podemos economizar componentes e espaço na montagem, pois as placas Raspberry Pi Pico e Pico W possuem esses resistores internamente, nos bastando ativá-los via código, feito na linha Pin.PULL_DOWN.

```
#=======================Variable=========================#
global ButtonPressed
ButtonPressed=False
```

Vamos agora criar um variável global chamada ButtonPressed e em ButtonPressed=False estamos iniciando-a como falsa, ou seja, o botão não está pressionado. Mas por que global? Bom, em MicroPython para Raspberry Pi Pico, uma variável global é uma variável que pode ser acessada e modificada em qualquer parte do programa, independentemente de seu escopo. Uma variável global é declarada fora de qualquer função ou classe e está disponível para uso em todo o código. Como vimos anteriormente, iremos trabalhar com dois programas simultâneos, por isso precisamos transitar entre eles, então precisamos da variável global. Com ela será possível trabalhar nos dois núcleos do RP2040.

10.4 – Funções e Variáveis em MicroPython

Em MicroPython, as variáveis são usadas para armazenar valores que podem ser referenciados e modificados ao longo do programa. Ao contrário de algumas outras linguagens, MicroPython é uma linguagem de tipagem dinâmica, o que significa que você não precisa declarar explicitamente o tipo de uma variável. Declaramos a variável e a definimos ela como False, que significa para o código que, como o botão não está pressionado, ela é falsa,

quando o botão for pressionado, ela se torna verdadeira, como veremos a frente. Em MicroPython, assim como em outras linguagens de programação, uma função é um bloco de código que realiza uma tarefa específica e pode ser chamado/executado em diferentes partes do programa. As funções permitem agrupar um conjunto de instruções que realizam uma tarefa específica, tornando o código mais organizado, reutilizável e fácil de manter. Uma função em MicroPython é definida usando a palavra-chave def, seguida pelo nome da função e parênteses contendo quaisquer argumentos necessários. Aqui está um exemplo básico de uma função MicroPython:

```
def saudacao(nome):                                    #Define a função
   print("Olá, " + nome + "! Bem-vindo(a) à MicroPython.") #Exibe mensagem
saudacao("João")                                       #Chama a função
```

Teríamos então a frase:

"Olá, João ! Bem-vindo(a) à MicroPython".

Neste exemplo, a função saudacao é definida com um argumento nome. Quando a função é chamada com um valor para o argumento, ela imprime uma mensagem de saudação personalizada. Vamos então criar a função que verifica o botão:

```
#=========================thread============================#
def Pressed():
   global ButtonPressed
   while True:
      if Button.value()==1:
         ButtonPressed=True
      sleep(0.1)
_thread.start_new_thread(Pressed,())
```

A função Pressed(): é definida. Ela não recebe nenhum argumento e a linha global ButtonPressed indica que a variável é global, o que significa que pode ser acessada tanto dentro da função quanto fora dela. Essa linha é necessária para informar ao interpretador do MicroPython que a variável está sendo usada globalmente, em vez de ser uma variável local específica da função, que permite usá-la para acessarmos os núcleos do RP2040.

Em seguida, temos o while True: que significa que o código dentro do loop será executado repetidamente até que ocorra uma interrupção externa ou uma condição dentro do loop seja falsa. No nosso caso estamos verificando o botão em loop, onde temos uma condição if Button.value()==1: onde Button é um objeto que representa um botão físico ou um pino de entrada e value() retorna o estado atual do botão ou pino, sendo 1 quando está pressionado e 0 quando não está pressionado. Se o botão estiver pressionado (valor 1), a condição é verdadeira. Basicamente, o if significa "se", ou seja, se o botão for pressionado a variável ButtonPressed será verdadeira (True). Tudo que

estiver na indentação do if será feito se a condição "se" for atendida. Quando if é verdadeira, a variável global ButtonPressed é definida como True, indicando que o botão foi pressionado. Após o if, chamamos sleep(0.1), que faz com que o código aguarde por 0,1 segundos, ou 100 milissegundos, antes de prosseguir para a próxima iteração do loop. Isso ajuda a evitar um consumo excessivo de recursos do processador, já que o loop seria executado rapidamente sem pausas além de evitar ruídos mecânicos dos botões.

Na próxima linha temos a função _thread.start_new_thread(), que é um recurso do MicroPython para iniciar uma nova thread que executará uma função específica. Passamos Pressed() como o primeiro argumento, que é o nome da função Pressed() definida anteriormente. Isso indica qual função será executada na nova thread, ou seja, a função que verifica o botão em loop, está rodando separadamente em um dos dois núcleos do RP2040, não interferindo no código principal. O segundo argumento está vazio "()", indicando que não há argumentos adicionais a serem passados para a função Pressed(). Resumindo, esse trecho de código cria uma nova thread que executará a função Pressed() em segundo plano que verifica continuamente o estado de um botão ou pino de entrada e define uma variável global ButtonPressed como True quando o botão for pressionado a qualquer momento. Vamos para o nosso loop principal.

```
    #==========================Main============================#
while True:
    LedYellow.value(0)
    LedRed.value(1)
    if ButtonPressed==True:
        for i in range(10):
            Buzzer.value(1)
            sleep(0.25)
            Buzzer.value(0)
            sleep(0.25)
        global ButtonPressed
        ButtonPressed = False
    else:
        sleep(5)
    LedRed.value(0)
    LedGreen.value(1)
    sleep(5)
    LedGreen.value(0)
    LedYellow.value(1)
    sleep(2)
```

Iniciamos o loop com while True: e os leds amarelo e vermelho são configurados com LedYellow.value(0) e LedRed.value(1), respectivamente. Nesse ponto estamos desligando o led amarelo e ligando o led vermelho. Mas por que não desligamos o verde? Isso acontece porque por padrão os pinos da placa são iniciados em nível baixo, porém, quando o loop reiniciar, precisamos desligar o led amarelo e ligar o vermelho para reiniciarmos.

Chegamos agora no nosso if e se ButtonPressed for True, ou seja, se no outro núcleo que está rodando o código que verifica o botão, ele foi pressionado e a variável mudou, no código principal iremos ter acesso a isso, pois como vimos, a variável é global. O laço for em conjunto com a função range() vai percorrer uma sequência de números. A função range() retorna uma sequência de números começando a partir de um valor inicial, indo até um valor final, com um intervalo especificado. Aqui está um exemplo que percorre os números de 0 a 4:

```
for i in range(5):
   print(i)
```

Basicamente estamos dizendo que: "Para variável i em um escala de 0 a 5, exiba cada valor que i assumir". A saída será os 5 valores: 0, 1, 2, 3 e 4. Nessa função acionaremos nosso buzzer na forma de um sinal típico de atenção. Temos a entrada em um loop for que se repete 10 vezes: O buzzer é ligado com Buzzer.value(1), aguardamos 0,25 segundos com sleep(0.25), e o buzzer é desligado com Buzzer.value(0) e aguardamos mais 0,25 segundos com sleep(0.25). Tudo isso no trecho de código abaixo:

```
for i in range(10):
            Buzzer.value(1)
            sleep(0.25)
            Buzzer.value(0)
            sleep(0.25)
        global ButtonPressed
        ButtonPressed = False
```

Esse aviso sonoro serve para indicar que está liberada a passagem para pedestres. Em seguida, a variável global ButtonPressed é definida como Falsa, para evitarmos entrar no laço if novamente sem que o botão tenha sido pressionado mais uma vez.

Se ButtonPressed não for verdadeira, ou seja, no outro núcleo não houve a solicitação de passagem com o pressionar do botão, o código aguarda 5 segundos com sleep(5). Esse é o tempo que vamos ficar com o led vermelho acesso. O led vermelho é desligado com LedRed.value(0) e o led verde é ligado com LedGreen.value(1). O código aguarda mais 5 segundos com sleep(5). O led verde é desligado com LedGreen.value(0) e o led amarelo é ligado com LedYellow.value(1). O código aguarda 2 segundos com sleep(2).

Então para reiniciamos nosso código, precisamos desligar o amarelo e ligar o vermelho. Note que poderíamos fazer esse código usando a função toggle. Para fins de estudo, tente refazer o código usando-a.

Aula 11 - Alarme e sensor de movimento

Um alarme com sensor de movimento é projetado para detectar a presença de movimento em uma área específica e, quando ativado, acionar um alarme sonoro ou enviar um alerta para notificar sobre a atividade suspeita. Vamos explicar como esse tipo de alarme funciona em detalhes.

O sensor de movimento é a parte central do sistema e é responsável por detectar mudanças no ambiente. Existem diferentes tipos de sensores de movimento, mas os mais comuns são os sensores infravermelhos passivos (PIR - Passive Infrared) que vamos usar nesse projeto. O sensor de movimento envia os dados coletados para o Raspberry Pi Pico e na programação decidimos o que fazer. Vamos usar um alarme Sonoro e um Alerta visual. O dispositivo sonoro será um buzzer. Na Figura 26 temos o circuito montado.

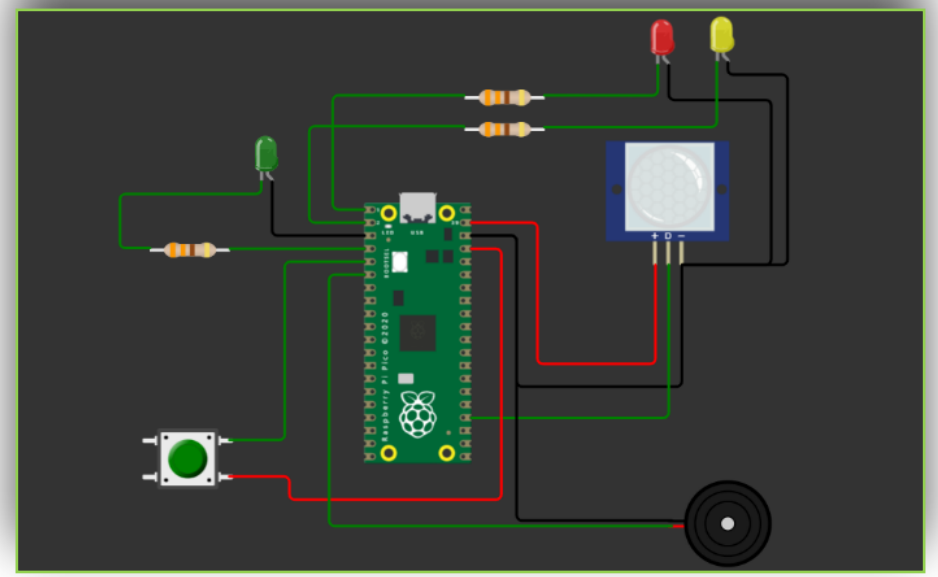


Figura 26: Circuito montado.

Geralmente, um sistema de alarme normalmente possui um período de tempo pré-definido durante o qual o alarme permanecerá ativo. Após esse período, o alarme pode ser desativado inserindo um código de segurança no teclado do sistema ou usando um controle remoto.

11.1 – Sensor de movimento PIR

Um sensor PIR (Sensor de Infravermelho Passivo) é um dispositivo eletrônico utilizado para detectar o movimento de objetos que emitem radiação infravermelha, como seres humanos e animais. Ele funciona utilizando o princípio de detecção de mudanças no calor emitido pelos

objetos em seu campo de visão. Os dois modelos mais comuns são vistos da Figura 27.

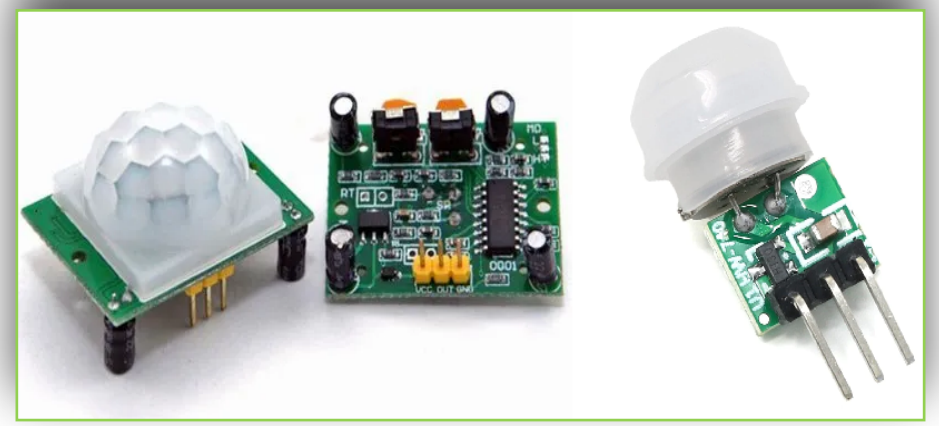


Figura 27: Sensor PIR e PIR mini.

O sensor PIR é composto por uma série de elementos sensíveis à radiação infravermelha, que geralmente são cristais piroelétricos. Esses cristais geram uma carga elétrica quando são expostos a mudanças no calor infravermelho e é coberto com lentes e divisores que dividem a área em múltiplas zonas. Cada zona está conectada a um elemento sensível no interior do sensor. Antes de ser ativado, o sensor PIR é ajustado à temperatura ambiente circundante para estabelecer um equilíbrio térmico em seus elementos sensíveis. Nesse estado, a carga elétrica gerada pela radiação infravermelha da zona e a temperatura ambiente se cancelam entre si. Quando um objeto em movimento (como uma pessoa ou um animal) entra no campo de visão do sensor PIR, sua temperatura muda em relação à temperatura ambiente. A mudança na radiação infravermelha provoca um desequilíbrio na carga elétrica dos elementos sensíveis na zona afetada.

A mudança na carga elétrica é convertida em um sinal elétrico, que é amplificado e processado por circuitos eletrônicos internos no sensor PIR. Se o sinal gerado exceder um determinado limite, é considerado que um movimento significativo foi detectado, e o sensor ativa sua saída. A saída pode ser uma mudança de estado em uma linha de sinal (alto ou baixo) ou a ativação de um alarme, dependendo do design e propósito do sensor. O modelo mini, quando detecta movimento, aciona o pino de saída (OUT) pra nível alto. No sensor padrão, temos ajuste de tempo em que o pino fica acionado, que pode ser ajustado de 300 milissegundos a 5 minutos. Temos também o ajuste de sensibilidade, que depende da aplicação, local e área que desejamos verificar, como mostra a Figura 28.

É importante mencionar que os sensores PIR apenas detectam mudanças na radiação infravermelha e não conseguem distinguir a forma ou identidade do objeto em movimento. Esses sensores tem preços acessível e boas respostas para projetos de automação em geral, desde acionamento de

lâmpadas até projetos mais complexos. Como dito, cada ajuste de sensibilidade e tempo depende da finalidade do projeto.

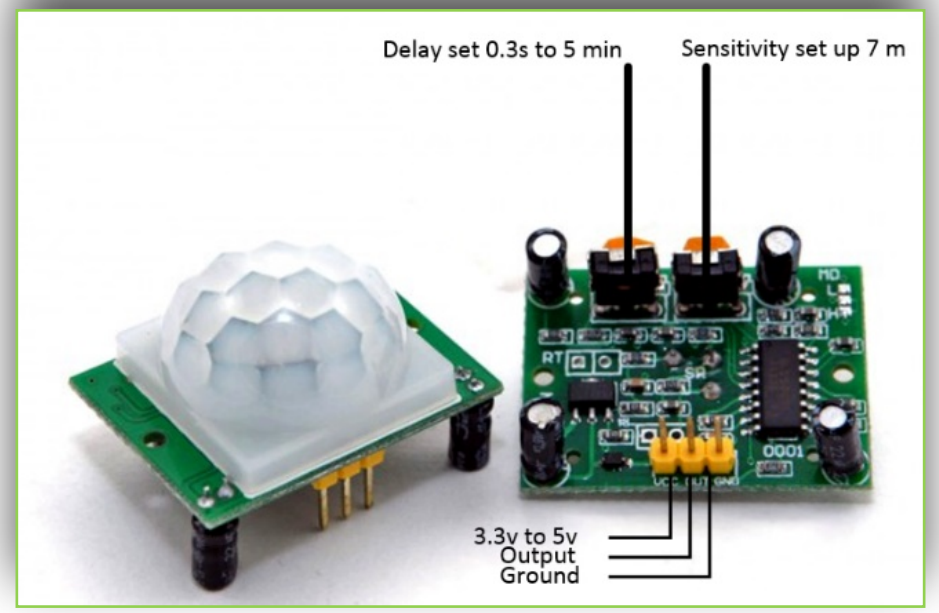


Figura 28: Pinos de ajustes PIR HC SR501.

11.2 – Análise do código

```
from machine import Pin
from utime import sleep
import _thread
#======================Configuration=========================#
LedRed=Pin(0,Pin.OUT)
LedYellow=Pin(1,Pin.OUT)
LedGreen=Pin(2,Pin.OUT)
Button=Pin(3,Pin.IN,Pin.PULL_DOWN)
Buzzer=Pin(4,Pin.OUT)
Sensor=Pin(18,Pin.IN,Pin.PULL_DOWN)
#=======================Variable============================#
global ButtonPressed
global SensorDetected
ButtonPressed=False
SensorDetected=False
#=======================threadAlarme=======================#
def Alarme():
    global ButtonPressed
    global SensorDetected
    while True:
        if Sensor.value()==1:
            SensorDetected=True
        if Button.value()==1:
```

```
            ButtonPressed=True
    sleep(0.1)
_thread.start_new_thread(Alarme,())
#=========================Main============================#
while True:
    ButtonPressed = False
    LedGreen.value(1)
    LedRed.value(0)
    LedYellow.value(0)
    Buzzer.value(0)
    while SensorDetected==True:
        LedGreen.value(0)
        LedRed.toggle()
        LedYellow.toggle()
        sleep(0.05)
        Buzzer.toggle()
      if ButtonPressed==True:
            SensorDetected=False
```

As importações das bibliotecas já são bem conhecidas: Nesta parte, o código está importando os módulos necessários para a programação no MicroPython no Raspberry Pi Pico. A machine é um módulo que permite controlar as entradas e saídas do hardware, como pinos e dispositivos. O utime fornece funcionalidades relacionadas ao tempo, como o atraso e _thread nos permite utilizar processos em paralelo, como já vimos anteriormente.

Logo depois, estamos configurando os pinos de entrada/saída para os LEDs vermelho, amarelo e verde, o botão, o buzzer e o sensor. Os números (0, 1, 2, etc.) correspondem aos números dos pinos físicos no Raspberry Pi Pico. O argumento Pin.OUT configura o pino como saída, Pin.IN configura-o como entrada, e Pin.PULL_DOWN ativa um resistor de pull-down interno nos pinos de entrada. Temos depois duas variáveis globais, ButtonPressed e SensorDetected, são declaradas e inicializadas como False. Essas variáveis serão usadas para rastrear o estado do botão pressionado e do sensor detectado. Todos essas linhas já foram explicadas de forma mais detalhada nos projetos anteriores.

```
#=======================threadAlarme=======================#

def Alarme():
    global ButtonPressed
    global SensorDetected
    while True:
        if Sensor.value()==1:
            SensorDetected=True
```

```
        if Button.value()==1:
            ButtonPressed=True
    sleep(0.1)
_thread.start_new_thread(Alarme,())
```

Aqui estamos definindo a função Alarme(). Essa função é executada em um thread separado para monitorar o estado do sensor e do botão. Se o sensor detectar um sinal (valor 1), a variável SensorDetected é definida como True. Assim como se o botão for pressionado (valor 1), a variável ButtonPressed é definida como True. A função é executada em um loop infinito, com um pequeno atraso com sleep(0.1) entre as verificações.
O _thread.start_new_thread(Alarme, ()) inicia a execução dessa função em um thread separada. Na nossa função principal temos:

```
#=========================Main============================#
while True:
    ButtonPressed = False
    LedGreen.value(1)
    LedRed.value(0)
    LedYellow.value(0)
    Buzzer.value(0)
    while SensorDetected==True:
        LedGreen.value(0)
        LedRed.toggle()
        LedYellow.toggle()
        sleep(0.05)
        Buzzer.toggle()
      if ButtonPressed==True:
            SensorDetected=False
```

Nesta parte do código, começa o loop principal que será executado continuamente. Inicialmente, os LEDs são configurados: o LED verde é ligado em LedGreen.value(1) e os LEDs vermelho e amarelo são desligados em LedRed.value(0) e LedYellow.value(0). O buzzer também é desligado em Buzzer.value(0). Dentro do segundo loop, while SensorDetected == True:, os LEDs vermelho e amarelo piscam alternadamente, e o buzzer alterna entre ligado e desligado, criando um efeito de alarme sempre que o SensorDetected for True, que acontece quando o sensor detectar algum movimento no processo em paralelo, em "threadAlarme".

Se o botão for pressionado, ButtonPressed == True, que é verificado também no processo em paralelo pela thread, a variável SensorDetected é definida como False, redefinindo o estado do sensor, indicando que o usuário pressionou o botão que desliga o alarme. Tornando essa variável falsa, assim, não entraremos no loop while SensorDetected, até que aconteça outra detecção de movimento pelo sensor.

Aula 12 - ADC e sensor de temperatura

O ADC (Conversor Analógico-Digital) é um componente fundamental em muitos dispositivos eletrônicos, incluindo o Raspberry Pi Pico. Ele permite que o microcontrolador converta sinais analógicos, como tensões elétricas, em valores digitais que podem ser processados pelo sistema digital.

12.1 – Como funciona o ADC?

O Raspberry Pi Pico possui um ADC de 3 canais de 12 bits integrado, que é usado para medir valores analógicos, como tensões provenientes de sensores, potenciômetro ou outros dispositivos analógicos. Temos na placa o ADC0, ADC1 e ADC2 que são conectados nos pinos GP26, GP27 e GP28, respectivamente. Temos algumas etapas importantes que acontecem para que ocorra a correta conversão desses valores, temos:

Amostragem: O processo começa com a amostragem do sinal analógico que se deseja medir. Isso geralmente envolve a leitura da tensão presente em um determinado pino analógico do Raspberry Pi Pico. O sinal analógico é uma representação contínua, e a amostragem envolve a captura de valores desse sinal em intervalos regulares.
Quantização: O valor contínuo da tensão é convertido em um valor discreto por meio do processo de quantização. No caso do Raspberry Pi Pico, que possui um ADC de 12 bits, isso significa que o valor contínuo é dividido em 2^{12} (4096) níveis diferentes. Isso permite que a tensão analógica seja representada por um valor digital de 0 a 4095.
Digitalização: Após a quantização, o valor digital resultante é armazenado em um registrador dentro do microcontrolador. Esse valor pode então ser usado pelo programa para análise, processamento ou exibição.
Referência: É importante mencionar que o ADC requer uma referência para determinar o valor absoluto da tensão analógica. O Raspberry Pi Pico permite a seleção de referências internas ou externas para o ADC, dependendo das necessidades do projeto. Temos o pino ADC_VREF que podemos ajustar esse valor, e se não formos usá-lo, o valor definido padrão é de 3.3V.
Programação: No software, podemos configurar o ADC para definir a tensão de referência, o canal de entrada que desejamos medir e outros parâmetros relevantes. Em linguagens de programação como C ou MicroPython, podemos usar as funções e bibliotecas fornecidas para realizar a leitura do ADC.
Leitura e Processamento: Uma vez configurado, o microcontrolador inicia o processo de conversão sempre que solicitarmos a leitura do ADC. O valor digital obtido é então usado em nosso programa para tomar decisões, exibir informações, ajustar saídas ou qualquer outra tarefa necessária.

O uso do ADC no Raspberry Pi Pico permite que trabalhemos com sinais analógicos do mundo real, convertendo-os em valores digitais que podem ser processados pelo microcontrolador. Isso é particularmente útil para aplicações que envolvem sensores analógicos, como sensores de luz, temperatura, pressão, entre outros.

Vamos agora medir a temperatura usando nossa placa, que dispõe de um sensor próprio. Agora, podemos estar pensando em qual pino este sensor de temperatura integrado está conectado? Este sensor de temperatura integrado está conectado a um dos ADCs ou conversores analógico-digital.

O sensor de temperatura não possui um pino físico na placa, mas é acessado como ADC4. Este sensor de temperatura integrado funciona fornecendo uma tensão ao pino ADC4 que é proporcional à temperatura. Se você verificar a folha de dados do Raspberry Pi Pico, descobrirá que uma temperatura de 27 graus Celsius fornece uma tensão de 0,706 volts.

O sensor de temperatura funciona fornecendo uma tensão ao pino ADC4 que é proporcional à temperatura. A partir da ficha técnica, uma temperatura de 27 graus Celsius fornece uma tensão de 0,706 V. Em MicroPython podemos dimensionar os valores ADC para um intervalo de 16 bits. Portanto, obtemos efetivamente o intervalo de 0 a 65535. O microcontrolador trabalha em 3,3V, o que significa que um pino ADC retornará um valor de 65535 quando 3,3V for aplicado a ele ou 0 quando não houver tensão.

Podemos obter todos os valores intermediários quando a tensão aplicada ao pino estiver entre 0 e 3,3 V. A cada grau adicional a tensão diminui em 1,721 mV ou 0,001721V. Para converter a temperatura de 16 bits, vamos convertê-la de volta em volts, o que é feito com base na tensão máxima de 3,3 V usada pela placa Pico, temos a formula fornecida pelo datasheet da placa:

$$temperatura = 27 - (leitura - 0{,}706)/0{,}001721$$

Com essa conversão, o valor da temperatura mantém um valor entre 0 e 3,3, que podemos usar em um display ou armazenar para um banco de dados. Agora já temos o valor em °C (Celsius), se e quisermos usar Fahrenheit, usaremos a formula:

$$fahrenheit = celsius\_degrees * 9 / 5 + 32$$

12.2 – Entendendo o código

Como iremos usar o sensor onboard, não precisaremos de componentes externos por enquanto. Pra isso, vamos verificar a temperatura o shell do Thonny ide, muito parecido com o terminal usado na plataforma Arduino.

```
from machine import ADC
from utime import sleep
```

```
Sensor=ADC(4)
#=========================main============================#
while True:
    Read=(Sensor.read_u16()*3.3)/65535
    Temperature = 27 - (Read - 0.706)/0.001721
    print(Temperature)
    sleep(1)
```

No Thonny IDE, o termo shell se refere a uma janela de interação interativa com o interpretador Python. Essa janela permite que você execute comandos Python diretamente e veja os resultados imediatamente. É semelhante a um ambiente de terminal Python, onde você pode experimentar e testar trechos de código rapidamente. Para acessar o shell no Thonny IDE, basta acessa a aba inferior, como mostra a Figura 29.

O shell do Thonny IDE é útil para testar pequenos trechos de código, experimentar funções e verificar rapidamente os resultados sem a necessidade de criar um arquivo separado. Além disso, pode ser uma ferramenta valiosa para aprender e depurar Python, especialmente para iniciantes. Aqui, podemos ver o valor da temperatura calculada do sensor onboard, pois usamos a função print, que nada mais é do que a exibição do valor de uma variável, no caso, Temperature, no shell da IDE.

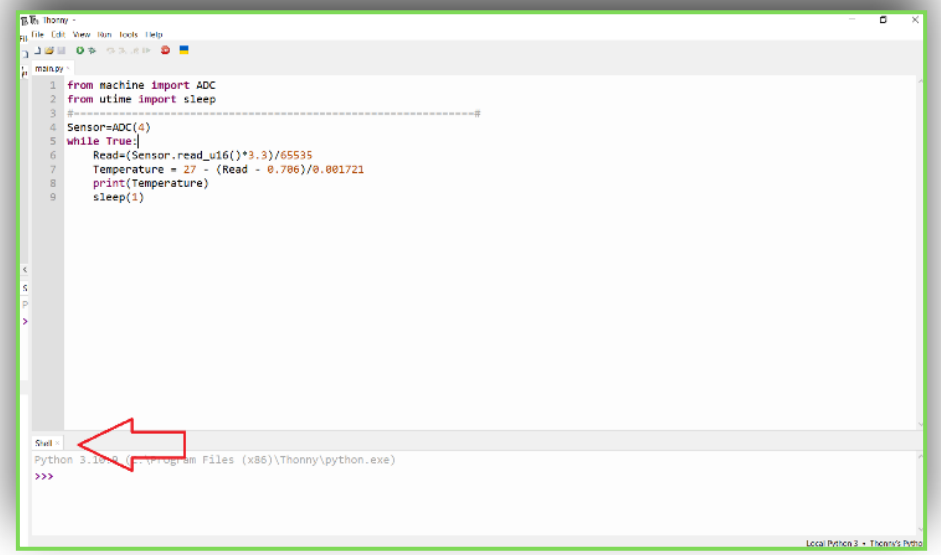


Figura 29: Shell da Thonny IDE.

Vamos agora explicar o código bem simples:

```
from machine import ADC
from utime import sleep
Sensor=ADC(4)
```

Aqui temos as bibliotecas já usadas anteriormente, mas com o ADC importado de machine, que irá nos permitir trabalharmos com o ADC da

placa Raspberry Pi Pico e Pico W. O que temos de diferente também é o objeto Sensor que está associado ao ADC 4 da nossa placa.

```
#=========================main============================#
while True:
    Read=(Sensor.read_u16()*3.3)/65535
    Temperature = 27 - (Read - 0.706)/0.001721
    print(Temperature)
    sleep(1)
```

Agora, o que vamos fazer é simplesmente criar nosso loop e fazer nossos cálculos. O programa lê o valor analógico do pino associado ao objeto Sensor. Em Read=(Sensor.read_u16()*3.3)/65535 realizamos a leitura do valor analógico e retornamos um valor inteiro de 16 bits.

A seguir, vamos multiplica esse valor por 3.3 para obter a tensão correspondente, pois temos uma faixa de 0 a 3.3 volts, e dividimos pelo valor máximo representável de 16 bits (65535) para normalizar o valor.

Em Temperature = 27 - (Read - 0.706)/0.001721 o código calcula a temperatura com base na leitura da tensão. A fórmula para o cálculo da temperatura é específica para o sensor ou circuito que está sendo usado.

O valor de 0.706 é um deslocamento, e 0.001721 é um fator de escala específico para essa aplicação. Esses valores devem ser ajustados de acordo com o circuito e o sensor real em uso. Não devemos decorar essas fórmulas, pois elas são disponibilizadas pelo fabricante no datasheet do componente, no nosso caso, o RP2040.

Se formos trabalhar com outros sensores, os devidos fabricantes devem fornecer todas os parâmetros e fórmulas em seus datasheets. Em print(Temperature) o valor calculado da temperatura é impresso no shell e damos um sleep(1) para uma pausa de 1 segundo até a próxima leitura.

Em resumo, esse código lê um valor analógico do ADC, converte esse valor em uma tensão, calcula a temperatura com base na tensão lida e exibe a temperatura calculada no shell. Basta conectarmos nossa placa via usb, enviarmos o código selecionando a placa como interpretador, como foi vimos nas primeiras aulas, e veremos a temperatura, como mostra a Figura 30.

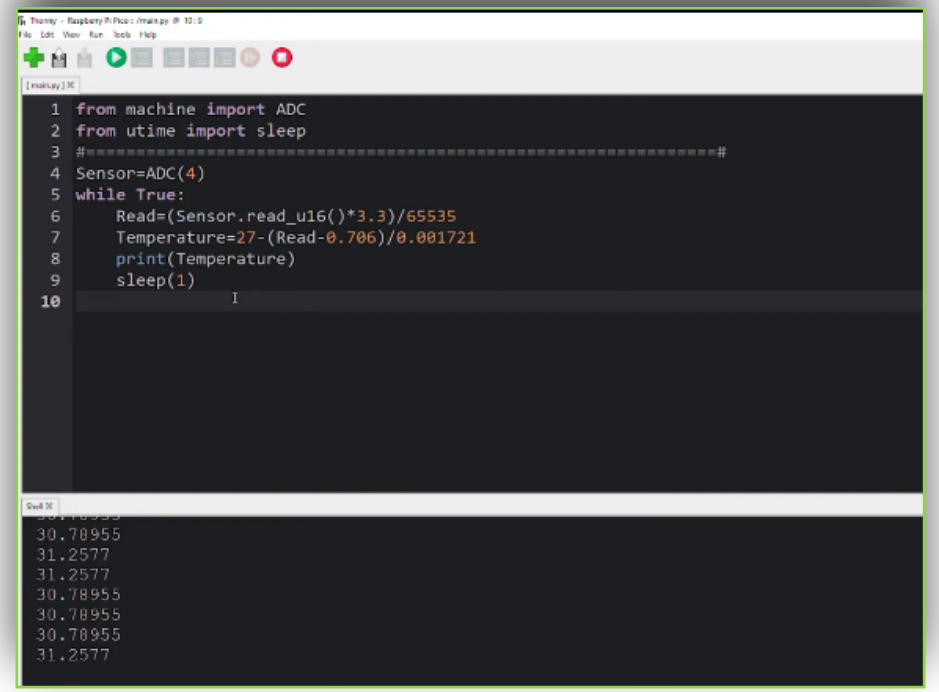


Figura 30: Exibição da temperatura no shell.

Aula 13 - PWM Pulse With Modulation

Imagine que temos um botão para ligar e desligar uma lâmpada, mas em vez de apenas ligar e desligar, poderíamos apertar o botão rapidamente para ajustar quanta luz a lâmpada emite. Vamos pensar nisso como piscar os olhos muito rapidamente. Se você piscar mais vezes, parece que está olhando para algo mais brilhante, mesmo que seus olhos estejam fechados na maior parte do tempo. Da mesma forma, com PWM, enviamos "piscadas" rápidas de eletricidade para um dispositivo, como um motor ou uma lâmpada. Se as "piscadas" forem mais longas, o dispositivo funciona mais forte, e se forem mais curtas, ele funciona mais fraco.

De forma mais técnica, o PWM ou Modulação por Largura de Pulso em português, é uma técnica usada para controlar a quantidade de energia entregue a um dispositivo elétrico. Em vez de simplesmente ligar ou desligar o dispositivo, o PWM ajusta a largura dos pulsos elétricos enviados em intervalos regulares.

Isso significa que, em vez de manter uma tensão constante, o PWM alterna rapidamente entre um nível alto e um nível baixo, criando uma série de pulsos. A proporção do tempo em que o sinal está em nível alto em relação ao tempo total determina a quantidade média de energia entregue ao dispositivo. Isso permite que você controle a velocidade de um motor ou o brilho de um LED, por exemplo, variando a frequência e a proporção do sinal PWM.

Dessa forma, com PWM, podemos ajustar o funcionamento de dispositivos eletrônicos de uma maneira inteligente, controlando o quanto eles trabalham forte ou fraco, sem precisar sempre ligar e desligar completamente. Isso é útil para economizar energia e fazer com que as coisas funcionem do jeito que queremos. Na Figura 31 temos o uso do PWM para controlar o brilho de uma lâmpada.

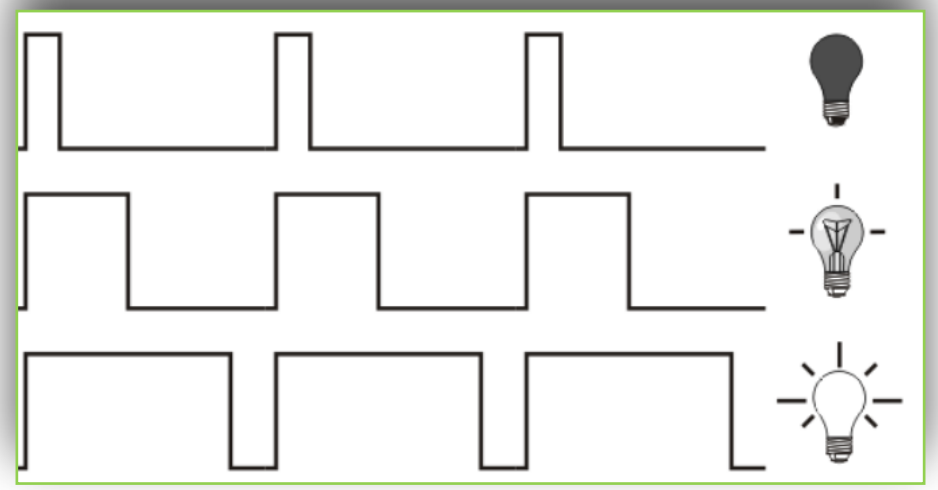


Figura 31: Uso do PWM para controle de brilho.

No contexto do Pulse Width Modulation (PWM), existem alguns parâmetros importantes que podemos ajustar para controlar a forma como o sinal PWM é gerado e seu comportamento. Esses parâmetros variam dependendo do sistema e do dispositivo que está sendo controlado. Aqui estão alguns dos principais parâmetros do PWM:

Frequência (Hz): A frequência do PWM é a taxa na qual os pulsos são gerados. É o número de pulsos por segundo. Uma frequência mais alta geralmente resulta em uma resposta mais rápida do dispositivo, enquanto uma frequência mais baixa pode ser usada para controlar dispositivos que não precisam de uma resposta rápida.

Ciclo de Trabalho (%): O ciclo de trabalho, ou Duty Cycle, muitas vezes representado como uma porcentagem, é a proporção de tempo durante um ciclo de PWM em que o sinal está em nível alto (ligado) em comparação com o tempo total do ciclo. Por exemplo, um ciclo de trabalho de 50% significa que o sinal está ligado metade do tempo e desligado a outra metade.

Resolução: A resolução se refere à precisão com que o ciclo de trabalho do PWM pode ser ajustado. Uma maior resolução permite ajustes mais finos no ciclo de trabalho, o que resulta em um controle mais preciso do dispositivo controlado.

Tensão de Referência: Em alguns sistemas, você pode ajustar a tensão de referência que determina o nível de tensão em relação ao qual o ciclo de trabalho é medido. Assim podemos ajustar a tensão de nível baixo em outro

valor, como 2V e o nível alto em 7V, por exemplo, assim o sinal nunca chegará a 0V.

Polaridade: A polaridade do PWM refere-se à ordem em que os níveis alto e baixo do sinal são alternados. Isso pode afetar a forma como um dispositivo responde ao sinal, já que podemos ter um sinal que vai de 0V até -5V, por exemplo.

Tempo de Subida e Descida: Esses parâmetros se referem à rapidez com que o sinal PWM muda de nível alto para baixo e vice-versa. Eles podem influenciar a eficiência e a resposta do dispositivo controlado.

Amplitude: A amplitude do sinal PWM pode afetar a faixa de controle de um dispositivo. Em alguns sistemas, o sinal PWM pode ter diferentes níveis de tensão, afetando como o dispositivo responde, como por exemplo, sinais que vão até 15V, não devem ser usados para controle de carga de 5V.

Os parâmetros podem variar dependendo do sistema ou dispositivo específico que está sendo controlado. Ajustar corretamente esses parâmetros é fundamental para alcançar o comportamento desejado no controle de dispositivos usando PWM. Na Figura 32 temos alguns paramentos do PWM e alguns valores para Duty Cycle.

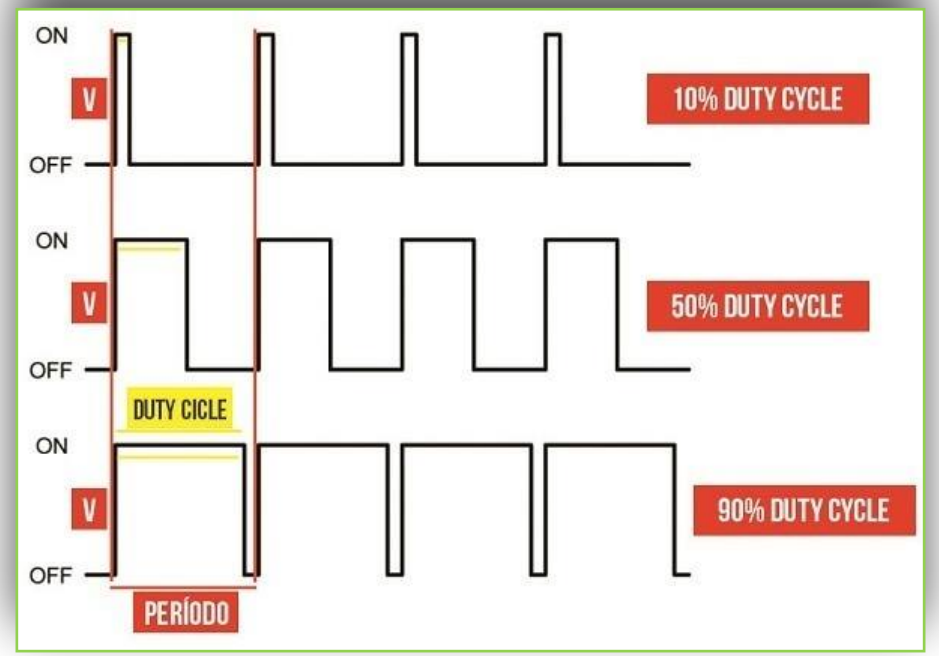


Figura 32: Exemplos de valores para Duty Cycle.

Embora o Pulse Width Modulation (PWM) seja uma técnica bastante útil e versátil para controlar dispositivos eletrônicos, também apresenta algumas desvantagens e limitações:

Ruído Eletromagnético (EMI): Em sistemas onde o PWM é usado para controlar dispositivos de alta potência, como motores, inversores ou fontes de alimentação, a rápida alternância entre os níveis alto e baixo de tensão

pode gerar ruído eletromagnético, causando interferência em outros componentes eletrônicos próximos.
Aquecimento: O uso de PWM pode levar a variações rápidas na tensão ou corrente aplicada aos dispositivos, o que pode resultar em perdas de energia e aquecimento excessivo em alguns componentes, especialmente se eles não forem projetados para operar com frequências altas.
Resposta Transiente: Alguns dispositivos podem não responder bem a mudanças rápidas no ciclo de trabalho do PWM. Isso pode resultar em comportamento imprevisível ou instável, especialmente quando o ciclo de trabalho é modificado rapidamente.
Requisitos de Hardware: A implementação eficaz do PWM pode exigir circuitos ou microcontroladores específicos, o que pode aumentar a complexidade do sistema e o custo.
Requisitos de Filtragem: Em alguns casos, é necessário usar filtros adicionais para suavizar o sinal PWM e torná-lo mais adequado para certos dispositivos. Isso pode adicionar complexidade ao projeto.
Efeitos Visuais Indesejados: Em aplicações de iluminação ou exibição, a variação rápida de intensidade luminosa ou brilho usando PWM pode ser perceptível ao olho humano e causar cintilação, o que pode ser indesejado em algumas situações.
Limitações de Resolução: Sistemas PWM com baixa resolução podem ter dificuldade em fornecer ajustes finos em dispositivos que requerem um controle muito preciso.
Complexidade de Projeto: Projetar sistemas que usam PWM de maneira eficaz e otimizada pode ser complexo, especialmente em aplicações de alta potência ou alta frequência.
Compatibilidade Eletromecânica: Alguns dispositivos mecânicos, como motores de baixa qualidade ou comutadores de relé, podem não responder bem a variações rápidas no ciclo de trabalho do PWM.

Apesar dessas desvantagens, o PWM ainda é amplamente utilizado em muitas aplicações de controle eletrônico, e muitas vezes os benefícios superam as limitações quando aplicados corretamente e considerando as necessidades específicas do sistema.

13.1 – Controle de Duty Cycle com Potenciômetro

Podemos realizar controles de atuadores com o ajuste do Duty Cycle a partir de um potenciômetro. O que faremos, é uma leitura analógica do ADC onde o potenciômetro estará ligado, e transformar em um valor de Duty proporcional. vamos digitar o nosso código e entender melhor.

```
from machine import Pin
from machine import ADC
from machine import PWM
from utime import sleep
```

```
#======================Variable================================#
Potentiometer=ADC(27)
PwmOut=PWM(Pin(15))
PwmOut.freq(1000)
#=======================Main=================================#
while True:
    PwmOut.duty_u16(Potentiometer.read_u16())
    sleep(0.1)
    print(PwmOut)
```

AS linhas iniciais já foram explicadas nos projetos anteriores, o que temos de novo é a importação do PWM, que nos permite trabalhar e ajustar o sinal. Em Potentiometer=ADC(27) temos a criação de um objeto ADC e associamos ao pino 27. Isso permite que leiamos valores analógicos desse pino, que está conectado o nosso potenciômetro e será usado para ajustar a intensidade do sinal PWM.

```
PwmOut=PWM(Pin(15))
PwmOut.freq(1000)
```

Aqui, o objeto PWM é criado e associado ao pino 15, ou seja, o pino 15 será onde o sinal PWM será gerado. A frequência do sinal PWM é definida como 1000 Hz (1 kHz). Isso significa que o sinal PWM terá 1000 ciclos por segundo.

```
while True:
    PwmOut.duty_u16(Potentiometer.read_u16())
    sleep(0.1)
    print(PwmOut)
```

Em PwmOut.duty_u16(Potentiometer.read_u16()), definimos o ciclo de trabalho do sinal PWM. O valor do ciclo de trabalho é lido do potenciômetro usando Potentiometer.read_u16(), que retorna um valor entre 0 e 65535 (16 bits), que controla a intensidade do sinal. Em sleep(0.1), temos a espera de 100 milissegundos antes de realizar a próxima iteração do loop. Em print(PwmOut) o estado atual do objeto PWM (PwmOut) é impresso no shell. Portanto, nosso código lê o valor analógico do potenciômetro e converte-o em um ciclo de trabalho PWM e, em seguida, gera um sinal PWM com a intensidade correspondente. O circuito é visto na Figura 33.

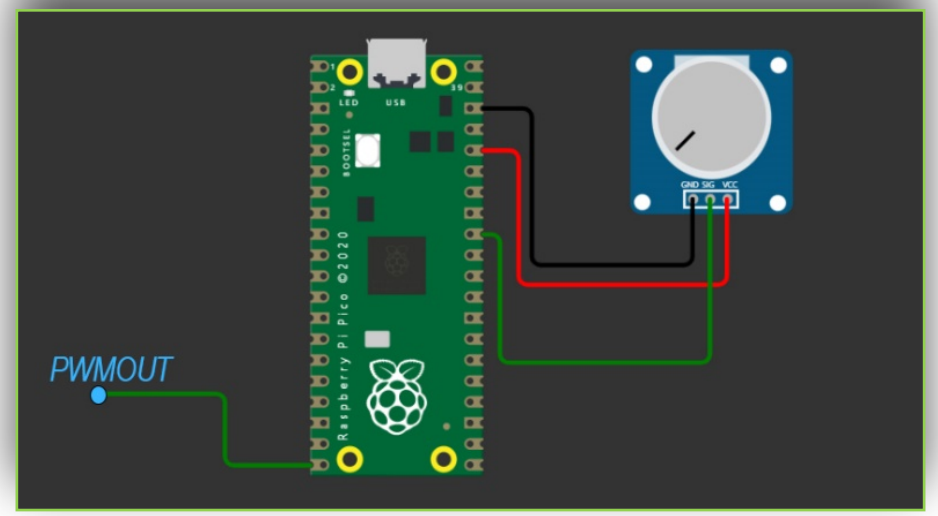


Figura 33: Circuito do controle PWM.

Esse projeto poderá ser usado para controlar o brilho de um led, conectado a saída PWM e com seu resistor. No nosso canal no YouTube temos a demonstração prática com testes e visualização no osciloscópio, disponível em: https://www.youtube.com/@Engenhariaentendida. Na Figura 34 temos os teste com osciloscópio e o led.

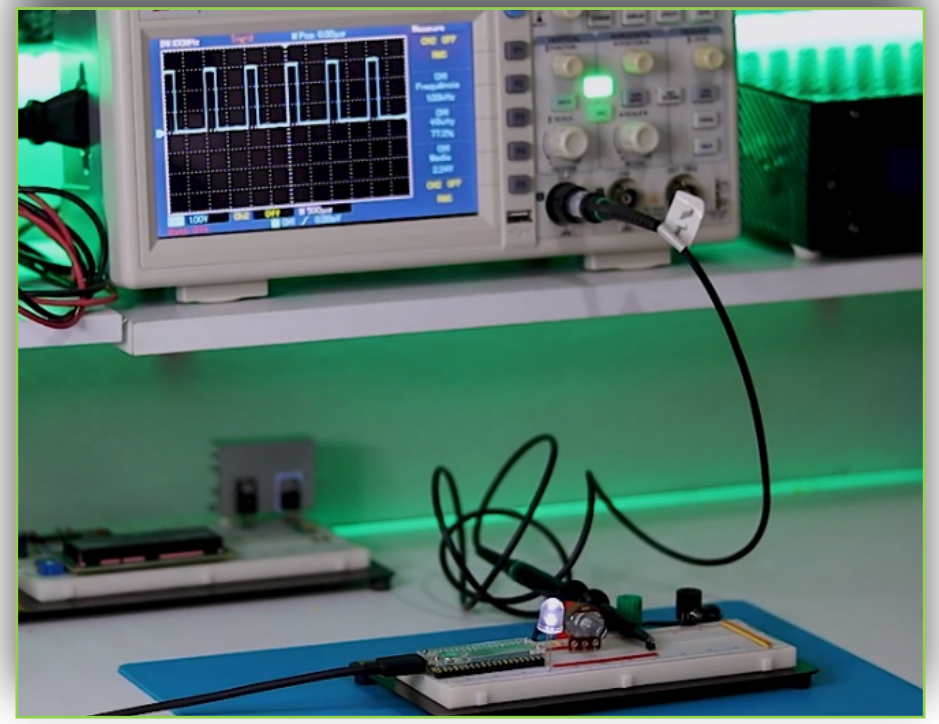


Figura 34: Bancada de testes.

Aula 14 - Display LCD e I2C

Quando precisamos receber ou enviar dados com equipamento e máquinas, estamos fazendo uso da Interface Homem-Máquina (IHM), que é um ponto de interação entre seres humanos e máquinas, onde os usuários podem controlar, monitorar e interagir com sistemas diversos. Vamos explicar

detalhadamente o conceito usando exemplos de botões, potenciômetro e um display LCD:

Botões: Botões são elementos de entrada simples que os usuários podem pressionar para executar ações específicas em um sistema. Eles são frequentemente usados para iniciar ou parar processos, selecionar opções ou fazer escolhas.
Potenciômetro: Potenciômetro são dispositivos analógicos de entrada que permitem aos usuários ajustar valores continuamente, como volume, brilho ou velocidade. Eles são compostos por um eixo giratório que pode ser ajustado em diferentes posições.
Display LCD (Tela de Cristal Líquido): Displays LCD são telas que exibem informações em formato alfanumérico ou gráfico. Eles são usados para mostrar dados, estados do sistema e outras informações relevantes aos usuários.

Na combinação desses elementos, podemos criar uma IHM mais sofisticada. Por exemplo, em um equipamento de áudio, poderíamos ter um display LCD que mostra informações como a faixa de música atual, um botão para alternar entre as faixas e um potenciômetro para ajustar o volume.

A interface IHM é projetada para tornar as interações entre humanos e máquinas mais intuitivas e eficientes. Isso envolve um design cuidadoso dos elementos visuais, dos controles de entrada e do feedback fornecido ao usuário.

A IHM deve ser fácil de entender e usar, minimizando a curva de aprendizado e proporcionando uma experiência agradável ao usuário. O vamos fazer agora nos nossos projetos, é adicionar um interface de informações visuais, utilizando o famoso display LCD 16x2.

14.1 – Conhecendo o display LCD

Os displays de caracteres LCD são um tipo exclusivo de display que só pode gerar caracteres ASCII de medidas únicas. Usando esses caracteres individuais, podemos formar texto. Se olharmos atentamente para a tela, podemos ver que existem algumas pequenas áreas retangulares compostas por uma grade de 5×8 pixels. Cada pixel pode ser iluminado individualmente, para que possamos gerar caracteres dentro de cada grade.

O número de áreas retangulares define o tamanho do LCD. O LCD mais popular é o 16×2, que possui 2 linhas com 16 colunas retangulares ou caracteres. Claro, existem outros tamanhos como 16x1, 16x4, 20x4 e assim por diante, mas todos funcionam com o mesmo princípio. Além disso, esses LCDs podem ter fundo e cores de texto diferentes. Temos um exemplo Figura 35.

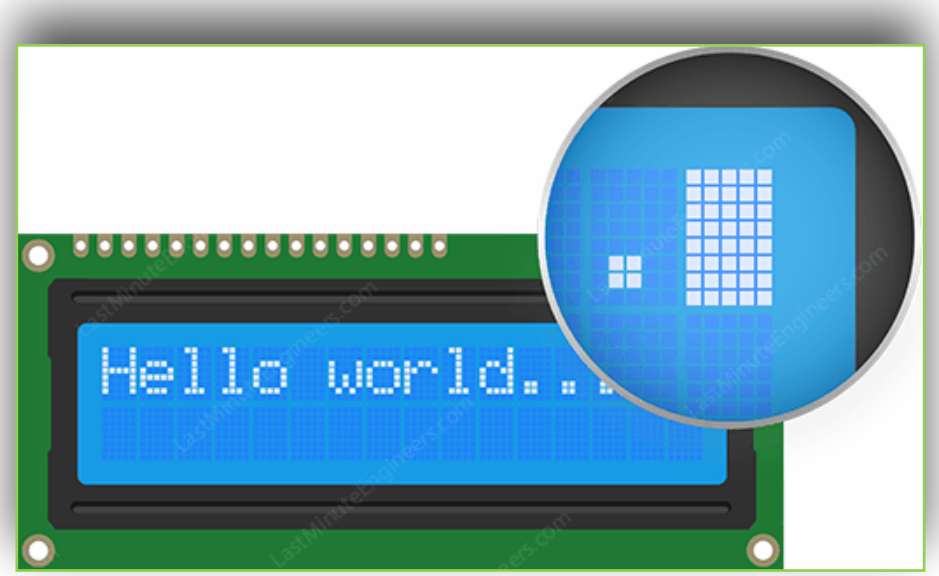


Figura 35: Display LCD 16x2.

O display possui 16 pinos no total, o primeiro da esquerda para a direita é o pino GND. O segundo pino é VCC, ao qual conectamos o pino de 5 volts. O próximo é o pino Vo onde podemos conectar um potenciômetro para controlar o contraste do display.

Em seguida, o pino RS ou pino de seleção de registro é usado para selecionar se deseja enviar comando ou dados para o LCD. Por exemplo, se o pino RS estiver configurado para baixa tensão ou zero volts, enviaremos comandos ao LCD como: colocar o cursor em uma posição específica, limpar o display, desligar o display, etc. Quando o pino RS estiver definido para o estado alto ou 5 volts, enviaremos dados ou caracteres para o LCD.

O próximo é o pino R/W, que seleciona o modo se vamos ler ou escrever no LCD. O modo de gravação é óbvio aqui e é usado para escrever ou enviar comandos e dados para o LCD.

O modo de leitura é usado pelo próprio LCD ao executar um programa, que não precisamos discutir neste tutorial. O pino E, permite escrever no registrador, ou nos próximos 8 pinos de dados (de D0 a D7).

Então quando escrevemos em um registrador enviaremos 8 bits de dados através desses pinos, ou por exemplo se quisermos ver a última letra A maiúscula no display enviaremos 0100 0001 para o registrador conforme a tabela ASCII. Os dois últimos pinos A e K, ou ânodo e cátodo, são para a iluminação LED do display. A figura 36 mostra o pinout.

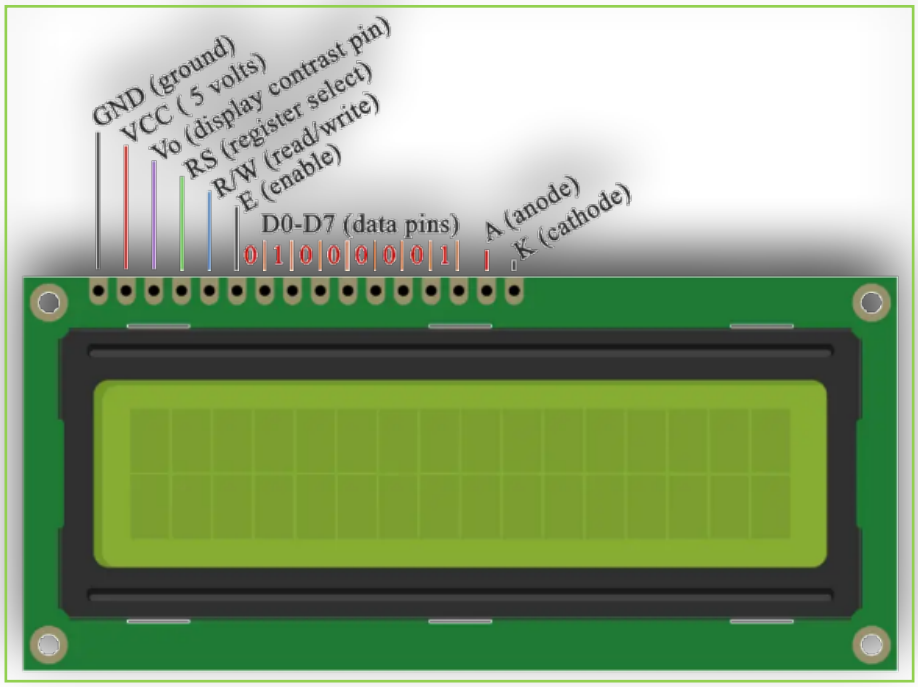


Figura 36: Pinos de conexão do display.

Não precisamos nos preocupar muito com o funcionamento do LCD, já que as bibliotecas cuidam de praticamente tudo. Podemos usar o display no modo de 4 ou 8 bits, que exige o pouco mais de conexões e era bastante utilizado até pouco tempo. Nos dias atuais, temos a comunicação i2c, que irá precisar apenas de 4 fio, sendo dois de alimentação e dois da dados. Vamos entender como essa comunicação funciona.

14.2 – Comunicação I2C

A comunicação I2C (Inter-Integrated Circuit) é um protocolo de comunicação serial síncrona que permite a troca de dados entre dispositivos eletrônicos. Foi desenvolvido pela Philips nos anos 1980 como uma maneira de permitir que vários dispositivos se comuniquem entre si usando apenas algumas linhas de sinal. A comunicação I2C envolve dois tipos principais de dispositivos: mestre (Master) e escravo (Slave). O dispositivo mestre é aquele que controla o fluxo da comunicação, enquanto os dispositivos escravos respondem às solicitações do mestre. O protocolo é bidirecional, o que significa que os dispositivos podem enviar e receber dados. A comunicação I2C é baseada em duas linhas principais:

Linha de Dados (SDA - Serial Data Line): Esta linha é usada para a transferência de dados entre o mestre e os escravos. Os dados são transmitidos em formato serial, um bit por vez. Tanto o mestre quanto os escravos podem conduzir essa linha, mas o mestre geralmente inicia e termina a comunicação.
Linha de Clock (SCL - Serial Clock Line): Esta linha é usada para sincronizar a transmissão de dados entre o mestre e os escravos. O sinal de clock é

gerado pelo mestre e determina quando os bits de dados na linha SDA devem ser lidos ou gravados.

O protocolo I2C possui endereços que são usados para identificar os dispositivos escravos na rede. Cada dispositivo escravo tem um endereço único, o que permite que o mestre selecione qual dispositivo deseja se comunicar. O fluxo básico da comunicação I2C é o seguinte:

1. O mestre envia um sinal de START (início) na linha SDA.
2. O mestre envia o endereço do dispositivo escravo que deseja acessar, juntamente com o bit de leitura/escrita.
3. O dispositivo escravo com o endereço correspondente responde com um sinal de ACK (acknowledge) na linha SDA.
4. A comunicação de dados começa, com os bits sendo transferidos na linha SDA e sincronizados pelo sinal de clock na linha SCL.
5. Após a transferência de dados, o mestre pode enviar um sinal de STOP (parada) na linha SDA para encerrar a comunicação.

A comunicação I2C oferece várias vantagens que a tornam uma escolha popular em muitas aplicações eletrônicas:

Simplicidade: O protocolo I2C é relativamente simples de entender e implementar. Ele requer apenas duas linhas de sinal (SDA e SCL), tornando a fiação e o design de circuitos mais simples em comparação com outros protocolos de comunicação.
Uso Eficiente de Pinos: Como mencionado, a comunicação I2C requer apenas duas linhas para transferência de dados e sincronização, o que economiza recursos de hardware em comparação com protocolos que exigem mais linhas de sinal.
Endereçamento de Dispositivos: O protocolo I2C utiliza endereços de dispositivos, permitindo que vários dispositivos compartilhem as mesmas linhas de comunicação. Isso é especialmente útil em sistemas com vários componentes, como sensores, atuadores e dispositivos de armazenamento.
Capacidade Multi-Mestre: O I2C suporta a operação de vários mestres na mesma rede. Isso significa que vários dispositivos mestres podem controlar diferentes partes da comunicação, possibilitando uma hierarquia mais complexa em sistemas interconectados.
Transmissão Bidirecional: A comunicação I2C permite que os dispositivos transmitam e recebam dados na mesma linha, facilitando a troca de informações de ida e volta entre dispositivos mestres e escravos.
Baixo Consumo de Energia: O protocolo I2C é conhecido por seu baixo consumo de energia. Isso é particularmente vantajoso para dispositivos alimentados por baterias ou energia limitada.
Ampla Aceitação: O I2C é amplamente adotado e suportado por muitos fabricantes de chips e componentes eletrônicos. Isso facilita a integração de diferentes dispositivos em um sistema.

Compatibilidade com Diferentes Dispositivos: O protocolo I2C não é restrito a um tipo específico de dispositivo. Ele pode ser usado para comunicação entre uma ampla gama de componentes, como sensores, atuadores, memórias, displays, conversores analógico-digitais, entre outros.
Transferência de Dados Serial: A comunicação I2C é baseada em transferência serial de dados, o que é útil para evitar problemas de temporização em sistemas mais complexos.
Larga Escala de Velocidades: O protocolo I2C suporta diferentes velocidades de comunicação, permitindo ajustar a taxa de transferência de acordo com as necessidades da aplicação.

O protocolo I2C é usado em uma variedade de aplicações, como comunicação entre sensores, displays, memórias EEPROM, componentes de áudio, entre outros. Ele é amplamente adotado devido à sua simplicidade, eficiência e baixo uso de recursos de hardware. Note que não precisamos configurar cada dado que vamos enviar ou quando iniciar a comunicação, as bibliotecas são prontas para isso já. A Figura 37 temos o diagrama de funcionamento dessa comunicação.

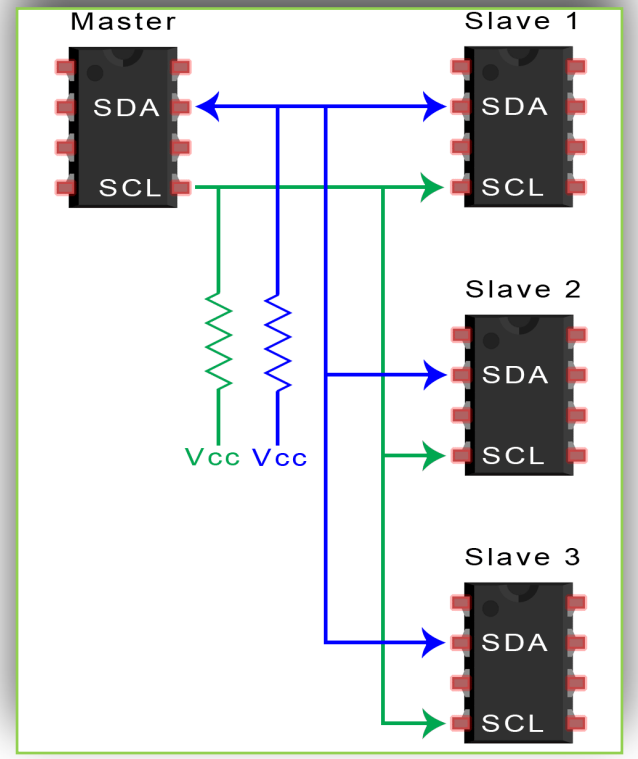


Figura 37: Exemplo de funcionamento I2C.

No entanto, é importante notar que o I2C também possui algumas limitações, como a velocidade de comunicação relativamente mais lenta em comparação com protocolos mais modernos, e pode ser sensível a interferências elétricas em ambientes ruidosos. Para realizarmos a conexão entre o display via I2C, podemos adotar um módulo que faça essa adaptação, visto na Figura 38.

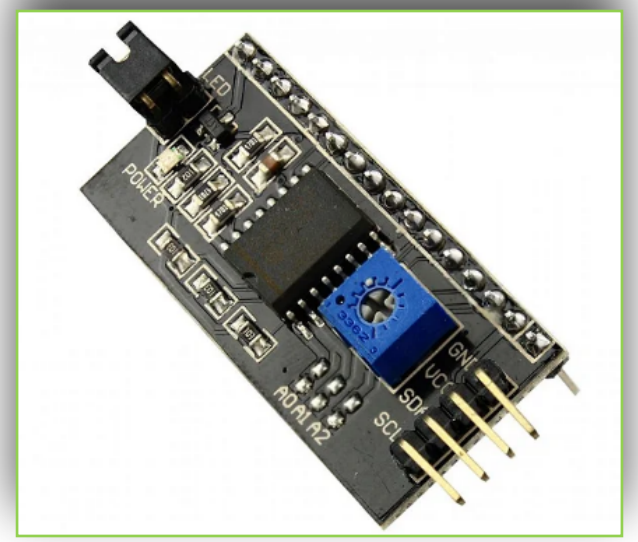


Figura 38: Módulo I2C display LCD.

Existem alguns modelos de display, como o OLED SSD1306, que já tem vem com comunicação I2C, como mostra a figura 39.

Figura 39: Display OLED SSD1306 128x64.

No caso do display 16x2, vamos usar nosso módulo. Esses módulos são frequentemente usados para simplificar a conexão e a comunicação entre um microcontrolador ou microprocessador e um display, pois reduzem a quantidade de pinos necessários para a comunicação. Em vez de precisar de múltiplos pinos para controlar diferentes aspectos do display, como controle de dados, controle de enable, etc., o módulo I2C simplifica isso ao usar apenas duas linhas para toda a comunicação, como já vimos. Temos o exemplo da montagem na Figura 40.

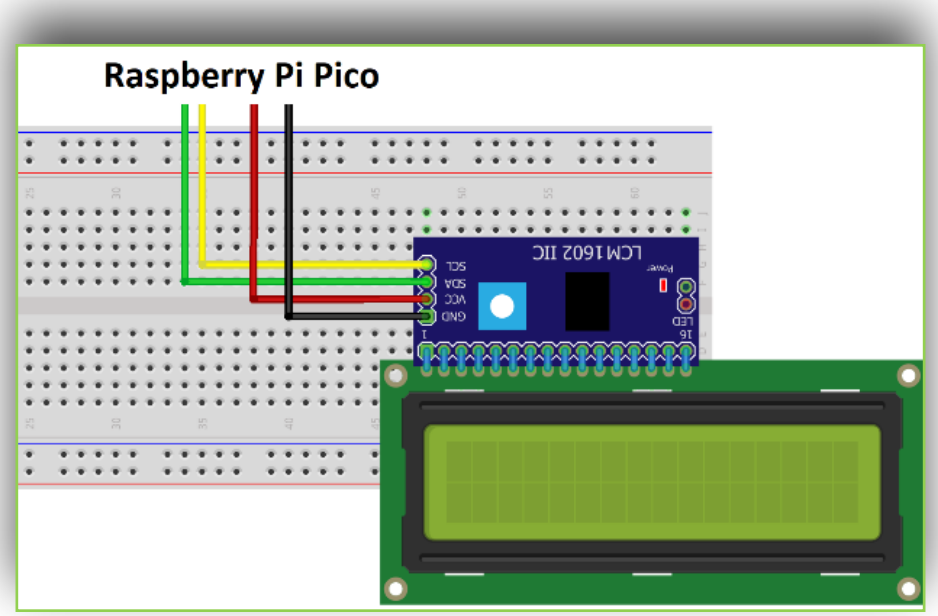


Figura 40: Conexão com módulo e display.

14.3 – Código básico I2C

Vamos agora digitar nosso código para teste:

```
from machine import Pin
from machine import I2C
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd
#============================Variable==========================#
Address = 0x27
Rows = 2
Cols = 16
i2cConfig=I2C(0,sda=Pin(0),scl=Pin(1),freq=40000)
lcd = I2cLcd(i2cConfig,Address,Rows, Cols)
#============================Main==============================#
lcd.move_to(2,0)
lcd.putstr("Hello World!")
lcd.move_to(1,1)
lcd.putstr("Eng. entendida")
```

Primeiro, as bibliotecas necessárias são importadas:

Pin: Para configurar os pinos do microcontrolador.
I2C: Para configurar a comunicação I2C.
lcd_api: Uma API para controlar o LCD.
pico_i2c_lcd: Uma implementação específica para I2C de um LCD para o Raspberry Pi Pico, ou seja, um conjunto de funções e métodos fornecidos por uma biblioteca ou pacote de software que permite a um programador interagir e controlar um display LCD de forma simplificada, sem precisar lidar diretamente com os detalhes de baixo nível da comunicação com o display.

Após isso, o endereço I2C do display é definido como 0x27(em hexadecimal). Esse é o endereço utilizado para a comunicação I2C com o display, pois como foi visto, precisamos saber em qual endereço está o periférico que queremos nos comunicar. Esse é o endereço padrão para o esse módulo, mas podemos exibir no shell o endereço caso quiséssemos confirmar isso, ou caso tenhamos um endereço diferente para nosso módulo. fazendo uma varredura, podemos usar o código a seguir para fazer isso, bastando enviar e aguardar a mensagem no shell, como mostra a Figura 41.

```
import machine
sda=machine.Pin(0)
scl=machine.Pin(1)
i2c=machine.I2C(0,sda=sda, scl=scl, freq=400000)
print('I2C address:')
print(i2c.scan(),' (decimal)')
print(hex(i2c.scan()[0]), ' (hex)')
```

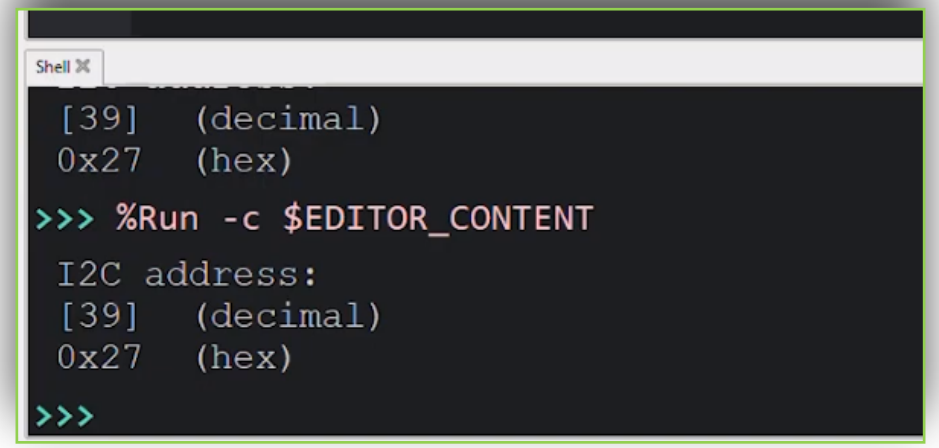


Figura 41: Mostrando o endereço do módulo I2C.

Continuando com o nosso código, o número de linhas (Rows) e colunas (Cols) do display são definidos como 2 e 16, respectivamente. A configuração I2C é realizada usando i2cConfig=I2C(0,sda=Pin(0),scl=Pin(1),freq=40000). Isso cria uma instância do objeto I2C no barramento 0, ou o canal 0, pois a placa possui mais de um canal de comunicação I2C. No nosso caso, estamos usando o canal 0. Estamos usando o pino 0 como SDA (linha de dados) e o pino 1 como SCL (linha de clock). A frequência de comunicação é definida como 40000 Hz. Uma instância do objeto I2cLcd é criada, passando a configuração I2C, o endereço do display, o número de linhas e o número de colunas, em lcd = I2cLcd(i2cConfig,Address,Rows, Cols). Agora, para iniciarmos a exibição da mensagem, precisamos definir a posição de cada palavra, vamos posicionar o cursor do display para a posição (2, 0) usando o método move_to(2, 0), ou seja, coluna 2 e na linha 0. A mensagem "Hello World!" é escrita no display usando putstr("Hello World!").Estamos enviando um variável do tipo string que contém nosso texto. Uma string é uma

sequência de caracteres. Ela é uma unidade básica de texto em programação. Os caracteres podem ser letras, números, símbolos de pontuação e outros caracteres especiais. A função putstr(string) é comumente encontrada em bibliotecas de controle de displays, especialmente em displays de caracteres, como LCDs alfanuméricos. Essa função é utilizada para escrever uma sequência de caracteres (uma string) no display, começando na posição atual do cursor. Temos:

putstr: É o nome da função que escreve uma string no display.
string: É o argumento que você passa para a função. É a sequência de caracteres que você deseja exibir no display.

O cursor do display é movido para a posição (1, 1) usando move_to(1, 1). A mensagem "Eng. entendida" é escrita no display usando putstr("Eng. entendida") novamente. Resumidamente, o nosso código configura o display LCD, movimenta o cursor para diferentes posições e escreve mensagens nessas posições específicas. O circuito é mostrado na Figura 42 e o teste de bancada que realizamos no nosso canal é mostrado na Figura 43.

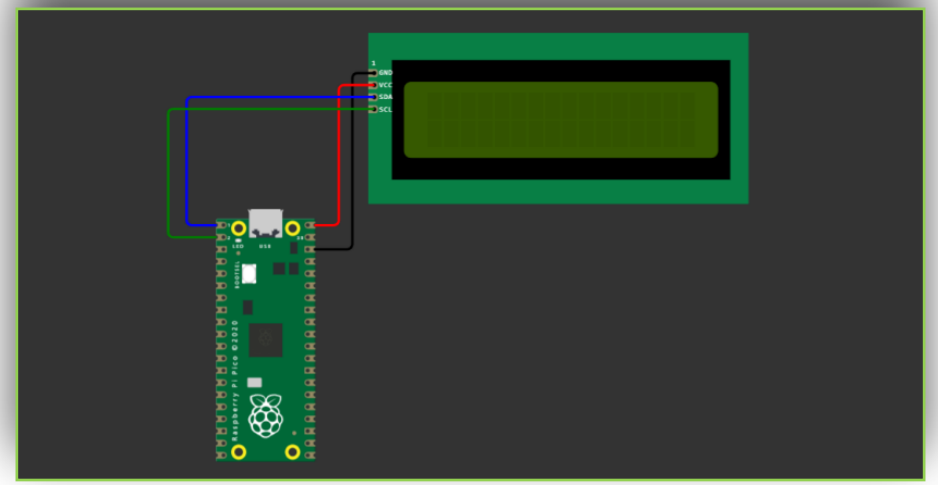


Figura 42: Esquema do circuito.

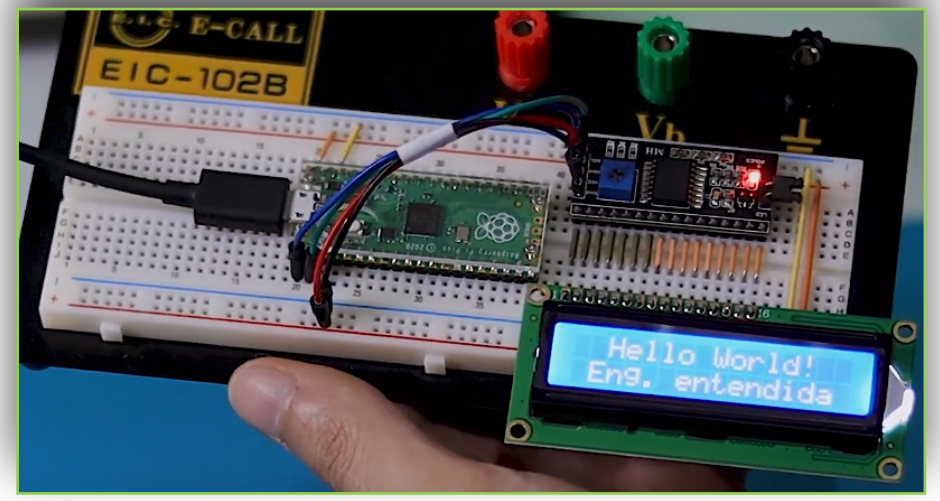


Figura 43: Teste de bancada.

Aula 15 - Sensor Ultrassônico

O sensor ultrassônico é um dispositivo que utiliza ondas sonoras de alta frequência, acima do limite de audição humana (geralmente acima de 20 kHz), para medir distâncias ou detectar objetos. Ele funciona com base no princípio de eco ou reflexão das ondas sonoras, da seguinte forma:

Emissão de Ondas Sonoras: O sensor emite um pulso de ondas sonoras ultrassônicas em direção a um objeto ou superfície.
Reflexão das Ondas: As ondas sonoras atingem o objeto e se refletem de volta em direção ao sensor. A velocidade do som é conhecida, então o sensor calcula o tempo que leva para as ondas retornarem.
Medição do Tempo de Voo: O sensor mede o intervalo de tempo entre a emissão das ondas e o recebimento do eco refletido. Isso é conhecido como "tempo de voo" das ondas sonoras.
Cálculo da Distância: Usando a velocidade do som no ar, o sensor calcula a distância até o objeto.
Saída de Dados: A distância calculada é então fornecida como saída pelo sensor. Alguns sensores ultrassônicos também podem fornecer outras informações, como a intensidade do sinal refletido.

Esses sensores são comumente usados em aplicações como medição de distância, detecção de obstáculos, controle de robôs, sistemas de estacionamento automático em carros, sistemas de segurança, entre outros.

No entanto, é importante notar que esses sensores podem ser afetados por fatores ambientais, como vento, umidade e materiais que absorvem ou refletem o som de maneira diferente, o que pode influenciar a precisão das medições.

O sensor mais comum e o que vamos utilizar, é o HC-SR04, que vamos usar no projeto dessa aula. A Figura 44 mostra as conexões e como ele funciona.

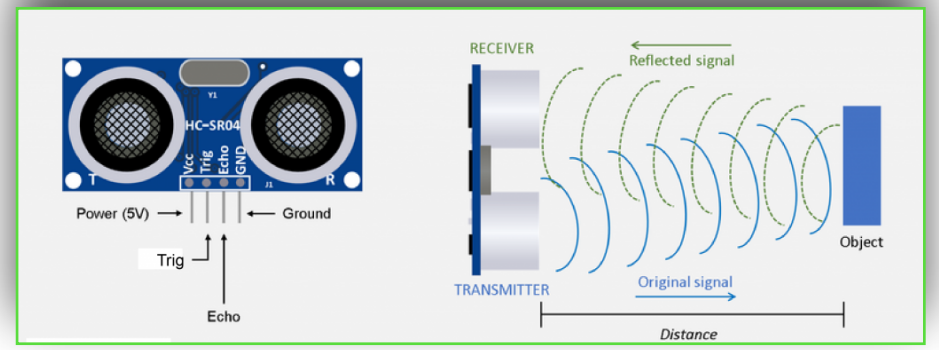


Figura 44: HC-SR04 e seu funcionamento.

Geralemente, o HC-SR04 possui quatro pinos principais: VCC, Trig, Echo e GND. Vamos entender cada um:

VCC (Voltage Common Collector): Este pino fornece a alimentação ao sensor. Geralmente é conectado a uma fonte de energia de +5V.
Trig (Trigger): O pino Trig é usado para iniciar o pulso de ultrassom. Para ativar a medição de distância, você precisa enviar um pulso curto (geralmente de pelo menos 10 microssegundos) para esse pino. O pulso inicia a emissão das ondas ultrassônicas.
Echo: O pino Echo é onde o sensor envia de volta o sinal de eco. Esse pino produz um pulso de duração proporcional ao tempo que as ondas levaram para percorrer o caminho até o objeto e retornar ao sensor.
GND (Ground): Este pino é a referência de terra para o sensor e deve ser conectado ao terra (0V) do sistema.

Quando enviamos um pulso para o pino Trig, o sensor emite uma série de pulsos ultrassônicos em alta frequência. Esses pulsos se propagam em direção ao objeto à sua frente.

Quando encontram um objeto, são refletidos de volta ao sensor. O pino Echo do sensor recebe o sinal refletido e produz um pulso elétrico cuja duração é proporcional ao tempo que as ondas levaram para viajar até o objeto e voltar.

Você pode medir a duração desse pulso utilizando um microcontrolador ou outro dispositivo eletrônico, como um Arduino, ESP32, e Claro, nossa placa Raspberry Pi Pico. A partir dessa duração, você pode calcular a distância usando a fórmula:

$$Distância = time / 58$$

$$time = EchoON - EchoOFF$$

O valor $time$ será o período que o pino Echo ficou em nível alto menos o período que ele ficou em nível baixo. A distância medida é geralmente em linha reta entre o sensor e o objeto. Portanto, se houver objetos obstruindo o caminho (por exemplo, uma mesa em que o sensor esteja montado), o sensor pode detectar o objeto obstruindo como o ponto mais distante.

15.1 – Trena digital com HC-SR04

Nossa trena digital utiliza o princípio básico do sensor ultrassônico para medir distâncias e exibir o resultado em um display. Vamos entender o código abaixo:

```
from machine import Pin
from machine import I2C
from lcd_api import LcdApi
from pico_i2c_lcd import I2cLcd
import utime
```

```
#============================Display=========================#
Address = 0x27
Rows = 2
Cols = 16
i2cConfig=I2C(0,sda=Pin(0),scl=Pin(1),freq=40000)
lcd = I2cLcd(i2cConfig,Address,Rows, Cols)
#============================Sensor==========================#
Trigger = Pin(3,Pin.OUT)
Echo = Pin(2,Pin.IN)
#============================Main============================#
while True:
    Trigger.value(1)
    utime.sleep_us(10)
    Trigger.value(0)
    while Echo.value()==0:
        Off=utime.ticks_us()
    while Echo.value()==1:
        On=utime.ticks_us()
        distance=(On-Off)/58
    lcd.move_to(1,0)
    lcd.putstr("Digital Scale!")
    lcd.move_to(1,1)
    lcd.putstr("In CM: "+str(distance)+"")
```

Nosso código é uma combinação de controle de display LCD e medição de distância utilizando um sensor ultrassônico. Vamos entender linha a linha, até mesmo as linhas já conhecidas, para relembrarmos:

-from machine import Pin: Importa a classe Pin do módulo machine. Essa classe é usada para controlar os pinos de entrada e saída no dispositivo.
-from machine import I2C: Importa a classe I2C do módulo machine. Essa classe é usada para comunicação I2C (Inter-Integrated Circuit), que é um protocolo de comunicação serial entre dispositivos.
-from lcd_api import LcdApi: Importa a classe LcdApi do módulo lcd_api. Essa classe é uma API (Interface de Programação de Aplicativos) para controlar displays LCD.
-from pico_i2c_lcd import I2cLcd: Importa a classe I2cLcd do módulo pico_i2c_lcd. Essa classe é uma implementação específica para controlar um display LCD via I2C no Raspberry Pi Pico.
-import utime: Importa o módulo utime, que fornece funções relacionadas ao gerenciamento de tempo.
-Address = 0x27: Define o endereço I2C do display LCD. Esse endereço pode variar dependendo do dispositivo.
-Rows = 2 e Cols = 16: Define o número de linhas e colunas do display LCD.

-i2cConfig=I2C(0,sda=Pin(0),scl=Pin(1),freq=40000): Configura uma instância da classe I2C para comunicação I2C. Os argumentos passados incluem o número do barramento (0), os pinos SDA (Serial Data) e SCL (Serial Clock), e a frequência de comunicação (40.000 Hz).
-lcd = I2cLcd(i2cConfig,Address,Rows, Cols): Cria uma instância da classe I2cLcd para controlar o display LCD. Os argumentos incluem a configuração I2C, o endereço do display e o número de linhas e colunas.
-Trigger = Pin(3,Pin.OUT): Configura o pino 3 como saída (OUT) e o atribui à variável Trigger. Esse pino é usado para controlar o sensor ultrassônico.
-Echo = Pin(2,Pin.IN): Configura o pino 2 como entrada (IN) e o atribui à variável Echo. Esse pino é usado para receber os sinais do sensor ultrassônico.
-while True: marca o início do loop principal do programa, que será executado continuamente.
-Trigger.value(0): Define o pino Trigger em nível alto, ativando o sensor ultrassônico.
-utime.sleep_us(10): Aguarda 10 microssegundos, provavelmente para dar tempo para o sensor se estabilizar.
-Trigger.value(0): Define o pino Trigger em nível baixo, desativando o sensor ultrassônico.
-Loops while Echo.value()==0: e while Echo.value()==1: são usados para medir o tempo de ida e volta do sinal ultrassônico para calcular a distância. Esses loops aguardam até que o pino Echo mude de estado.
-Off=utime.ticks_us(): Marca o tempo que o pino Echo está em nível baixo.
-On=utime.ticks_us(): Marca o tempo em que o pino Echo está em nível alto, ou seja, que ele recebeu o sinal refletido no objeto.
-distance=(On-Off)/58: Calcula a distância com base no tempo de ida e volta do sinal ultrassônico, considerando a velocidade do som no ar. O fator de divisão 58 é uma constante que permite converter o tempo em microssegundos para centímetros.

As linhas seguintes movem o cursor do display LCD para as posições desejadas e escrevem mensagens relacionadas à medição de distância, como já vimos. Escrevemos "Digital Scale!" e abaixo temos a string "IN CM:" seguida do valor lido, que está armazenado em distance.

Na Figura 45 temos o esquemático e na Figura 46 temos o teste realizado na bancada, disponível no nosso canal no YouTube.

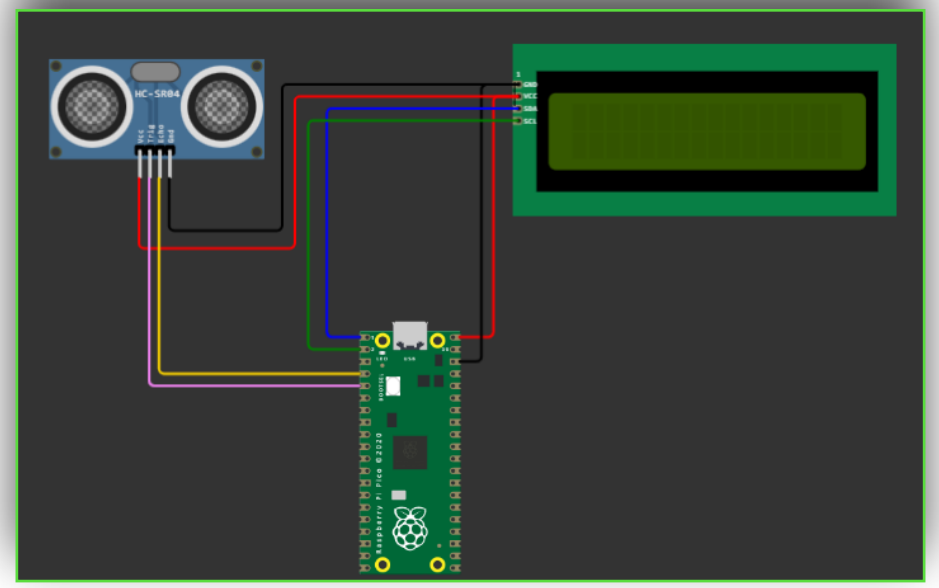


Figura 45: Esquemático da trena digital.

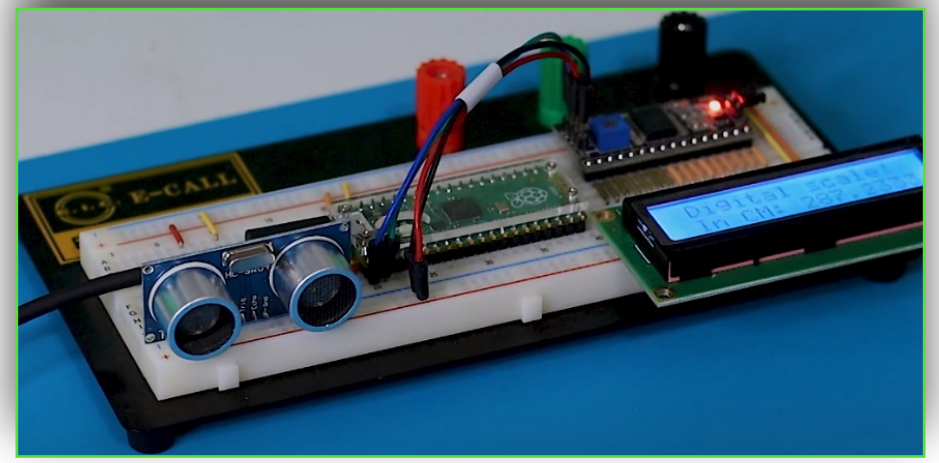


Figura 46: Teste realizado no nosso canal no YouTube.

Aula 16 - Controle de motor DC com PWM

O controle de um motor DC (corrente contínua) por PWM (Modulação por Largura de Pulso) é uma técnica amplamente utilizada para variar a velocidade e a direção de rotação de motores DC. PWM envolve a variação da largura dos pulsos de um sinal para controle de atuadores, como já foi visto, e o motores DC é um deles.

É importante entender como um motor DC funciona. Um motor DC é composto por um rotor (a parte que gira) e um estator (a parte estacionária). Ele opera com base na interação entre um campo magnético fixo (criado no estator) e um campo magnético variável (gerado pelo rotor). Quando uma corrente elétrica é aplicada às bobinas do motor, ela cria um campo magnético no rotor, que interage com o campo magnético do estator e gera torque, fazendo o motor girar.

O controle de velocidade de um motor DC com PWM envolve o uso de pulsos elétricos de largura variável (Duty Cycle) para controlar a quantidade de energia entregue ao motor. Quanto maior a largura dos pulsos, maior a média da corrente elétrica entregue ao motor e, portanto, maior a velocidade do motor. Por outro lado, pulsos mais estreitos resultam em uma velocidade menor.

16.1 – Motores DC

Um motor de corrente contínua (DC) é um dispositivo que converte energia elétrica em movimento rotativo. Ele funciona com base nos princípios do eletromagnetismo e utiliza um campo magnético para gerar torque e fazer com que o motor gire. Na Figura 47 temos um exemplo de dois modelos.

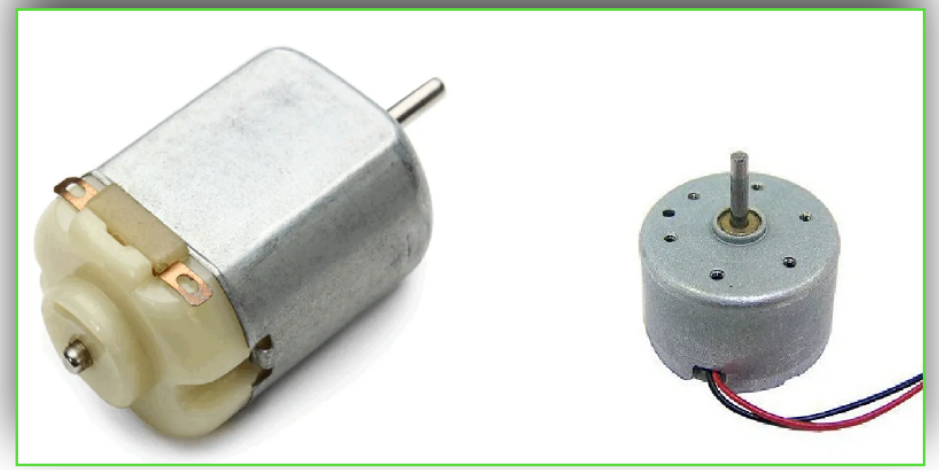


Figura 47: Motores DC simples.

Um motor DC é composto por várias partes principais, incluindo um rotor (a parte que gira), um estator (a parte estacionária) e um comutador. Ele opera usando um ímã permanente ou um eletroímã para criar um campo magnético no interior do motor.

O rotor é uma parte central do motor DC e geralmente é uma bobina de fio de cobre enrolada em um núcleo de ferro (Coils). Esta bobina é montada de forma que possa girar dentro do campo magnético criado pelo estator.

O estator é a parte fixa do motor e contém o ímã permanente ou o eletroímã (Stator). Quando uma corrente elétrica é aplicada ao estator, através das escovas e do comutador (Commutator), ele gera um campo magnético fixo no interior do motor.

O comutador é uma parte crítica do motor DC. Ele é montado no rotor e é composto por segmentos de metal separados por isolantes. As escovas (Brushes), geralmente feitas de grafite ou materiais semelhantes, são pressionadas contra o comutador e fornecem a conexão elétrica entre o rotor e a fonte de energia externa. Na figura 48 temos as partes do motor aberto.

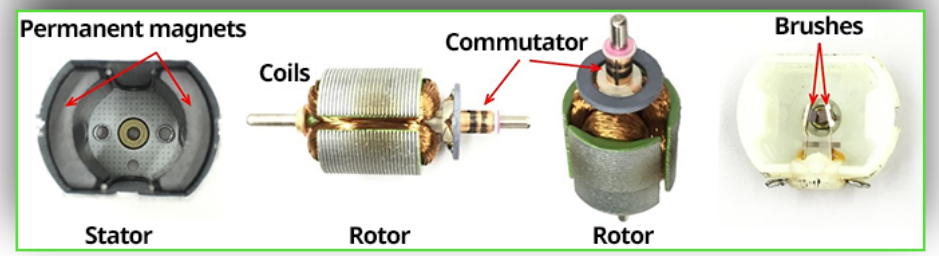


Figura 48: Partes de um motor DC simples.

Quando a corrente elétrica é aplicada ao estator, ele cria um campo magnético fixo. A bobina do rotor, que está conectada ao comutador, fica imersa nesse campo magnético. Devido às leis da eletricidade e do magnetismo, quando a corrente elétrica flui pela bobina do rotor dentro do campo magnético, ela experimenta uma força conhecida como força de Lorentz. Isso cria um torque no rotor, fazendo-o girar.

À medida que o rotor gira, o comutador gira junto com ele. Isso muda a conexão elétrica da bobina do rotor com a fonte de energia, invertendo a direção da corrente elétrica na bobina a cada meia volta (180 graus) do rotor. Essa inversão da corrente elétrica altera a direção do campo magnético gerado pela bobina do rotor, mantendo o motor girando continuamente na mesma direção.

Para controlar a velocidade e a direção do motor DC, podemos variar a tensão aplicada ao estator com o nosso sistema de controle PWM ou inverter a polaridade da tensão para inverter a direção de rotação. Porém, nossa placa não fornece corrente suficiente para controlar nosso motor, podendo causar danos ao pino da placa, precisamos usar então, um transistor.

16.2 – Diodos

O diodo é um dispositivo eletrônico semicondutor que permite a passagem da corrente elétrica somente em um sentido. É composto por um cristal semicondutor de silício ou germânio, com uma junção dividida em duas partes que possuem dopagens distintas e separadas por uma camada de depleção.

A condução da corrente elétrica em um diodo é baseada na diferença de potencial entre as duas regiões do cristal. Quando a tensão aplicada ao diodo é positiva na região do ânodo (lado P) e negativa na região do cátodo (lado N), a corrente elétrica flui facilmente do ânodo para o cátodo. Esta é a polarização direta do diodo. Temos o aspecto na Figura 49.

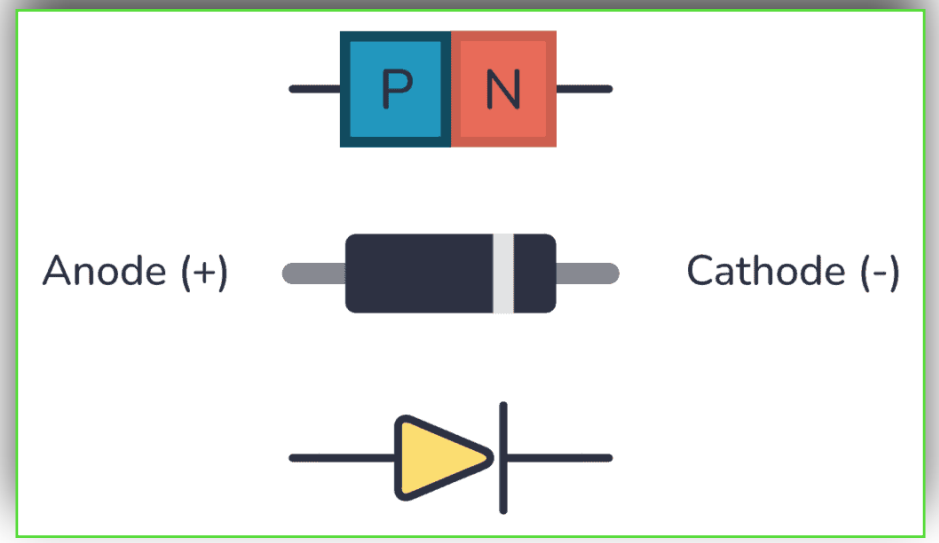


Figura 49: Aspecto de um diodo simples.

Quando a tensão aplicada ao diodo é negativa na região do ânodo e positiva na região do cátodo, a corrente elétrica é muito pequena ou nula. Esta é a polarização reversa do diodo. O diodo é um dispositivo muito versátil e é utilizado em uma ampla variedade de aplicações. Algumas das aplicações mais comuns dos diodos incluem:

Retificadores: Os diodos são usados para converter corrente alternada (AC) em corrente contínua (DC).
Detecção: Os diodos podem ser usados para detectar a presença de sinais elétricos.
Regulação de tensão: Os diodos podem ser usados para regular a tensão de um circuito.
Proteção: Os diodos podem ser usados para proteger circuitos de danos causados por sobrecargas ou sobretensões. Essa será a configuração que precisaremos nesse projeto.

Em um computador ou fontes de alimentação, os diodos são usados para converter a tensão da tomada, que é alternada, em tensão contínua para alimentar os componentes internos, como mostra a Figura 50., isso devido ao diodo só permitir a passagem da corrente em um sentido, como já vimos. Em uma televisão, os diodos são usados para gerar os pixels da tela, que são os LED, sim, o Led é um diodo emissor de luz.

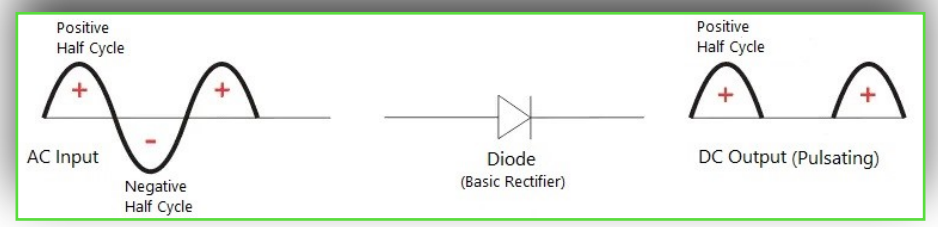


Figura 50: Retificação de uma onda senoidal.

O diodo para proteção de motor DC, também conhecido como diodo de roda livre ou diodo flyback, é um dispositivo que ajuda a proteger o motor de danos causados por picos de tensão reversa. Quando o motor é desligado, ele pode gerar uma tensão reversa na bobina do motor. Esta tensão reversa pode ser alta o suficiente para danificar outros componentes do circuito.

O diodo para proteção de motor DC é conectado em paralelo com a bobina do motor, com o ânodo conectado ao positivo da fonte de alimentação e o cátodo conectado ao negativo da fonte de alimentação, como mostra a Figura 51.

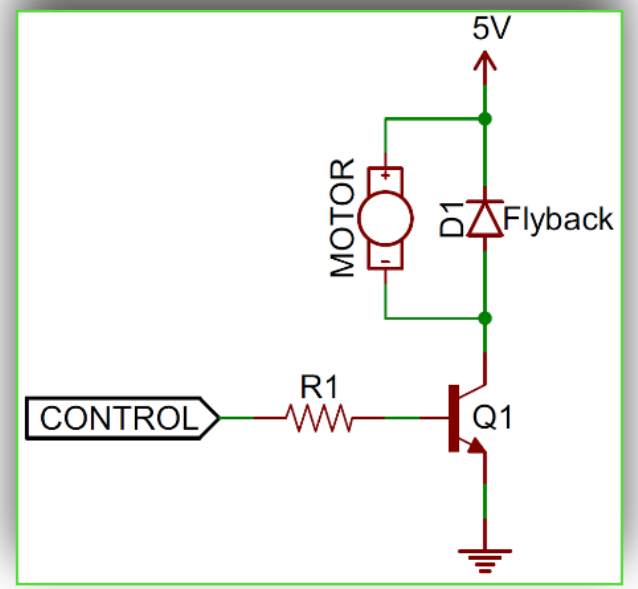


Figura 51: Diodo flyback em um motor DC.

Quando o motor está ligado, a tensão reversa é bloqueada pelo diodo, evitando que ela chegue à bobina do motor. Quando o motor é desligado, a bobina do motor se torna um gerador, gerando uma tensão reversa. Esta tensão reversa é aplicada ao diodo, que permite que a corrente flua do positivo para o negativo da fonte de alimentação. Isso ajuda a dissipar a energia da bobina do motor, evitando que a tensão reversa se torne muito alta.

O diodo para proteção de motor DC é um componente importante em qualquer circuito que use um motor DC. Ele ajuda a proteger o motor de danos causados por picos de tensão reversa, o que pode prolongar a vida útil do motor e evitar a necessidade de reparos caros, além de proteger o circuito de acionamento.

É importante escolher um diodo para proteção de motor DC que tenha uma tensão reversa nominal (VRRM) que seja pelo menos igual à tensão de alimentação do motor. Também é importante escolher um diodo que tenha uma corrente de condução nominal (IRM) que seja pelo menos igual à corrente máxima do motor. No nosso projeto, iremos usar o 1N4001.

16.3 – Transistores

Os transistores são dispositivos eletrônicos que desempenham um papel fundamental na eletrônica moderna e na tecnologia de semicondutores. Eles funcionam como interruptores ou amplificadores de sinais elétricos e são essenciais para o funcionamento de quase todos os dispositivos eletrônicos que usamos hoje, como computadores, smartphones, rádios, TVs etc.

Existem vários tipos de transistores, mas os transistores de junção bipolar (BJT) e os transistores de efeito de campo (FET) são os tipos mais comuns. O transistor de junção bipolar (BJT), que vamos usar nesse projeto, é um dispositivo eletrônico que opera com base nas propriedades de junção PN, que é a interface entre duas camadas semicondutoras de diferentes tipos de condutividade (uma tipo P e outra tipo N ou vice-versa). Existem dois tipos principais de transistores BJT: NPN e PNP, mostrado na Figura 52. Vamos explicar o funcionamento do BJT utilizando um transistor NPN como exemplo:

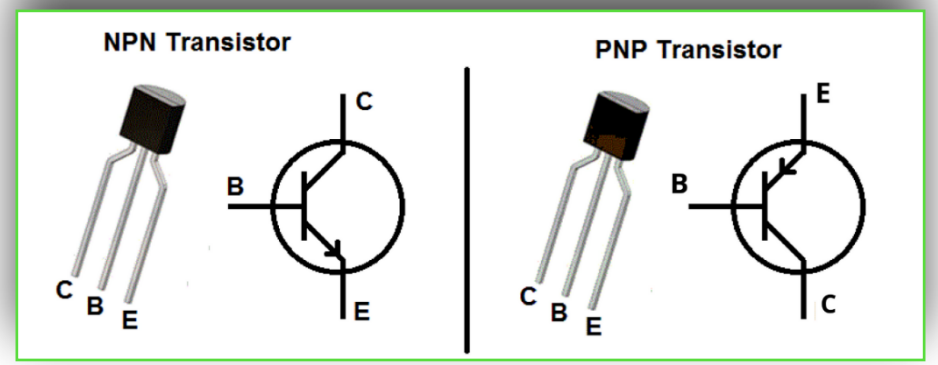


Figura 52: Transistores PNP e NPN.

Um transistor de junção bipolar (BJT) pode operar em três modos principais: corte, saturação e ativa.

Modo de corte: No modo de corte, a tensão base-coletor é maior que a tensão base-emissor. Isso significa que a junção base-coletor está reversamente polarizada e a junção base-emissor está diretamente polarizada. Nesse modo, a corrente do coletor é muito pequena, da ordem de microamperes. O transistor está desligado.

Modo de saturação: No modo de saturação, a tensão base-coletor é menor que a tensão base-emissor. Isso significa que a junção base-coletor está diretamente polarizada e a junção base-emissor está reversamente polarizada. Nesse modo, a corrente do coletor é grande, da ordem de miliamperes ou mais. O transistor está ligado.

Modo ativo: No modo ativo, a tensão base-coletor é ligeiramente menor que a tensão base-emissor. Isso significa que ambas as junções estão parcialmente polarizadas. Nesse modo, a corrente do coletor é proporcional à corrente da base. O transistor pode ser usado como amplificador.

O modo de operação de um BJT é determinado pelas tensões aplicadas aos seus terminais. Em geral, o modo de corte é usado quando o transistor deve ser desligado, o modo de saturação é usado quando o transistor deve ser ligado com alta corrente e o modo ativo é usado quando o transistor deve ser usado como amplificador. Para configurar o transistor de forma correta, devemos ter em mãos o datasheet do componente.

Em resumo, o transistor BJT é um dispositivo de três camadas que atua como um interruptor ou amplificador de corrente. A corrente de base (IB) controla a corrente de coletor (IC), permitindo seu uso em aplicações de amplificação de sinal, chaveamento e controle de corrente em circuitos eletrônicos.

O modo de operação e as características específicas de um transistor BJT podem variar dependendo do tipo (NPN ou PNP) e das condições de polarização. Para nosso projeto, vamos precisar controlar uma carga que consome maior corrente, a partir de uma baixa corrente da nossa placa, o transistor BC548 NPN irá fazer isso, como mostra a Figura 53.

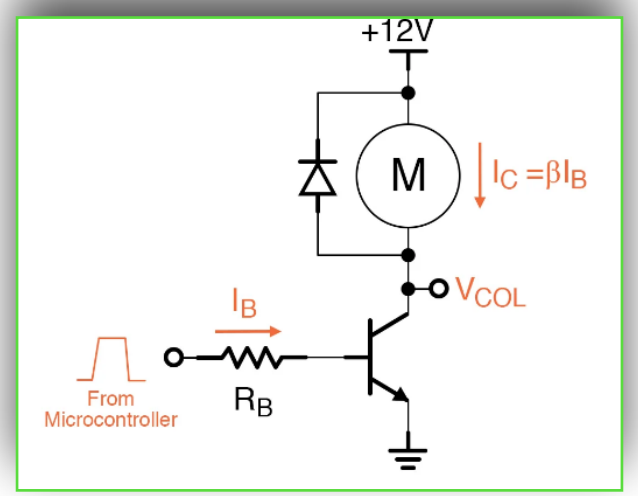


Figura 53: Controle de motor DC com transistor.

16.4 – Código principal e esquemático

Vamos deixar nosso código totalmente comentado, por boas práticas.

```
from machine import Pin              #Biblioteca dos pinos
from machine import I2C              #Biblioteca do I2C
from lcd_api import LcdApi           #Biblioteca de configuração do display
from pico_i2c_lcd import I2cLcd #Biblioteca de configuração do display
from machine import PWM              #Biblioteca do PWM
from utime import sleep              #Biblioteca do delay
import _thread                       #Biblioteca da thread
#=========================Display=========================#
Address = 0x27                       #Endereço I2C
```

```
Rows = 2                              #Número de linhas
Cols = 16                             #Número de colunas
i2cConfig=I2C(0,sda=Pin(0),scl=Pin(1),freq=40000) #Configurações do I2C
lcd = I2cLcd(i2cConfig,Address,Rows, Cols)       #Configurações do display
#=========================Variable=========================#
global UpButton                       #Variável global para botão Up
global DownButton                     #Variável global para botão Down
UpButton = False                      #Up como falso
DownButton=False                      #Down como falso
PwmOut=PWM(Pin(28))                   #Pino do PWM
PwmOut.freq(1000)                     #Frequência
Up = Pin(3,Pin.IN,Pin.PULL_DOWN)      #Configura botão Up
Down = Pin(2,Pin.IN,Pin.PULL_DOWN)    #Configura botão Down
Value=50                              #Valor inicial do PWM
#=========================Thread=========================#
def Pressed ():                                 #Thread
   global UpButton                              #Chama as variáveis globais
   global DownButton                            #Chama as variáveis globais
   while True:                                  #Loop
      if Up.value()==1:                         #Se botão Up pressionado...
         UpButton=True                          #Variável vira Verdadeira
         sleep(0.2)                             #delay 200mS
      if Down.value()==1:                       #Se botão Down pressionado...
         DownButton=True                        #Variável vira Verdadeira
         sleep(0.2)                             #delay 200mS
_thread.start_new_thread(Pressed,())            #Inicia a thread
#=========================Main=========================#
while True:                                     #Loop principal
   if UpButton==True and Value<95:              #Se Up pressionado e PWM < 95%...
      Value=Value+5                             #Incrementa o Value de 5 em 5
      global UpButton                           #Chama as variáveis globais
      UpButton=False                            #Define Up como falso
   if DownButton==True and Value>10:            #Se Down pressionado e PWM >10%...
      Value=Value-5                             #Decrementa o Value de 5 em 5
      global DownButton                         #Chama as variáveis globais
      DownButton=False                          #Define Down como falso
   ValuePwm=Value*650                           #Converte valor do PWM para 16 bits
   PwmOut.duty_u16(ValuePwm)                    #Aplica Duty da saída PWM
   lcd.move_to(1,0)                             #Movimenta o cursor
   lcd.putstr("Motor Control")                  #Imprime na tela
   lcd.move_to(1,1)                             #Movimenta o cursor
   lcd.putstr("PWM in: "+str(Value)+"%")        #Imprime 'Value' no display
```

No geral, nosso código é um exemplo de controle de motor com interface de usuário usando botões e um display LCD. Em um botão temos o incremento do Duty Cycle, e no outro, o decremento, assim, ajustando a velocidade do motor. Também demonstramos como criar threads para monitorar botões de forma paralela e exibimos todas as informações no display.

Na Figura 54 temos o esquemático e na Figura 55 temos os testes realizados na bancada no nosso canal no YouTube.

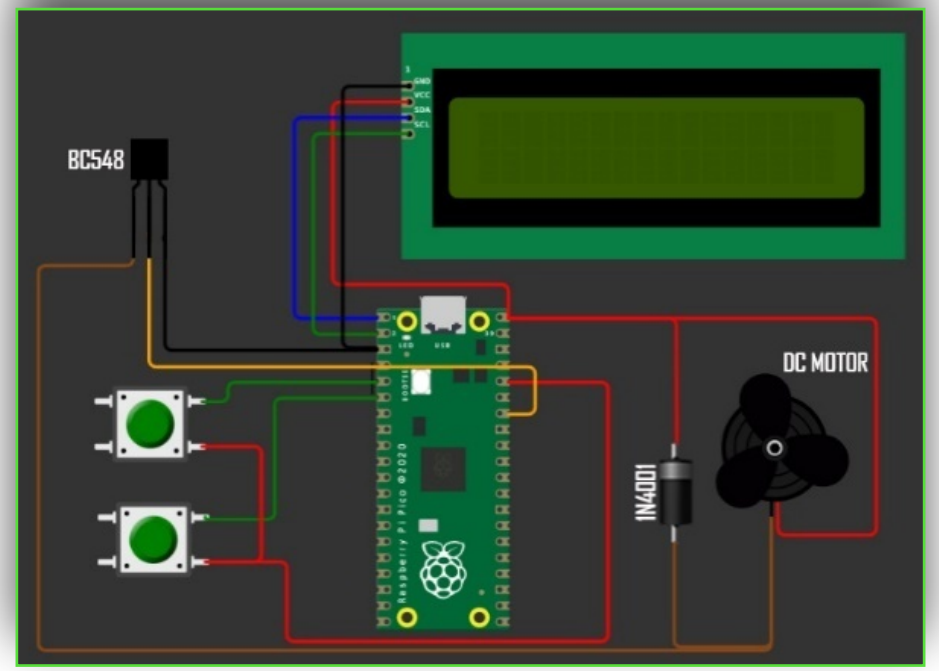


Figura 54: Esquemático do projeto.

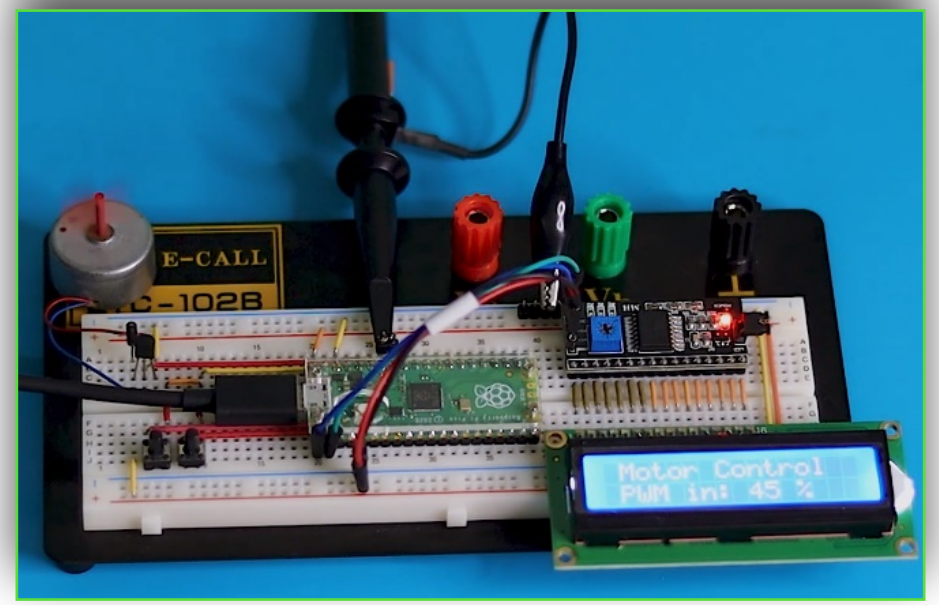


Figura 55: Teste realizamos na bancada.

Aula 17 - Conectando Pi Pico W via Wi-Fi

Wi-Fi, abreviação de "Wireless Fidelity", é uma tecnologia de comunicação sem fio que permite que dispositivos eletrônicos se conectem a uma rede local de alta velocidade, como a internet, sem a necessidade de cabos físicos. Essa tecnologia é amplamente utilizada em residências, escritórios, espaços públicos e praticamente em qualquer lugar onde a conectividade sem fio seja necessária.

O desenvolvimento da tecnologia Wi-Fi remonta ao final do século XX. Aqui temos um resumo do seu desenvolvimento:

1985-1991: A Agência de Projetos de Pesquisa Avançada em Defesa dos Estados Unidos (DARPA) começou a pesquisar e desenvolver o que viria a ser conhecido como Wi-Fi.
1991: A empresa NCR Corporation/AT&T desenvolveu o conceito de comunicação sem fio entre dispositivos de computador em locais próximos usando ondas de rádio. Isso levou ao desenvolvimento do padrão IEEE 802.11.
1997: O primeiro padrão IEEE 802.11 foi lançado, permitindo taxas de transferência de até 2 Mbps (802.11), utilizando a faixa de frequência de 2,4 GHz.
1999: O Wi-Fi Alliance foi formado para promover a interoperabilidade e a certificação dos produtos Wi-Fi. Eles lançaram o termo "Wi-Fi" como um nome de marca para produtos certificados pela Wi-Fi Alliance.
2003: O padrão IEEE 802.11g foi lançado, oferecendo velocidades de até 54 Mbps e operando na mesma faixa de frequência de 2,4 GHz.
2009: O padrão IEEE 802.11n foi lançado, oferecendo velocidades teóricas de até 600 Mbps e melhorando significativamente o alcance e a confiabilidade em comparação com os padrões anteriores.
2013: O padrão IEEE 802.11ac foi lançado, com velocidades teóricas de até vários gigabits por segundo (Gbps) e operando nas faixas de frequência de 5 GHz.
2019: O padrão IEEE 802.11ax, também conhecido como Wi-Fi 6, foi lançado, oferecendo melhorias significativas na eficiência espectral e na capacidade de lidar com uma grande quantidade de dispositivos conectados simultaneamente.

Hoje em dia, o Wi-Fi é uma tecnologia essencial que permite a conectividade sem fio em uma ampla variedade de dispositivos, desde smartphones e laptops até eletrodomésticos inteligentes e sistemas de automação residencial. Sua evolução contínua está impulsionando o desenvolvimento de novas aplicações e serviços que dependem de conectividade sem fio rápida e confiável. O Raspberry Pi Pico W, equipado com Wi-Fi integrado, abre um mundo de possibilidades para projetos de IoT e automação. Para te ajudar a

aproveitar ao máximo essa funcionalidade, preparei um guia completo sobre como funciona a conexão Wi-Fi do Pico W e como configurá-la para se conectar à sua rede.

17.1 – Modos de operação

O Pico W possui um chip Wi-Fi CYW43439 da Infineon, operando na frequência de 2,4 GHz. Isso permite que ele se conecte à maioria dos roteadores Wi-Fi domésticos e comerciais. A antena embutida no módulo facilita a conectividade Wi-Fi, sem necessidade de componentes externos. Temos os Modos de Operação Wi-Fi:

Estação: O modo mais comum, onde o Pico W se conecta a um roteador Wi-Fi existente, como em sua casa ou escritório.
Ponto de Acesso: O Pico W se torna um roteador Wi-Fi, permitindo que outros dispositivos se conectem a ele.
Cliente Ad-Hoc: O Pico W se conecta diretamente a outro dispositivo Wi-Fi, sem a necessidade de um roteador.

Algumas bibliotecas são importantes como configurar o Wi-Fi de forma correta, vamos entender melhor no nosso código.

17.2 – Código principal

```
import network
from utime import sleep
from machine import Pin
#============================================================#
ssid = "Dhanuzio 2.4Ghz"
password = "Error"
led=Pin("LED",Pin.OUT)
z=0
#================Function to connect Wi-Fi=====================#
while True:
    wlan = network.WLAN(network.STA_IF)
    wlan.active(True)
    wlan.connect(ssid,password)
    while wlan.isconnected() == False and z<10:
        print('Waiting for connection...')
        sleep(1)
        z=z+1
    print('Erro na conexão.')
    while wlan.isconnected() == True:
        print(wlan.ifconfig())
        led.toggle()
        sleep(1)
```

O código demonstra a funcionalidade básica de se conectar a uma rede Wi-Fi utilizando o Raspberry Pi Pico W e a biblioteca MicroPython. Vamos analisar o código em seções para entender melhor sua lógica e funcionamento:

- network: Importa a biblioteca network e gerencia a conexão Wi-Fi.
- utime: Importa a biblioteca utime para utilizar a função sleep(), que cria delays no programa.
- machine: Importa a biblioteca machine para controlar os pinos do Raspberry Pi Pico W.
- ssid: Armazena o nome da rede Wi-Fi à qual o Pico W se conectará ("Dhanuzio 2.4Ghz").
- password: Armazena a senha da rede Wi-Fi ("Error").
- led: Define o pino GPIO conectado ao LED onboard como saída ("LED", Pin.OUT).
- z: Variável auxiliar utilizada no loop de conexão.

Vamos usar o LED para piscar a cada 1 segundo quando houver a perfeita conexão com a nossa rede. Para o loop principal e teste de conexão, teremos as próximas linhas:

- while True: Um loop infinito que garante a execução contínua do programa.
- wlan=network.WLAN(network.STA_IF): Cria objeto wlan da biblioteca network e configura o Pico W como estação (STA) para se conectar a um ponto de acesso.
- wlan.active(True): Ativa o módulo Wi-Fi do Pico W.
- wlan.connect(ssid, password): Tenta se conectar à rede Wi-Fi especificada por ssid e password.
- while wlan.isconnected() == False and z<10: Entra em um loop enquanto a conexão não for bem-sucedida e z for menor que 10. Ou seja, temos 10 tentativas de conexão.
- print('Waiting for connection...') Exibe uma mensagem informando que o Pico W está aguardando conexão.
- sleep(1): Aguarda um segundo antes de verificar novamente a conexão.
- z=z+1: Incrementa a variável z a cada iteração.
- print('Erro na conexão.'): Se a conexão falhar após 10 tentativas, exibe uma mensagem de erro.
- while wlan.isconnected() == True: Entra em um loop enquanto a conexão Wi-Fi estiver ativa.
- print(wlan.ifconfig()): Exibe as informações de configuração da rede Wi-Fi.
- led.toggle(): Inverte o estado do LED onboard (liga/desliga).
- sleep(1): Aguarda um segundo antes de executar a próxima iteração.

De forma resumida, nosso código irá conectar um dispositivo Raspberry Pi a uma rede Wi-Fi específica e, quando a conexão é estabelecida com sucesso, alternar o estado de um LED onboard, exibindo também as configurações de rede no Shell, como mostra a Figura 56.

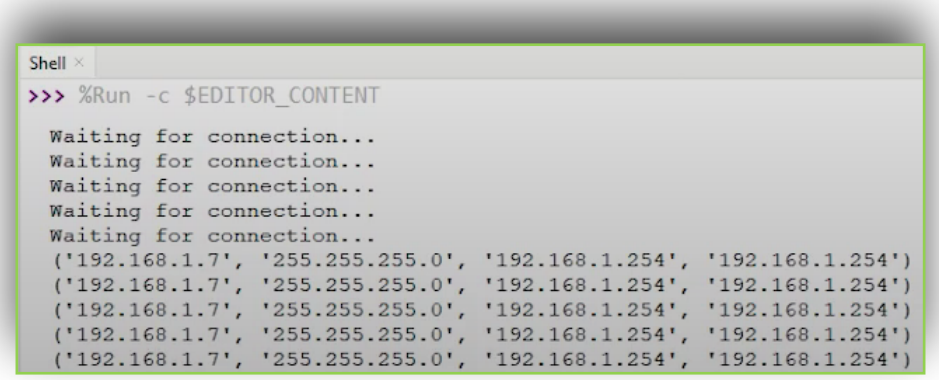


Figura 56: Mensagem no Shell quando a conexão for bem-sucedida.

17.2 - Controle de Led via web

Agora iremos fazer o controle do led onboard da nossa placa Raspberry Pi Pico W via Wi-Fi e também iremos verificar a temperatura ambiente com o sensor onboard. Vamos detalhar todo o código e em seguida verificar o funcionamento.

De forma resumida Este código configura e conecta o Raspberry Pi Pico a uma rede Wi-Fi, cria um servidor web simples que permite controlar um LED onboard via uma interface web, e lê a temperatura do sensor onboard, exibindo essas informações na página web. Vamos analisar o nosso código:

```
import machine
import socket
import network
from time import sleep
#======================Configurações iniciais====================#
wlan = network.WLAN(network.STA_IF)
wlan.active(True)
wlan.connect("Dhanuzio 2.4Ghz","Error401")
led=machine.Pin('LED',machine.Pin.OUT)
#======================Configurações iniciais====================#
while wlan.isconnected() == False:
    print('Waiting connection...')
    sleep(1)
if wlan.isconnected() == True:
    print('connected')
    ip=wlan.ifconfig()[0]
    print('IP: ', ip)
```

```
#============================Página web=======================#
def webpage(value):
    html = f"""
          <!DOCTYPE html>
          <html>
          <body>
          <form action="./on">
          <input type="submit" value="Led On" />
          </form>
          <form action="./off">
          <input type="submit" value="Led Off" />
          </form>
          </body>
          </form>
          <p>Temperature is {value} degrees Celsius</p>
          </body>
          </html>
          """
    return html
#====================Verifica a conexão com o serve===============#
def serve(connection):
    while True:
        client = connection.accept()[0]
        request = client.recv(1024)
        request = str(request)
        try:
            request = request.split()[1]
        except IndexError:
            pass
        print(request)
#=============Verifica os dados que retornam da página==============#
        if request == '/on?':              #Se o dado recebido é "on".
            led.high()                     #Liga o led.
        elif request == '/off?':           #Se o dado recebido é "off".
            led.low()                      #Desliga o led.
        Sensor = machine.ADC(4)      #Configura o sensor onboard.
        conversion_factor = 3.3 / (65535)         #Fator de conversão
      value = 27 - ((Sensor.read_u16() *conversion_factor) - 0.706)/0.001721
        html=webpage(value)                              #Html recebe value
        client.send(html)                                #Envia o para a página
        client.close()                                    #Encerra
#=======================Inicia o socket=========================#
def open_socket(ip):
```

```
    address = (ip, 80)
    connection = socket.socket()
    connection.bind(address)
    connection.listen(1)
    print(connection)
    return(connection)
#=======================Tenta a conexão=====================#
try:
    if ip is not None:
        connection=open_socket(ip)
        serve(connection)
except KeyboardInterrupt:
    machine.reset()
```

Vamos para a explicação detalhada do nosso código:

- machine: Manipulação dos pinos GPIO.
- socket: Criação e manipulação de sockets de rede.
- network: Configurações de rede Wi-Fi.
- time: Funções relacionadas a tempo, como sleep.
- wlan = network.WLAN(network.STA_IF): Cria uma instância de interface de rede no modo estação (STA_IF), que permite ao Raspberry Pi Pico se conectar a uma rede Wi-Fi.
- wlan.active(True): Ativa a interface Wi-Fi.
- wlan.connect("Dhanuzio 2.4Ghz","Error401"): Conecta-se à rede Wi-Fi especificada com o SSID "Dhanuzio 2.4Ghz" e senha "Error401".
- led=machine.Pin('LED',machine.Pin.OUT): Configura o LED onboard como um pino de saída.
- while wlan.isconnected() == False: Loop que continua executando enquanto o Raspberry Pi Pico não estiver conectado à rede Wi-Fi.
- print('Waiting connection...'): Imprime uma mensagem indicando que está aguardando a conexão.
- sleep(1): Espera 1 segundo antes de verificar novamente.
- if wlan.isconnected() == True: Verifica se o Pico está conectado à rede.
- print('connected'): Imprime uma mensagem indicando que está conectado.
- ip=wlan.ifconfig()[0]: Obtém o endereço IP do Pico e o armazena na variável ip.
- print('IP: ', ip): Imprime o endereço IP do Pico.
- def webpage(value):Define uma função chamada webpage que gera e retorna uma página HTML.
- html = f"""... """: Cria uma string HTML formatada com botões para ligar e desligar o LED e uma linha para exibir a temperatura atual (value).
- return html: Retorna a string HTML.

- def serve(connection): Define uma função chamada serve que lida com as conexões dos clientes.
- while True: Loop infinito para manter o servidor em execução.
- client = connection.accept()[0]: Aceita uma nova conexão de um cliente.
- request = client.recv(1024): Recebe a requisição HTTP do cliente.
- request = str(request): Converte a requisição para uma string.
- try...except: Tenta extrair a URL da requisição. Se não for possível (por exemplo, se a requisição for inválida), ignora o erro.
- print(request): Imprime a requisição recebida.
- if request == '/on?': Verifica se a requisição é para ligar o LED.
- led.high(): Liga o LED.
- elif request == '/off?': Verifica se a requisição é para desligar o LED.
- led.low(): Desliga o LED.
- Sensor = machine.ADC(4): Configura o sensor de temperatura onboard.
- conversion_factor = 3.3 / (65535): Define o fator de conversão para a leitura do sensor.
- value = 27 - ((Sensor.read_u16() * conversion_factor) - 0.706) / 0.001721: Converte a leitura do sensor para temperatura em Celsius.
- html = webpage(value): Gera a página HTML com a temperatura atual.
- client.send(html): Envia a página HTML para o cliente.
- client.close(): Fecha a conexão com o cliente.
- def open_socket(ip): Define uma função chamada open_socket que abre um socket de rede.
- address = (ip, 80): Define o endereço do socket (IP e porta 80).
- connection = socket.socket(): Cria um novo socket.
- connection.bind(address): Associa o socket ao endereço definido.
- connection.listen(1): Configura o socket para ouvir conexões (máximo de 1 conexão pendente).
- print(connection): Imprime informações sobre o socket.
- return(connection): Retorna o socket configurado.
- try: Inicia um bloco de código que tenta abrir o socket e iniciar o servidor.
- if ip is not None: Verifica se o endereço IP foi obtido.
- connection = open_socket(ip): Abre o socket usando o endereço IP.
- serve(connection): Inicia o servidor para lidar com as conexões.
- except KeyboardInterrupt: Captura a interrupção do teclado (Ctrl+C).
- machine.reset(): Reinicia o Raspberry Pi Pico.

Agora temos, na Figura 57 e teste realizado no nosso canal. Podemos observar nossa página com dois botões e a exibição da temperatura atual na placa, disponível em https://youtu.be/HKFDrRWn-8k?si=Y3T1ITFS3gWOeHF3.

Figura 57: Teste prático realizado no nosso canal.

Aula 18 - Servomotor e Display OLED

Os servomotores surgiram como parte do desenvolvimento de sistemas de controle automático no início do século 20. A necessidade de sistemas precisos de controle de movimento se intensificou com o avanço da tecnologia e a demanda por mecanismos mais sofisticados em aplicações militares, industriais e científicas. Durante a Segunda Guerra Mundial, os servomecanismos tiveram um papel fundamental no controle de sistemas de artilharia e torpedos. Após a guerra, as pesquisas e inovações continuaram, expandindo o uso dos servomotores para uma ampla gama de aplicações civis. Com a miniaturização dos componentes eletrônicos e o desenvolvimento de novos materiais, os servomotores tornaram-se mais compactos, eficientes e acessíveis. Na Figura 58 temos alguns modelos de servo, tanto industriais como de uso mais simples.

Figura 58: Modelos de Servomotores.

Um servomotor é um tipo de motor que é projetado para controlar a posição angular ou linear de um objeto com alta precisão. Ele é amplamente utilizado

em sistemas de controle automático onde é necessário um controle preciso de movimento. No nosso estudo, iremos adotar o modelo mais famoso: SG90.

O servomotor é parte fundamental de sistemas de automação, robótica, e máquinas CNC, entre outros. Com esses fatores em mãos, podemos determinar se esse tipo de motor é ideal para os nossos projetos. Lembrando que:

Vantagens dos Servomotores:

Alta Precisão: São capazes de controlar a posição com alta precisão.
Resposta Rápida: Respondem rapidamente aos sinais de controle, permitindo movimentos rápidos e precisos.
Versatilidade: Podem ser usados em uma ampla variedade de aplicações devido à sua capacidade de controlar tanto movimentos angulares quanto lineares.

Desvantagens dos Servomotores:

Complexidade: Podem ser mais complexos de projetar e controlar em comparação com motores simples.
Custo: Geralmente são mais caros do que motores convencionais devido aos componentes adicionais necessários para o controle preciso.

18.1 - Princípios de Operação

Como já vimos, um servomotor é um dispositivo que permite o controle preciso da posição angular (ou linear), velocidade e aceleração. Se trata de atuador mais complexo que os motores convencionais. Os principais componentes de um servomotor são vistos na Figura59.

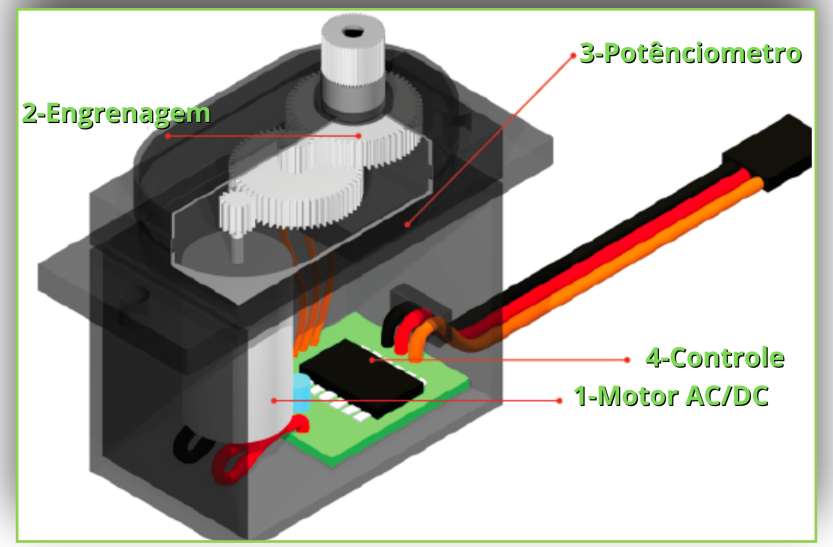


Figura 59: Partes de um servomotor SG90.

Motor DC ou AC: Fornece a força necessária para o movimento.
Sistema de Engrenagens (Redutor): Reduz a velocidade e aumenta o torque.

Potenciômetro ou Sensor de Posição: Monitora a posição atual da saída.
Controlador Eletrônico: Recebe sinais de controle e ajusta a posição do motor conforme necessário.

O funcionamento de um servomotor pode ser descrito pelos seguintes passos:

Recepção do Sinal de Controle: Um sinal de controle, geralmente um pulso de largura modulada (PWM), é enviado ao controlador do servomotor. A largura do pulso determina a posição desejada do eixo.
Comparação de Posição: O controlador compara a posição atual do eixo (fornecida pelo potenciômetro ou sensor de posição) com a posição desejada (determinada pelo sinal PWM).
Ajuste do Motor: O controlador ajusta a rotação do motor para alinhar a posição atual com a posição desejada. Se a posição atual estiver diferente da desejada, o motor gira até que ambas coincidam.
Feedback Contínuo: O potenciômetro ou sensor de posição fornece feedback contínuo sobre a posição do eixo, permitindo ajustes constantes para manter a precisão.

O servomotor SG90 opera com uma tensão de alimentação de 5V e recebe um sinal de controle no formato PWM (Pulse Width Modulation). Este sinal varia entre 0V e 5V. O circuito de controle interno do servomotor monitora esse sinal em intervalos de 20 ms ou seja, uma frequência de 50 Hz. Dependendo da largura do pulso do sinal PWM, o servomotor ajusta a posição do seu eixo, por exemplo:

0.5 ms: Corresponde à posição totalmente à esquerda do eixo, ou 0 grau.
1 ms: Corresponde à posição central do eixo, ou 90 graus.
2.5 ms: Corresponde à posição totalmente à direita do eixo, ou 180 graus.

Esses valores podem variar de acordo com o modelo, mas geralmente, os valores vistos acima, são os aproximados. Quando o servomotor SG90 recebe um sinal PWM, ele verifica a largura do pulso, por exemplo, se o servomotor recebe um pulso de 1 ms, ele verifica se o potenciômetro interno está na posição correspondente (90 graus). Se o potenciômetro já estiver na posição correta, nenhuma ação é tomada. Caso contrário, o circuito de controle aciona o motor.

O motor gira na direção necessária para alinhar a posição do potenciômetro com a posição indicada pelo sinal PWM. A direção de rotação do motor depende da posição atual do potenciômetro em relação à posição desejada. O potenciômetro fornece feedback contínuo sobre a posição do eixo, permitindo ajustes contínuos até que a posição correta seja alcançada. Na Figura 60 temos mais detalhes.

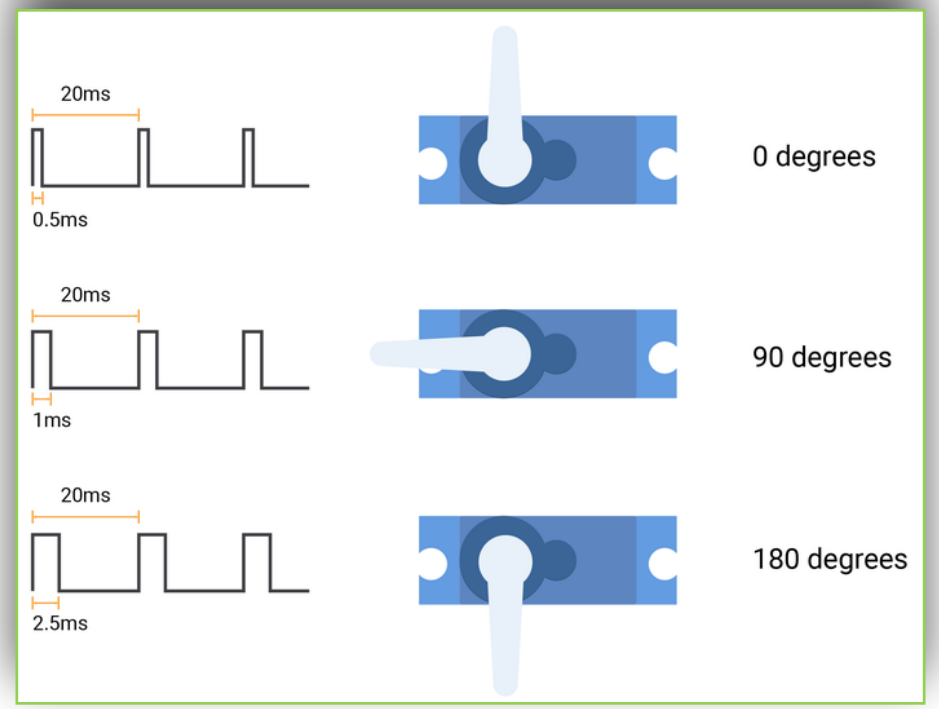


Figura 60: Pulso e respectiva posição do servomotor SG90.

18.2 - Display SSD1306

O display SSD1306 é um OLED controlado pelo driver de micro-chip SSD1306, que atua como uma ponte entre a matriz do display e o microcontrolador. Devido à luz natural dos LEDs orgânicos na matriz, os displays SSD1306 são brilhantes e possuem um amplo ângulo de visão. Esses pequenos displays são semelhantes àqueles monocromáticos usados nos antigos celulares. Esses displays são funcionais e capazes de exibir imagens complexas. A acessibilidade desse modelo o torna popular no mundo do Arduino, ESP32, Raspberry Pi Pico e da eletrônica para hobby.

As placas de breakout dos displays pertencentes à família SSD1306 podem trabalhar com a Resolução de0,91" (128x32 pixels) e 0,96" (128x64 pixels). Com cores do Monocromático, preto e branco, e duas cores assim, os OLEDs podem ter cores diferentes, por exemplo, uma parte do display é preto/branco e a outra é preto/amarelo. A comunicação pode ser realizada via Interface SPI (software ou hardware) ou Interface I^2C.

OLED significa Diodo Emissor de Luz Orgânico (Organic Light-Emitting Diode). Esse tipo de display é composto por uma camada de material orgânico posicionada entre dois eletrodos, sendo que um deles geralmente é transparente. Quando esse material orgânico é estimulado por um campo eletromagnético, ele emite luz própria, eliminando a necessidade de uma luz de fundo (backlight). A ausência de backlight torna esses displays extremamente econômicos em termos de consumo de energia. Na Figura 61 temos alguns modelos mais utilizados em projetos.

Figura 61: Alguns modelos de display OLED.

18.3 - Controlando SG90 com PWM e IHM

Vamos iniciar nosso próximo circuito, agora iremos fazer o ajuste da posição do SG90 com um potenciômetro e exibir a posição atual em um display. Na Figura 62 temos o circuito que iremos usar.

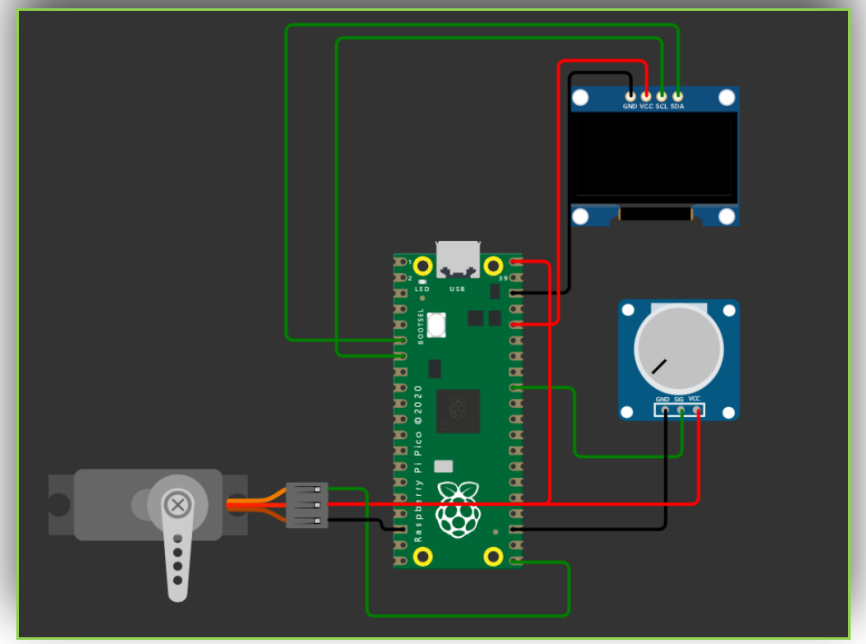


Figura 62: Circuito prático.

Nosso circuito irá funcionar com o ajuste do potenciômetro. No valor mínimo, teremos o servomotor em 0° e no valor máximo teremos o servomotor na posição 180°. Vamos analisar o código a seguir e logo teremos os testes de bancada realizados em nosso canal.

```
from machine import Pin, I2C, PWM,ADC
from ssd1306 import SSD1306_I2C
import utime
#=========================================================#
sensor = machine.ADC(27)
PwmOut=PWM(Pin(16))
PwmOut.freq(50)
WIDTH  = 128
HEIGHT = 64
i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
#=========================================================#
while True:
    PotValue=sensor.read_u16()
    angulo=int((PotValue - 0) * (7864.2 - 1638.37) / (65535 - 0) + 1638.27)
    PwmOut.duty_u16(angulo)
    reading = sensor.read_u16() /364
    oled.fill(0)
    oled.text("Engenharia",24,2)
    oled.text("entendida",30,14)
    oled.text("Servo Position:",2,30)
    oled.text(str(round(reading)),2,44)
    oled.text("Graus ",30,44)
    oled.show()
```

Vamos entender o código linha a linha:

- Pin, I2C, PWM, ADC da biblioteca machine: Estas são classes que permitem o controle de pinos de entrada/saída, comunicação I2C, controle de largura de pulso e leitura de valores analógicos.
- SSD1306_I2C da biblioteca ssd1306: Esta classe permite controlar o display OLED usando o protocolo I2C.
- utime: Esta biblioteca fornece funções relacionadas ao tempo.
- sensor = machine.ADC(27): Configura o pino 27 como entrada analógica para o sensor de posição (um potenciômetro, por exemplo).
- PwmOut = PWM(Pin(16)): Configura o pino 16 como saída PWM para controlar o servo motor.
- PwmOut.freq(50): Define a frequência do PWM para 50 Hz, que é a frequência típica para controlar servos.
- WIDTH e HEIGHT: Definem as dimensões do display OLED.

- i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000): Configura a comunicação I2C no canal 0 com o pino 5 como SCL (clock) e o pino 4 como SDA (dados) a uma frequência de 200 kHz.
- oled = SSD1306_I2C(WIDTH, HEIGHT, i2c): Inicializa o display OLED com as dimensões e a interface I2C configurada.
- PotValue = sensor.read_u16(): Lê o valor analógico do sensor (0 a 65535).
- angulo = int((PotValue - 0) * (7864.2 - 1638.37) / (65535 - 0) + 1638.27): Converte o valor lido do sensor em um valor de PWM adequado para o servo. Este cálculo ajusta o valor lido (de 0 a 65535) para o intervalo de pulsos PWM necessários (1638.27 a 7864.2).
- PwmOut.duty_u16(angulo): Define o valor do PWM para o servo motor baseado no ângulo calculado.
- reading = sensor.read_u16() / 364: Lê novamente o valor do sensor e o divide por 364 para ajustar a escala de leitura para graus.
- oled.fill(0): Limpa o display OLED.
- oled.text("Engenharia", 24, 2): Exibe o texto "Engenharia" no display na posição (24, 2).
- oled.text("entendida", 30, 14): Exibe o texto "entendida" no display na posição (30, 14).
- oled.text("Servo Position:", 2, 30): Exibe o texto "Servo Position:" no display na posição (2, 30).
- oled.text(str(round(reading)), 2, 44): Exibe o valor da leitura do sensor (convertido em graus) no display na posição (2, 44).
- oled.text("Graus ", 30, 44): Exibe o texto "Graus" no display na posição (30, 44).
- oled.show(): Atualiza o display para mostrar todas as informações definidas.

Um ponto importante é que precisamos gravar o driver para que tenhamos o correto funcionamento do display OLED, temos disponibilizado o driver logo abaixo, não precisamos explicar pois é o código padrão para esse tipo de display. Para uma explicação mais detalhada, assista nosso vídeo sobre o SSD1306 no nosso canal, disponível em: https://youtu.be/qpr2-JyHWK4?si=82Q_0LJeIXhd-KBA.

```
#=======================DriverOLED==========================#
from micropython import const
import framebuf
import time
# Register definitions
SET_CONTRAST = const(0x81)
SET_ENTIRE_ON = const(0xA4)
SET_NORM_INV = const(0xA6)
SET_DISP = const(0xAE)
```

```
SET_MEM_ADDR = const(0x20)
SET_COL_ADDR = const(0x21)
SET_PAGE_ADDR = const(0x22)
SET_DISP_START_LINE = const(0x40)
SET_SEG_REMAP = const(0xA0)
SET_MUX_RATIO = const(0xA8)
SET_COM_OUT_DIR = const(0xC0)
SET_DISP_OFFSET = const(0xD3)
SET_COM_PIN_CFG = const(0xDA)
SET_DISP_CLK_DIV = const(0xD5)
SET_PRECHARGE = const(0xD9)
SET_VCOM_DESEL = const(0xDB)
SET_CHARGE_PUMP = const(0x8D)
# Subclassing FrameBuffer provides support for graphics primitives
# http://docs.micropython.org/en/latest/pyboard/library/framebuf.html
class SSD1306(framebuf.FrameBuffer):
    def __init__(self, width, height, external_vcc):
        self.width = width
        self.height = height
        self.external_vcc = external_vcc
        self.pages = self.height // 8
        self.buffer = bytearray(self.pages * self.width)
                            super().__init__(self.buffer,  self.width,  self.height,
framebuf.MONO_VLSB)
        self.init_display()
    def init_display(self):
        for cmd in (
            SET_DISP | 0x00,  # off
            # Address setting
            SET_MEM_ADDR,
            0x00,  # horizontal
            # Resolution and layout
            SET_DISP_START_LINE | 0x00,
            SET_SEG_REMAP | 0x01,  # Column addr 127 mapped to SEG0
            SET_MUX_RATIO,
            self.height - 1,
            SET_COM_OUT_DIR | 0x08,  # Scan from COM[N] to COM0
            SET_DISP_OFFSET,
            0x00,
            SET_COM_PIN_CFG,
            0x02 if self.width > 2 * self.height else 0x12,
            # Timing and driving scheme
            SET_DISP_CLK_DIV,
```

```
            0x80,
            SET_PRECHARGE,
            0x22 if self.external_vcc else 0xF1,
            SET_VCOM_DESEL,
            0x30,  # 0.83*Vcc
            # Display
            SET_CONTRAST,
            0xFF,  # maximum
            SET_ENTIRE_ON,  # Output follows RAM contents
            SET_NORM_INV,  # Not inverted
            # Charge pump
            SET_CHARGE_PUMP,
            0x10 if self.external_vcc else 0x14,
            SET_DISP | 0x01,
        ):  # On
            self.write_cmd(cmd)
        self.fill(0)
        self.show()
    def poweroff(self):
        self.write_cmd(SET_DISP | 0x00)
    def poweron(self):
        self.write_cmd(SET_DISP | 0x01)
    def contrast(self, contrast):
        self.write_cmd(SET_CONTRAST)
        self.write_cmd(contrast)
    def invert(self, invert):
        self.write_cmd(SET_NORM_INV | (invert & 1))
    def show(self):
        x0 = 0
        x1 = self.width - 1
        if self.width == 64:
            # Displays with width of 64 pixels are shifted by 32
            x0 += 32
            x1 += 32
        self.write_cmd(SET_COL_ADDR)
        self.write_cmd(x0)
        self.write_cmd(x1)
        self.write_cmd(SET_PAGE_ADDR)
        self.write_cmd(0)
        self.write_cmd(self.pages - 1)
        self.write_data(self.buffer)
class SSD1306_I2C(SSD1306):
    def __init__(self, width, height, i2c, addr=0x3C, external_vcc=False):
```

```
        self.i2c = i2c
        self.addr = addr
        self.temp = bytearray(2)
        self.write_list = [b"\x40", None]  # Co=0, D/C#=1
        super().__init__(width, height, external_vcc)
    def write_cmd(self, cmd):
        self.temp[0] = 0x80  # Co=1, D/C#=0
        self.temp[1] = cmd
        self.i2c.writeto(self.addr, self.temp)
    def write_data(self, buf):
        self.write_list[1] = buf
        self.i2c.writevto(self.addr, self.write_list)
# Only required for SPI version (not covered in this project)
class SSD1306_SPI(SSD1306):
    def __init__(self, width, height, spi, dc, res, cs, external_vcc=False):
        self.rate = 10 * 1024 * 1024
        dc.init(dc.OUT, value=0)
        res.init(res.OUT, value=0)
        cs.init(cs.OUT, value=1)
        self.spi = spi
        self.dc = dc
        self.res = res
        self.cs = cs
        self.res(1)
        time.sleep_ms(1)
        self.res(0)
        time.sleep_ms(10)
        self.res(1)
        super().__init__(width, height, external_vcc)
    def write_cmd(self, cmd):
        self.spi.init(baudrate=self.rate, polarity=0, phase=0)
        self.cs(1)
        self.dc(0)
        self.cs(0)
        self.spi.write(bytearray([cmd]))
        self.cs(1)
    def write_data(self, buf):
        self.spi.init(baudrate=self.rate, polarity=0, phase=0)
        self.cs(1)
        self.dc(1)
        self.cs(0)
        self.spi.write(buf)
        self.cs(1)
```

Salve arquivo com o nome ssd1306.py e envie para a placa. Em seguida o código principal (main.py) também deverar ser gravado, como de costume. A placa deverá ter dois arquivos, como mostra a Figura 63.

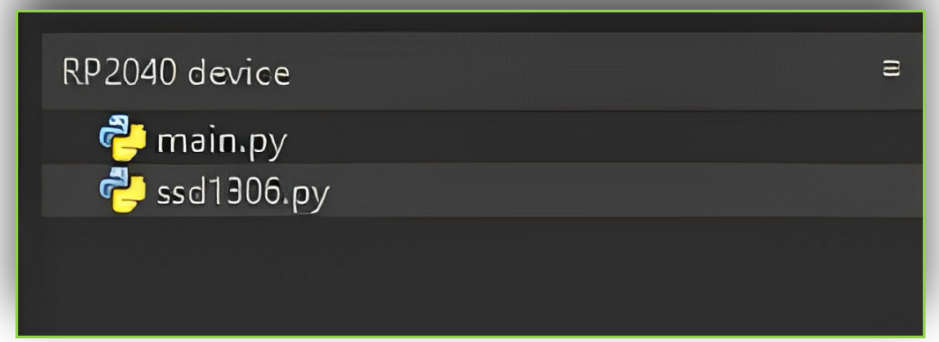


Figura 63: Arquivos que devem estar na placa.

Agora temos tudo pronto para testarmos nosso circuito. No nosso canal temos os testes completos, incluindo a verificação do sinal enviado para o servomotor com o auxílio do osciloscópio. Na Figura 64 temos o teste de bancada e Para baixar os arquivos, acesse nosso GitHub e siga nossos projetos: https://github.com/Engenharia-entendida/RaspberryPiPico/tree/Intermedi%C3%A1rio-Aula%2301.

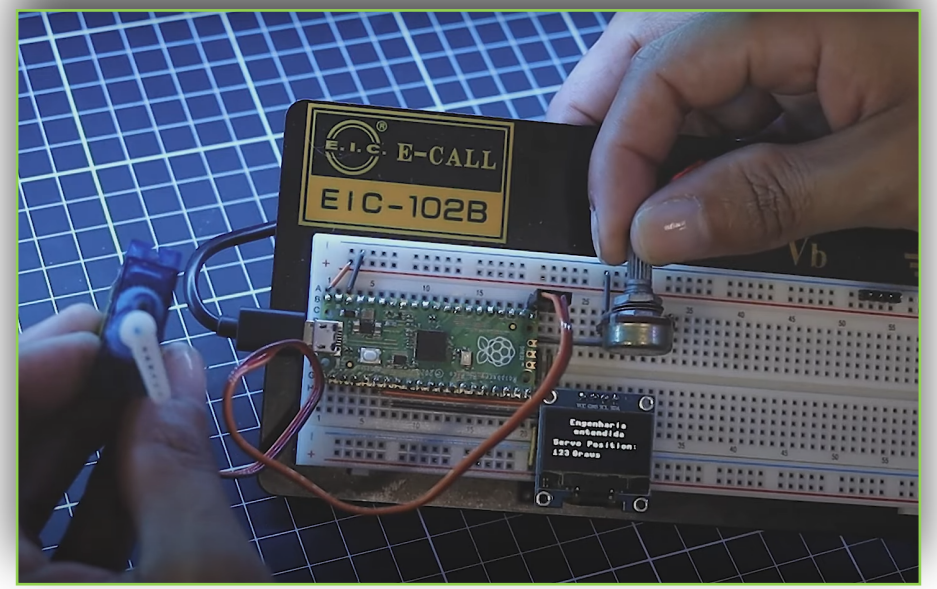


Figura 64: Teste realizado no nosso canal Engenharia entendida.

Aula 19 - DHT22 / DHT11 E Display OLED

Os sensores DHT22 e DHT11 são sensores de temperatura e umidade bastante populares e acessíveis para projetos eletrônicos, mostrado na Figura 65. Eles funcionam de maneira semelhante, mas possuem algumas diferenças em termos de precisão e faixa de medição.

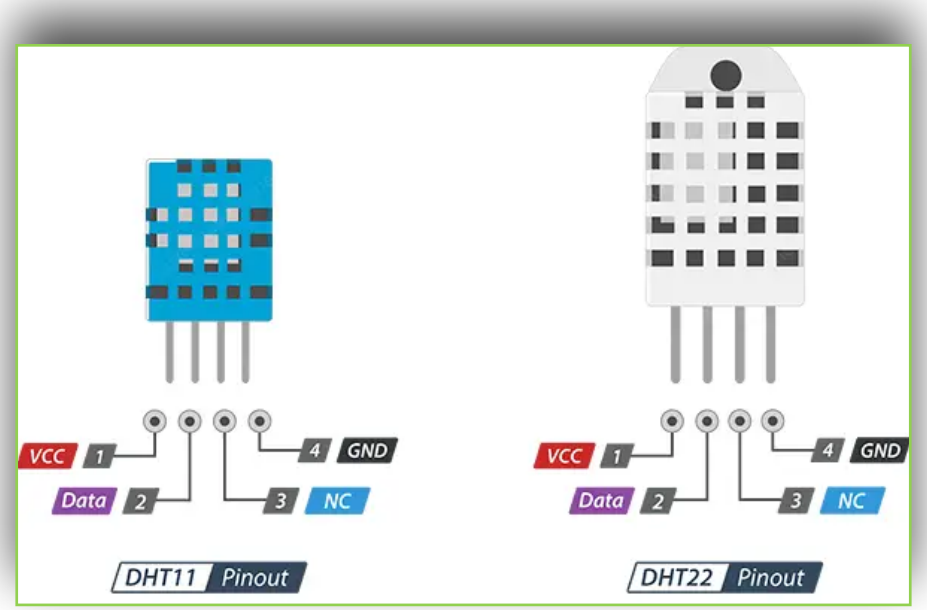


Figura 65: DHT11/22 e suas conexões.

VCC: Alimentação (3.3V a 5V, a depender do sensor e microcontrolador).
Data Out: Saída de dados (conecta ao pino digital do microcontrolador).
Não Conectado (DHT22 e 11): pino não conectado.
GND: Terra (conecta ao GND do microcontrolador).

Sensor DHT22:

Faixa de Medição:

Temperatura: -40 a 80°C
Umidade: 0 a 100% UR (Umidade Relativa)

Precisão:

Temperatura: ±0,5°C
Umidade: ±2% UR

Pinagem: Normalmente possui quatro pinos (VCC, GND, Data out, Não conectado).

Comunicação: A comunicação com o sensor é feita através de um protocolo de um único fio, onde os dados são transmitidos em forma de pulso.

Sensor DHT11:

Faixa de Medição:

Temperatura: 0 a 50°C
Umidade: 20 a 80% UR

Precisão:

Temperatura: ±2°C
Umidade: ±5% UR

Ambos os sensores são relativamente fáceis de usar com microcontroladores como Arduino, Raspberry Pi, entre outros. Utilizam um sensor capacitivo de umidade e um termistor para medir temperatura. Um microcontrolador interno faz a conversão analógico-digital e transmite os

dados através de um sinal digital único. O processo típico de leitura envolve: O microcontrolador envia um sinal de inicialização para o sensor, que responde com os dados de temperatura e umidade. O microcontrolador converte os dados recebidos em valores de temperatura e umidade utilizáveis.Os dados podem ser então processados ou exibidos conforme necessário para controle de ambiente, monitoramento, ou outras aplicações, tudo isso enviado pelo pino DATA. Esses sensores são utilizados em:

Projetos de IoT: Para monitorar condições ambientais.
Estações Meteorológicas Caseiras: Para registrar e exibir informações de temperatura e umidade.
Controle de Ambiente: Sistemas de climatização residencial ou industrial.

Em resumo, os sensores DHT22 e DHT11 são escolhas populares para medição de temperatura e umidade em projetos DIY devido à sua facilidade de uso e custo acessível. Na Figura 66 temos sua estrutura interna.

19.1 - Controle por temperatura

Nosso próximo projeto irá exibir a temperatura e umidade, provenientes do sensor DHT22, no display OLED usado no capítulo anterior, e quando tivermos a temperatura acima de um valor pré definido, iremos acionar um led, que poderia ser uma ventilação para controle de temperatura, por exemplo. Temos o circuito que iremos utilizar na Figura 66. A simulação pode ser feita no wokwi.com.

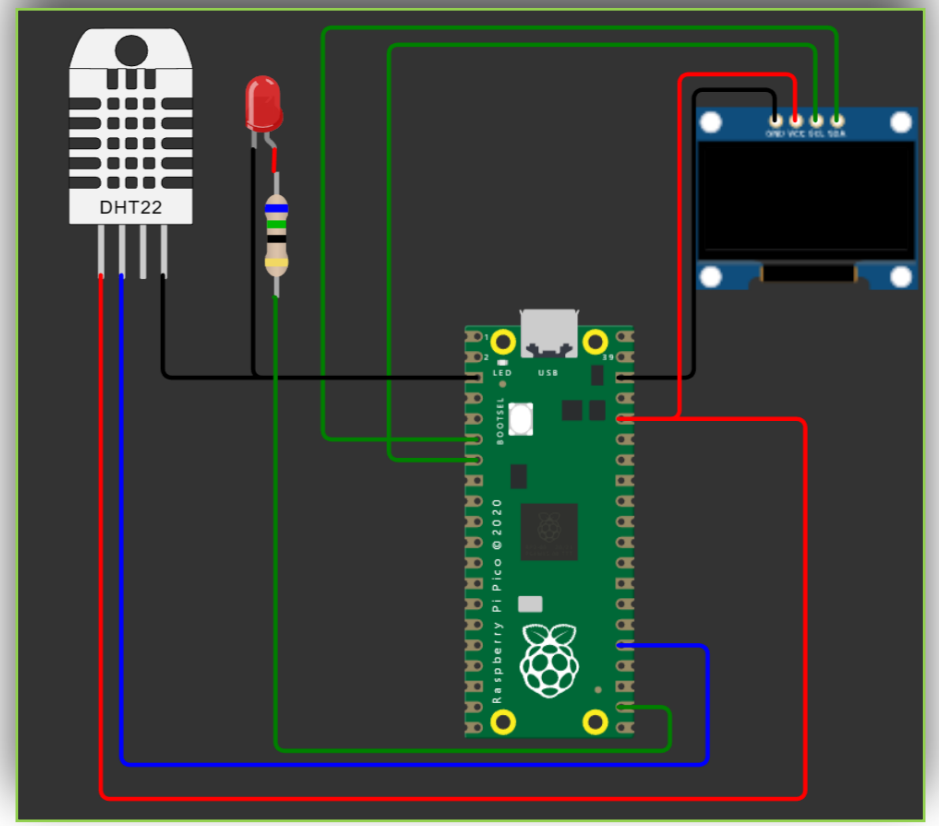


Figura 66: Circuito de controle por temperatura.

Vamos analisar o código, simples e intuitivo. Quando a temperatura estiver acima de 45°C, iremos acionar o led via pino 16, que iremos chamar de "fan", para simbolizar o acionamento de uma ventilação. Temos o código a seguir:

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
from time import sleep
import dht
WIDTH  = 128
HEIGHT = 64
sensor = dht.DHT22(Pin(19)) .
fan = Pin(16,Pin.OUT)
i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
while True:
    oled.fill(0)
    sensor.measure()
    temp = (sensor.temperature())
    hum = (sensor.humidity())
    oled.text("Engenharia",24,0)
    oled.text("entendida",30,12)
    oled.text("Temperatura:",2,22)
    oled.text(str(temp) + "C",2,32)
    oled.text("Umidade",2,44)
    oled.text(str(hum) + "%",2,53)
    oled.show()
    if temp>45:
        fan.on()
    else:
        fan.off()
    sleep(1)
```

Entendendo o código:

- from machine import Pin, I2C: Importa as classes Pin e I2C da biblioteca machine, que são usadas para controlar os pinos GPIO e a comunicação I2C, respectivamente.
- from ssd1306 import SSD1306_I2C: Importa a classe SSD1306_I2C da biblioteca ssd1306, que é usada para controlar o display OLED via I2C.
- from time import sleep: Importa a função sleep da biblioteca time, que é usada para pausar a execução do código por um período de tempo.
- import dht: Importa a biblioteca dht, que é usada para interagir com o sensor DHT22.

- WIDTH = 128 e HEIGHT = 64: Define a largura e a altura do display OLED em pixels.
- sensor = dht.DHT22(Pin(19)): Inicializa o sensor DHT22 conectado ao pino 19.
- fan = Pin(16, Pin.OUT): Configura o pino 16 como uma saída para controlar o ventilador.
- i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000): Inicializa a comunicação I2C no canal 0, com SCL no pino 5 e SDA no pino 4, e define a frequência de comunicação em 200 kHz.
- oled = SSD1306_I2C(WIDTH, HEIGHT, i2c): Inicializa o display OLED com as dimensões definidas e a interface I2C.
- while True: Inicia um loop infinito para atualizar continuamente os valores exibidos no display e controlar o ventilador.
- oled.fill(0): Limpa o display OLED preenchendo-o com preto (0).
- sensor.measure(): Lê os valores de temperatura e umidade do sensor DHT22.
- temp = sensor.temperature(): Armazena a temperatura lida em uma variável.
- hum = sensor.humidity(): Armazena a umidade lida em uma variável.
- oled.text(...): Exibe vários textos no display OLED nas posições especificadas.
- oled.show(): Atualiza o display OLED com os textos definidos.
- if temp > 45: Se a temperatura lida for maior que 45 graus Celsius, liga o ventilador; caso contrário, desliga o ventilador.
- sleep(1): Aguarda por 1 segundo antes de repetir o loop.

Aula 20 - Lendo tensões positivas e negativas

Para medir tensões positivas e negativas, precisamos usar um circuito conhecido como "Passive Averager" com microcontroladores. Ele é necessário para entender a configuração do circuito e como ele funciona para ajustar os sinais de entrada de forma que possam ser lidos corretamente pelo ADC do microcontrolador. Esse circuito é basicamente uma configuração de resistores que permite ajustar o nível de uma tensão para dentro da faixa de operação do ADC. Ele será usado para criar um ponto médio que se torna uma referência para medir tensões tanto positivas quanto negativas.

20.1 - Circuito de média passiva

Nos Amplificadores Somadores, as tensões ou sinais aplicados às múltiplas entradas de um circuito com amplificador operacional inversor podem ser "somados" para produzir uma única saída, e dependendo da configuração dos amplificadores, inversores ou não inversores, o sinal de saída será uma soma positiva ou negativa de todas as suas entradas. O amplificador somador

multiplica cada tensão de entrada pelo seu ganho ponderado determinado pela razão de Rf/Rin , que é a razão entre o resistor de feedback (Rf) e o resistor de entrada correspondente, (Rin), como visto na Figura 67.

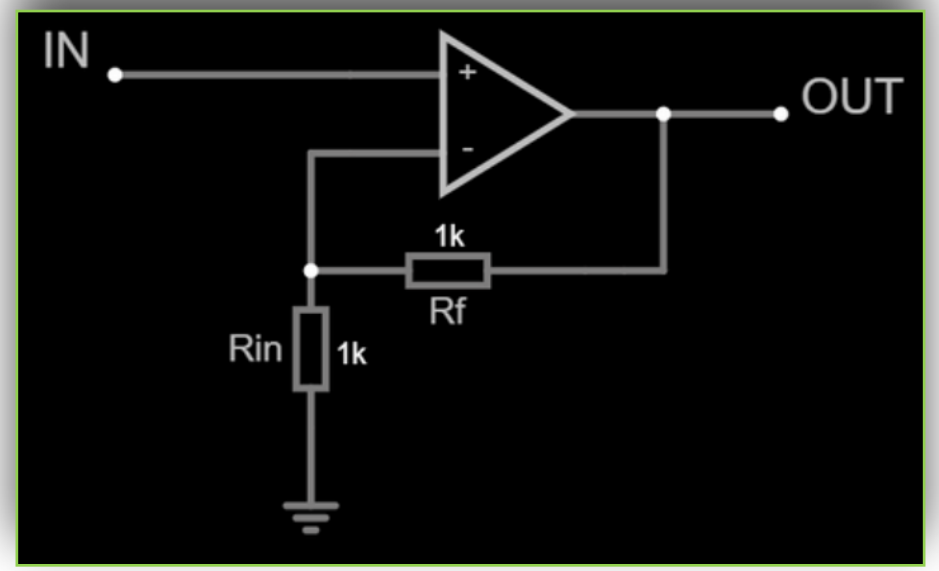


Figura 67: Amplificador somador

Mas, além de usar amplificadores operacionais como amplificadores somadores (soma) ou como amplificadores diferenciais (subtração), também podemos configurar circuitos amplificadores operacionais de múltiplas entradas para funcionar como um circuito Averager que pode produzir uma tensão de saída que corresponde ao valor médio da tensão de duas ou mais entradas. O Passive Averager é basicamente uma rede ou circuito resistivo configurado para fornecer uma tensão de saída cujo valor é igual à média matemática de todas as suas tensões de entrada. Qualquer número de entradas pode ser usado para formar um circuito médio, seja passivo ou ativo. Considere o circuito resistivo de 2 entradas Na Figura 68.

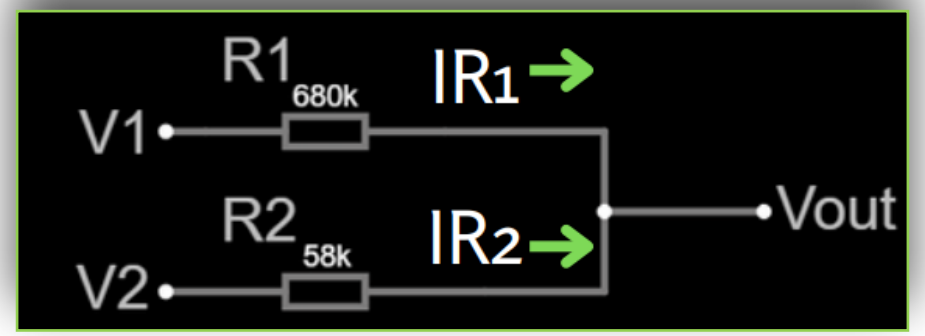


Figura 68: Circuito de média passiva com duas entradas.

Aqui, os dois resistores, R1 e R2, são conectados entre si de modo que uma extremidade de cada resistor forma uma junção ou nó comum, enquanto uma fonte de tensão é aplicada à outra extremidade de cada resistor, conforme mostrado. Isso então forma a base de um circuito médio passivo que produz uma tensão de saída igual ao valor médio das duas tensões de entrada, pois

elas são efetivamente conectadas entre si através dos resistores. Esta configuração básica de circuito também pode ser usada para circuitos somadores e subtratores. A lei das correntes de Kirchhoff (KCL) afirma que a soma algébrica de todas as correntes elétricas que entram e saem de uma junção ou nó de circuito deve ser igual a zero. Portanto a soma das correntes que passam por este circuito resistivo passivo será igual a: Corrente Total = IR1 + IR2. Portanto:

$$I_{R1} = \frac{V_1}{R_1} \quad \text{e} \quad I_{R2} = \frac{V_2}{R_2}$$

$$I_T = I_{R1} + I_{R2} = \frac{V_1}{R_1} + \frac{V_2}{R_2}$$

$$I_T = \frac{V_{OUT}}{R_T} = V_{OUT} \times \frac{1}{R_T}, \quad \text{onde} \quad \frac{1}{R_T} = \frac{1}{R_1} + \frac{1}{R_2}$$

$$\therefore V_{OUT} = \frac{I_T}{\frac{1}{R_T}} = \frac{\frac{V_1}{R_1} + \frac{V_2}{R_2}}{\frac{1}{R_1} + \frac{1}{R_2}}$$

Isto basicamente significa que Vout é igual à soma das correntes de entrada dividida pelo valor recíproco dos resistores individuais, já que os resistores estão efetivamente conectados entre si em paralelo através das fontes de tensão, e esta ideia faz parte do Teorema de Millman. Isso é V = I/G, onde "G" é condutância. Então podemos expandir esta equação básica da média passiva de 2 entradas para circuitos resistivos com múltiplas entradas de 3, 4 ou mais resistores e tensões.

$$V_{OUT} = \frac{\frac{V_1}{R_1} + \frac{V_2}{R_2} + \frac{V_3}{R_3} + \frac{V_4}{R_4} + ... + \frac{V_n}{R_n} ..etc.}{\frac{1}{R_1} + \frac{1}{R_2} + \frac{1}{R_3} + \frac{1}{R_4} + ... + \frac{1}{R_n} ..etc.}$$

Por exemplo, em circuito médio passivo de 2 entradas é construído usando um resistor de 2kΩ e um resistor de 4kΩ conectados entre si. Se uma fonte de tensão de 12 volts CC estiver conectada a uma extremidade do resistor de 2kΩ e uma segunda fonte de tensão de 6 volts CC estiver conectada a uma extremidade do resistor de 4kΩ. Calcule a tensão de saída na junção comum. Assumindo: R1 = 2kΩ, R2 = 4kΩ, V1 = 12V e V2 = 6V, logo:

$$V_{OUT} = \frac{\frac{V_1}{R_1} + \frac{V_2}{R_2}}{\frac{1}{R_1} + \frac{1}{R_2}} = \frac{\frac{12}{2000} + \frac{6}{4000}}{\frac{1}{2000} + \frac{1}{4000}}$$

$$V_{OUT} = \frac{\frac{3}{400}}{\frac{3}{4000}} = \frac{0.0075}{0.00075} = 10 \text{ Volts}$$

Temos o circuito na Figura 69:

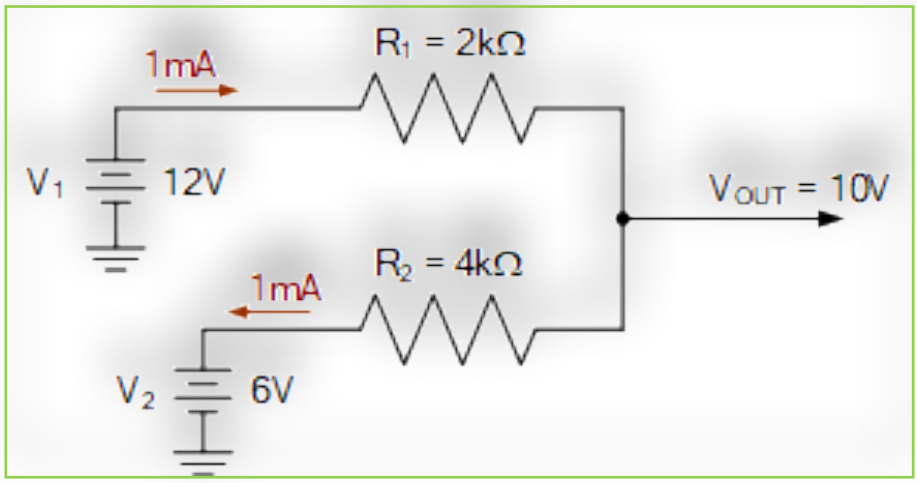


Figura 69: Circuito de exemplo.

Portanto, a tensão de junção do nó comum foi calculada como 10 volts. Mas você pode estar sentado aí pensando que: (12 + 6)/2 = 9 volts. A saída de tensão média deve ser de 9 volts, e você está correto. No entanto, os dois resistores usados neste exemplo são de valores diferentes, 2kΩ e 4kΩ, portanto influenciarão as correntes que fluem através da rede resistiva produzindo o que é conhecido como Circuito Médio Ponderado . Ou seja, cada entrada é multiplicada por seu fator de ponderação antes de ser calculada a média. Se tivéssemos resistores de mesmo valor, ai sim, teríamos a média de 9V.

20.2 - Amplificador operacional

Vamos imaginar o amplificador operacional (OPAMP) como um chef de cozinha muito talentoso que pode preparar pratos incríveis com ingredientes muito simples. Vamos explorar como ele funciona e suas configurações de forma lúdica:

"O Chef Operacional"

Imagine que o chef (OPAMP) trabalha em um restaurante com duas entradas para ingredientes (os terminais de entrada) e uma saída para os

pratos finalizados (o terminal de saída). As duas entradas são chamadas de entrada inversora (–) e entrada não inversora (+). O chef tem uma regra muito clara para cozinhar: ele compara os ingredientes que recebe nas duas entradas. Se a quantidade de ingredientes na entrada não inversora (+) é maior do que na entrada inversora (–), ele prepara um prato muito grande (saída positiva). Se a quantidade na entrada inversora (–) é maior, ele prepara um prato muito pequeno ou até um prato invertido (saída negativa). Se as duas entradas têm a mesma quantidade, o prato finalizado tem um tamanho zero (0V).Temos algumas configurações:

Amplificador Inversor: Aqui, o chef pega ingredientes de uma só entrada e faz ajustes com base na entrada inversora (–). A quantidade de ingredientes (tensão de entrada) na entrada não inversora (+) é mantida em zero (ou aterrada). Ele usa a entrada inversora (–) e ajusta os pratos de acordo com uma receita específica (determinada por resistores de feedback):

- Ingrediente na entrada inversora (–).
- Recebe ajuda de um ajudante (resistor de feedback) que pega uma porção do prato finalizado e a compara com a entrada.
- O prato finalizado (saída) é invertido (ou seja, se a entrada é positiva, a saída é negativa e vice-versa) e ampliado (multiplicado por um fator determinado pela razão dos resistores).

Amplificador Não Inversor: Nesta configuração, o chef usa a entrada não inversora (+):

- Ingredientes entram pela entrada não inversora (+).
- A entrada inversora (–) é conectada à saída através de um resistor de feedback, garantindo que a saída seja uma versão amplificada da entrada não inversora (+).
- O prato finalizado (saída) é maior e da mesma "polaridade" que a entrada (não invertido).

Seguidor de Tensão (Buffer): Às vezes, o chef precisa preparar um prato que deve ter exatamente a mesma quantidade de ingredientes da entrada:

- Ingredientes entram pela entrada não inversora (+).
- O chef ajusta a saída para que seja exatamente igual à entrada.
- Não há amplificação (o prato finalizado tem o mesmo tamanho dos ingredientes), mas o chef garante que a qualidade e a força do prato são mantidas, mesmo que a entrada seja fraca.

Os amplificadores operacionais têm algumas características especiais que fazem deles chefs excepcionais.

- Ganho Alto: Eles podem comparar diferenças minúsculas nas entradas e produzir saídas muito grandes.

- Impedância de Entrada Alta: Eles não "roubam" muitos ingredientes das entradas, permitindo que outros equipamentos compartilhem os ingredientes.
- Impedância de Saída Baixa: Eles conseguem fornecer pratos fortes e robustos, mesmo para clientes exigentes (cargas pesadas).

"Alimentação para o chef operar"

Os amplificadores operacionais são componentes essenciais em muitos circuitos eletrônicos, conhecidos por sua versatilidade e capacidade de amplificar sinais. No entanto, o desempenho e o comportamento desses dispositivos estão diretamente ligados à sua alimentação. A alimentação de um amp op refere-se às tensões fornecidas aos seus terminais de alimentação, que normalmente consistem em uma tensão positiva (+Vcc) e uma tensão negativa (-Vcc) ou uma única tensão positiva em alguns casos.

Alimentação Simétrica: Na maioria das vezes, o chef opera com uma alimentação simétrica, onde ele tem acesso tanto à alimentação positiva (+Vcc) quanto à negativa (-Vcc). Isso permite que ele prepare pratos (saídas) que podem ser tanto positivos quanto negativos.

- Vantagem: O chef pode equilibrar melhor os ingredientes e preparar uma gama maior de pratos com precisão.
- Aplicação: Usada em aplicações onde sinais de entrada podem variar tanto positiva quanto negativamente, como em amplificadores de áudio.

Alimentação Assimétrica: Às vezes, o chef só tem acesso a uma alimentação positiva e a um ponto de referência (GND). Isso é como ter um suprimento limitado de ingredientes positivos.

- Limitação: O chef pode preparar pratos apenas dentro do intervalo de 0V até +Vcc. Ele não pode preparar pratos negativos.
- Aplicação: Usada em circuitos digitais ou onde apenas sinais positivos são necessários.

Vamos considerar um amplificador operacional alimentado com +15V e -15V. Quando o chef (Operacional) compara os ingredientes nas entradas (tensões de entrada), ele pode ajustar a saída entre -15V e +15V. Se a diferença de ingredientes é muito grande, ele atinge um limite (saturação) e não consegue mais aumentar a saída. Se a alimentação for reduzida para +5V e -5V, o chef ainda segue as mesmas regras, mas agora ele só pode preparar pratos dentro desse novo limite. Portanto, a saída máxima será de +5V e a mínima de -5V. Se tivermos a alimentação entre +5V e GND, o saída se limitará a essa faixa também. A fórmula para o ganho de um amplificador operacional depende da configuração do circuito. As fórmulas e o circuito são vistas a seguir, mais detalhadamente.

Amplificador Inversor

No amplificador inversor, a entrada de sinal é aplicada ao terminal inversor (-) do amp op, enquanto o terminal não inversor (+) é conectado ao terra (0V). O ganho de tensão (Av) é determinado pelos resistores de entrada (Rf) e de feedback (Rin). Fórmula para o ganho de um amplificador inversor é:

$$\boldsymbol{Av = -\frac{Rf}{Rin}}$$

- Av é o ganho de tensão.
- Rf é o resistor de feedback, conectado entre a saída e a entrada inversora.
- Rin é o resistor de entrada, conectado entre a fonte de sinal e a entrada inversora.

O sinal de saída é invertido em relação ao sinal de entrada, daí o sinal negativo na fórmula. Mas para que tenhamos esse sinal diferente de 0V(GND), precisamos de uma fonte simétrica para o operacional, como vimos acima. Temos a configuração mostrada na Figura 70.

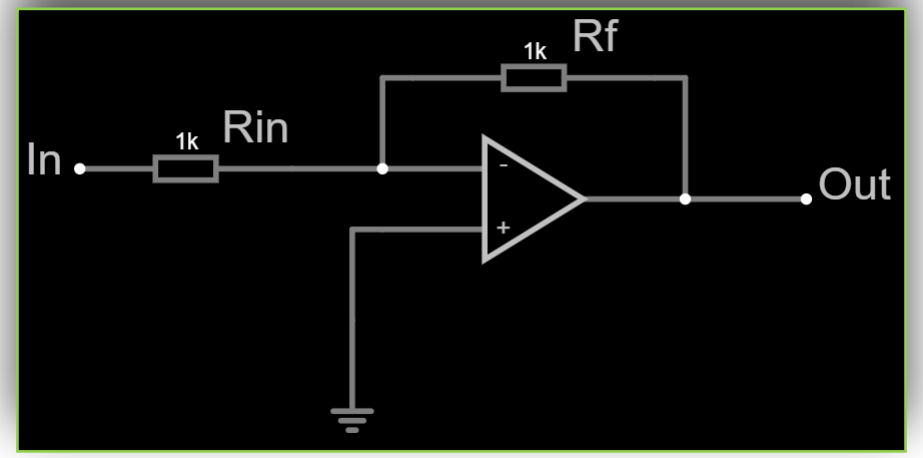


Figura 70: Circuito para amplificador inversor.

Amplificador Não-Inversor

No amplificador não-inversor, a entrada de sinal é aplicada ao terminal não inversor (+) do amp op, e o terminal inversor (-) está conectado ao terra através de um divisor de tensão feito por Rf e Rg. A fórmula para o ganho de um amplificador não inversor é:

$$\boldsymbol{Av = 1 + \frac{Rf}{Rg}}$$

- Av é o ganho de tensão.
- Rf é o resistor de feedback conectado entre a saída e entrada inversora.

- Rg é o resistor de ganho, conectado entre a entrada inversora e o terra.

O sinal de saída não é invertido em relação ao sinal de entrada. Temos a configuração mostrada na Figura 71.

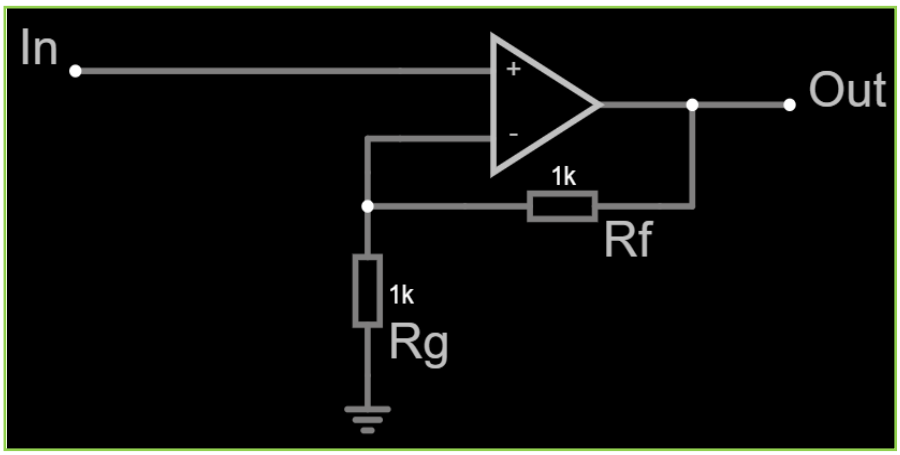


Figura 71: Figura 70: Circuito para amplificador não- inversor.

Amplificador Buffer

Um buffer é um circuito que fornece uma saída que segue exatamente a tensão de entrada. Ele tem a vantagem de fornecer alta impedância de entrada e baixa impedância de saída. O buffer ou seguidor de tensão é extremamente útil quando você precisa isolar um sinal de uma fonte de alta impedância e fornecer uma saída de baixa impedância. A fórmula para o buffer é:

$$Av = 1$$

Temos a configuração mostrada na Figura 72.

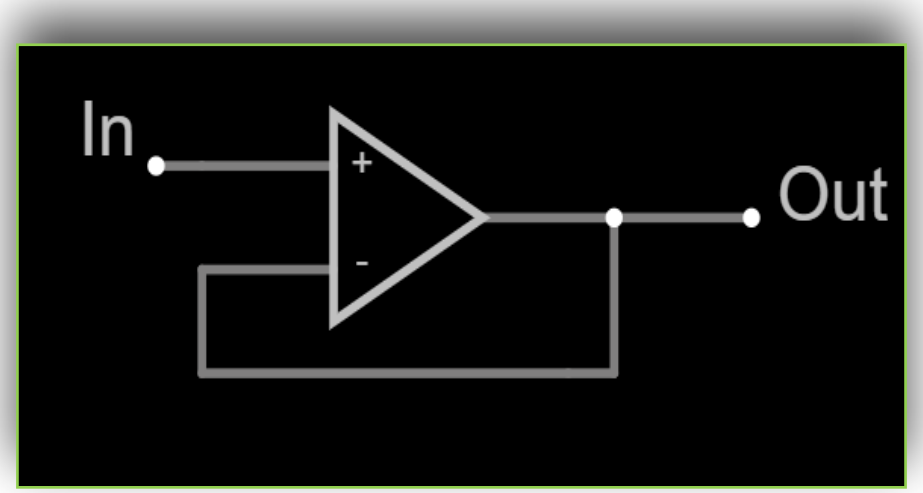


Figura 72: Circuito para buffer.

O operacional LM358, que usaremos no nosso projeto, é mostrado na Figura 70.

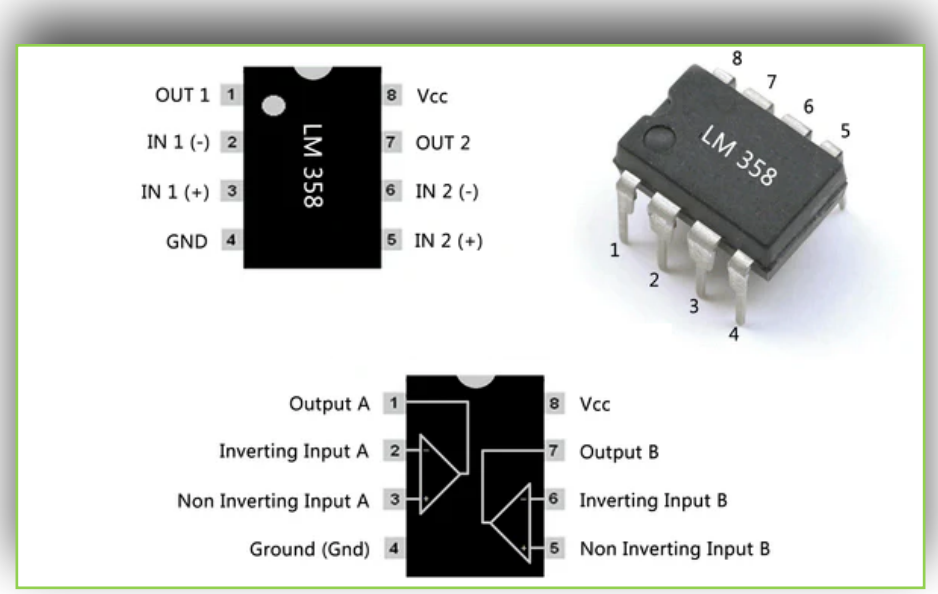


Figura 73: LM358 e suas conexões.

20.3 - Circuito do voltímetro

Sabemos que o ADC da nossa placa, lê valores de 0 a 3.3 V, que tem a faixa de valores digitais de 0 a 65536. Para usamos nossa placa para valores de tensão positivos e negativos, iremos deslocar a referência, que é 0 V, para a metade de 3.3 V, ou seja, 1.65 V. Assim teremos a relação mostrada na Figura 74.

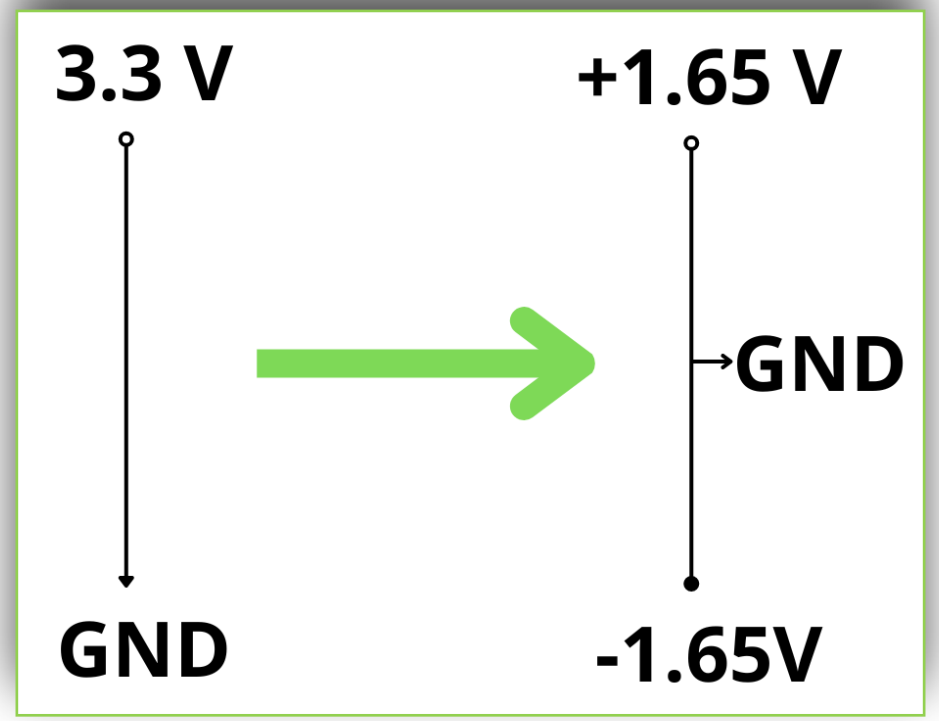


Figura 74: Deslocando a referência do ADC.

Para gerarmos esse valor de referência, iremos usar um divisor resistivo, a partir de um operacional como buffer e um trimpot. Um trimpot, ou trimmer potentiometer (potenciômetro de ajuste), é um tipo de potenciômetro ajustável de pequeno porte que é usado para calibrar, ajustar e sintonizar

circuitos elétricos e eletrônicos. Ao contrário dos potenciômetros comuns que possuem um eixo para ajuste frequente, os trimpots são projetados para ajustes esporádicos e são frequentemente ajustados uma vez durante a montagem do equipamento e depois deixados na mesma posição. Mostrado na Figura 75.

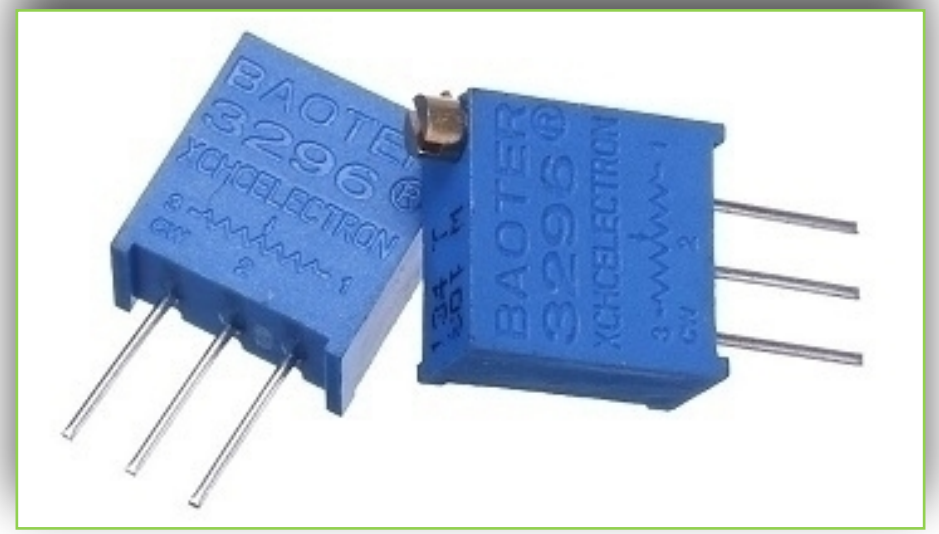


Figura 75: Modelo de trimpot do nosso projeto.

Iremos usar o LM358, pois temos dois operacionais no mesmo encapulamento. Vamos usar dois buffers para nosso projeto, pois instrumento de medição deve ter pouca ou quase nada de influência no circuito de teste. Assim temos o circuito final mostrado na Figura 76.

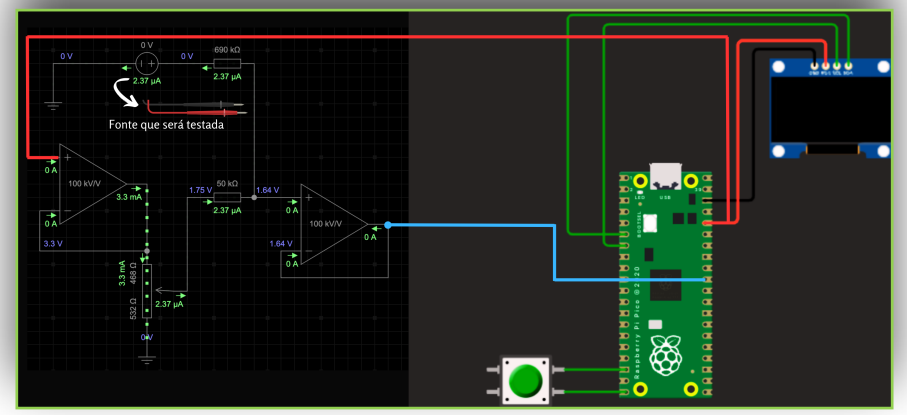


Figura 77: Circuito do voltímetro completo.

Como visto no circuito de média móvel, podemos utilizar nosso circuito com a referência de 1.65 V com um resistor de 50K e nossa fonte de teste, estará com o resistor de 690K. Se aplicarmos esses valores na equação da média móvel, iremos ler valores de aproximadamente -20 V até +20 V, sem ultrapassar o limite de 3.3 V do ADC da nossa placa. Agora, tudo que temos que fazer é ajustar o código para que quando tivermos 1.65 V, tenhamos 0 V. Entre 1.65 e 3.3 V temos valores entre 0 V e +20 V. E entre 1.65 e 0 V temos valores entre 0 V e -20 V. Nosso projeto terá um botão, para

alternar entre voltímetro e amperímetro, para nosso próximo projeto, alternar entre as duas funções. Para uma aula mais detalhada e explicativa, acesso o vídeo sobre amperímetro no nosso canal, para uma melhor análise e entendimento. Disponível em: https://youtu.be/H1vo3ReMOMg?si=c-P1Z-IdqG25YBx4.

20.4 - Código do voltímetro

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
from time import sleep
from machine import ADC
sleep(1.5)
WIDTH = 128
HEIGHT = 64
Button = Pin(15, Pin.IN, Pin.PULL_UP)
i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
Run = True
SensorVolts2=ADC(27)
SensorVolts1=ADC(26)
def voltimeter():
    oled.fill(0)
    oled.text("Voltimetro", 24, 0)
    oled.text("Volts: {:.3f}V".format(Volts), 2, 18)
    oled.show()
    sleep(0.01)
def amperimeter():
    oled.fill(0)
    oled.text("Amperimetro", 24, 0)
    oled.show()
    sleep(0.01)
while True:
    Total=0
    for i in range(150):
        Volts1 = SensorVolts1.read_u16()
        Total += Volts1
        sleep(0.012)
    Total=Total/150
    Volts=Total*(3.28/65535)
    Volts=(Volts - 1.64) * 14.16
    ButtonState = Button.value()
    if ButtonState == 0:
        Run = not Run
        sleep(0.1)
```

```
if Run:
    voltimeter()
else:
    amperimeter()
```

- machine: Importa classes para trabalhar com pinos (Pin), I2C (I2C), e ADC (ADC).
- ssd1306: Importa a classe SSD1306_I2C para controlar o display OLED.
- time: Importa a função sleep para introduzir atrasos no código.
- sleep(1.5): Aguarda 1.5 segundos.
- WIDTH e HEIGHT: Define a largura e a altura do display OLED.
- Button: Configura o pino 15 como entrada com um resistor de pull-up.
- i2c: Inicializa a comunicação I2C usando os pinos 5 (SCL) e 4 (SDA) com frequência de 200kHz.
- oled: Inicializa o display OLED com as dimensões e a interface I2C configuradas.
- Run: Variável de controle do estado do display (voltímetro ou amperímetro).
- SensorVolts2 e SensorVolts1: Configura os pinos 27 e 26 como entradas analógicas (ADC).
- oled.fill(0): Limpa o display.
- oled.text("Voltimetro", 24, 0): Escreve "Voltímetro" no display na posição (24, 0).
- oled.text("Volts: {:.3f}V".format(Volts), 2, 18): Mostra a leitura de tensão formatada com três casas decimais na posição (2, 18).
- oled.show(): Atualiza o display para mostrar o conteúdo.
- sleep(0.01): Introduz um pequeno atraso.
- Total = 0: Inicializa a variável Total.
- for i in range(150): Realiza 150 leituras do ADC.
- Volts1 = SensorVolts1.read_u16(): Lê o valor de 16 bits do ADC.
- Total += Volts1: Acumula os valores lidos.
- sleep(0.012): Introduz um atraso de 12ms entre as leituras.
- Total = Total / 150: Calcula a média das leituras do ADC.
- Volts = Total * (3.28 / 65535): Converte o valor médio do ADC para tensão (considerando a referência de 3.28V).
- Volts = (Volts - 1.64) * 14.16: Ajusta a tensão subtraindo 1.64V e multiplicando por 14.16 para deslocar e escalar a leitura.
- ButtonState = Button.value(): Lê o estado do botão.
- if ButtonState == 0: Se o botão estiver pressionado (valor 0):
- Run = not Run: Alterna o estado de Run.
- sleep(0.1): Introduz um atraso para debounce.
- if Run: Se Run for True, chama a função voltimeter().
- else: Caso contrário, chama a função amperimeter().

Aula 21 - Amperímetro com proteção

Um amperímetro é um instrumento de medição usado para medir a corrente elétrica em um circuito. A unidade de medida da corrente elétrica é o ampère (A), e o amperímetro é projetado para medir essa quantidade de forma precisa. O amperímetro é conectado em série com o circuito cuja corrente se deseja medir. Isso é necessário porque a corrente que flui pelo circuito deve ser e mesma que passa através do amperímetro para ser medida. A nossa leitura será feita através do resistor Shunt. Um resistor shunt é um resistor de baixa resistência e alta precisão utilizado em conjunto com um amperímetro para medir correntes elétricas. Quando a corrente elétrica passa pelo shunt, gera uma pequena queda de tensão proporcional à corrente. Essa queda de tensão é então medida pelo amperímetro e convertida em uma leitura de corrente. O resistor shunt é conectado em série com o circuito onde a corrente precisa ser medida. A maior parte da corrente do circuito passa através do shunt. De acordo com a Lei de Ohm, a corrente que passa pelo shunt gera uma queda de tensão V que é proporcional à corrente I e à resistência Rs do shunt. O amperímetro (ou um voltímetro configurado para medir a pequena tensão gerada) é conectado em paralelo com o shunt. Medindo essa pequena tensão e sabendo o valor exato da resistência do shunt, a corrente total pode ser calculada usando a fórmula:

$$I = \frac{V}{Rs}$$

Suponha que tenhamos um resistor shunt com uma resistência de 0,01 ohms (10 mΩ) e a corrente que você deseja medir é de 10 amperes (A). A queda de tensão através do shunt seria:

$$V = I*Rs \qquad 10A \times 0{,}01\Omega = 0{,}1V = (100\ mV).$$

O amperímetro ou voltímetro mede essa tensão de 100 mV. Conhecendo a resistência do shunt, a corrente pode ser calculada:

$$I = \frac{V}{Rs} \qquad I = \frac{0{,}01V}{0{,}01\Omega} = 10A$$

Assim, o uso de um resistor shunt com um amperímetro é uma técnica eficaz e segura para medir correntes elétricas altas com precisão. Eles geralmente são resistores de baixo valor e alta potência, como mostra a Figura 78.

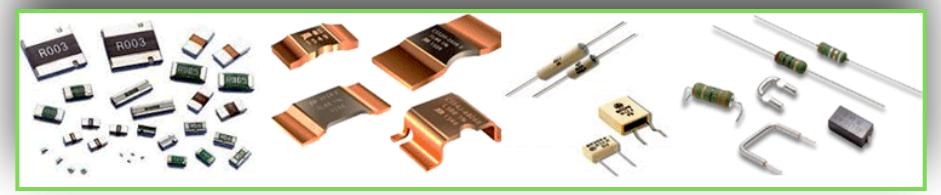


Figura 78: Alguns modelos de shut.

Agora, que devemos fazer é usar um circuito semelhante do voltímetro para temos uma tensão de referência de 1.65 V e com operacional. Lembrando que, para leitura de corrente, devemos colocar nossa pronta de prova em série com o circuito de testes, ou seja, devemos abrir o circuito. Nosso resistor shunt será de 200mΩ. Nosso operacional na configuração não-inversora, irá ter resistores de 5.1kΩ e 3.3kΩ, o que nos dá um ganho de 1.54, tendo em vista que o resistor shunt, terá baixas variações, vamos então multiplicar por 1.54 para uma melhor leitura, na saída do operacional. O circuito completo é visto na Figura 79.

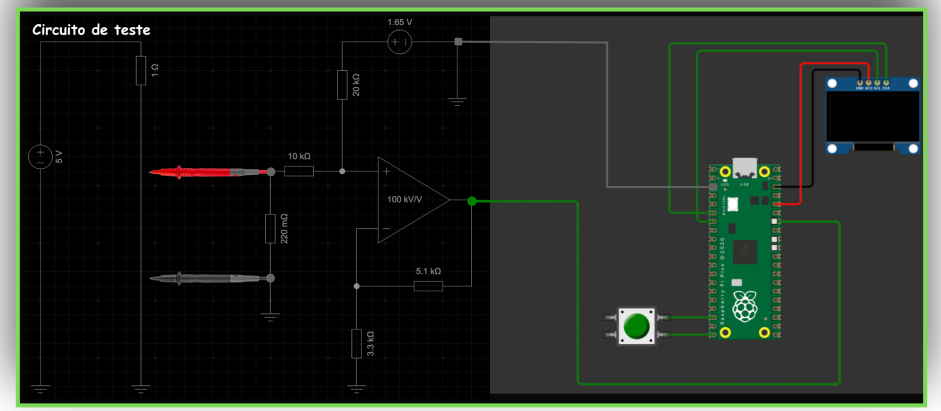


Figura 79: Circuito completo do amperímetro.

21.1 - Código do amperímetro

```
from machine import Pin, I2C
from ssd1306 import SSD1306_I2C
from time import sleep
from machine import ADC
import _thread
sleep(1.5)
WIDTH = 128
HEIGHT = 64
Button = Pin(15, Pin.IN, Pin.PULL_UP)
i2c = I2C(0, scl=Pin(5), sda=Pin(4), freq=200000)
oled = SSD1306_I2C(WIDTH, HEIGHT, i2c)
global Run
Run = True
SensorVolts1=ADC(26)
SensorCurrent=ADC(28)
```

```
Total = 0
Volts = 0
Amp = 0
#=========================================================#
def voltimeter():
    oled.fill(0)
    oled.text("Voltimetro", 24, 0)
    oled.text("Volts: {:.3f}V".format(Volts), 2, 18)
    oled.show()
    sleep(0.1)
def amperimeter():
    oled.fill(0)
    oled.text("Amperimetro", 24, 0)
    oled.text("Current: {:.3f}A".format(Amp), 2, 18)
    oled.show()
    sleep(0.1)
def Alert():
    oled.fill(0)
    oled.text("Amperimetro", 24, 0)
    oled.text("Pontas Invertidas!", 2, 20)
    sleep(1)
    oled.show()
#=======================Thread============================#
def Pressed():
    global Run
    while True:
        if Button.value()==0:
            Run = not Run
        sleep(0.003)
_thread.start_new_thread(Pressed,())
#=========================Loop============================#
while True:
    while (Run==True):
        Total=0
        for i in range(50):
            Volts1 = SensorVolts1.read_u16()
            Total += Volts1
            sleep(0.01)
        Total=Total/50
        Volts=Total*(3.3/65535)
        Volts=(Volts - 1.623) * 14.9981
        voltimeter()
    while (Run==False):
```

```
Total=0
for i in range(50):
    Amp1 = SensorCurrent.read_u16()
    Total += Amp1
    sleep(0.01)
Total=Total/50
Amp=Total*(3.3/65535)
Amp=(Amp - 1.411) * 0.985
if Amp < 0:
    Alert()
else:
    amperimeter()
```

Nosso código implementa um voltímetro e um amperímetro usando um Raspberry Pi Pico, um display OLED e sensores ADC da placa. Ele configura pinos GPIO e I2C, inicializa os ADCs para leitura de tensão e corrente, e inicializa o display OLED. O código contém funções para exibir a tensão, exibir a corrente e exibir um alerta. A função `Pressed()` verifica continuamente o estado do botão e alterna entre os modos de voltímetro e amperímetro, sendo executada em uma thread separada. No loop principal, quando `Run` é `True`, o código realiza a leitura média de 50 valores do sensor de tensão, converte e ajusta o valor para volts, e exibe no display.

Quando `Run` é `False`, realiza a leitura média de 50 valores do sensor de corrente, converte e ajusta o valor para amperes, e exibe no display, mostrando um alerta se a leitura for negativa para o amperímetro, indicando que o usuário está com as pontas invertidas. Vamos analisar o código a seguir.

- from machine import Pin, I2C: Importa as classes Pin e I2C para manipulação de GPIO e comunicação I2C.
- from ssd1306 import SSD1306_I2C: Importa a classe SSD1306_I2C para controlar o display OLED.
- from time import sleep: Importa a função sleep para introduzir atrasos no código.
- from machine import ADC: Importa a classe ADC para leitura de valores analógicos.
- import _thread: Importa o módulo _thread para manipulação de threads.
- sleep(1.5): Aguarda 1.5 segundos.
- WIDTH e HEIGHT: Define a largura (128) e a altura (64) do display OLED.
- Button: Configura o pino 15 como entrada com um resistor de pull-up.
- i2c: Inicializa a comunicação I2C usando os pinos 5 (SCL) e 4 (SDA) com frequência de 200 kHz.

- oled: Inicializa o display OLED com as dimensões definidas e a interface I2C configurada.
- global Run: Declara a variável Run como global e a inicializa como True.
- SensorVolts1 e SensorCurrent: Inicializa os ADCs nos pinos 26 e 28 para ler as tensões correspondentes aos sensores de tensão e corrente.
- Total, Volts e Amp: Inicializa variáveis para armazenar os valores de totalizações, tensão e corrente.
- voltimeter(): Função que exibe a leitura de tensão no display OLED:
- oled.fill(0): Limpa o display.
- oled.text("Voltimetro", 24, 0): Exibe o texto "Voltímetro" no display.
- oled.text("Volts: {:.3f}V".format(Volts), 2, 18): Exibe a leitura de tensão formatada.
- oled.show(): Atualiza o display.
- sleep(0.1): Aguarda 0.1 segundos.
- amperimeter(): Função que exibe a leitura de corrente no display OLED:
- oled.fill(0): Limpa o display.
- oled.text("Amperimetro", 24, 0): Exibe "Amperímetro" no display.
- oled.text("Current: {:.3f}A".format(Amp), 2, 18): Exibe a leitura de corrente formatada.
- oled.show(): Atualiza o display.
- sleep(0.1): Aguarda 0.1 segundos.
- Alert(): Função que exibe uma mensagem de alerta no display OLED:
- oled.fill(0): Limpa o display.
- oled.text("Amperimetro", 24, 0): Exibe o texto "Amperímetro" no display.
- oled.text("Pontas Invertidas!", 2, 20): Exibe a mensagem de alerta.
- sleep(1): Aguarda 1 segundo.
- oled.show(): Atualiza o display.
- Pressed(): Monitora o estado do botão e alterna o valor de Run.
- global Run: Declara a variável Run como global.
- while True: Loop infinito.
- if Button.value() == 0: Verifica se o botão foi pressionado (valor 0).
- Run = not Run: Alterna o valor de Run.
- sleep(0.003): Aguarda 0.003 segundos.
- _thread.start_new_thread(Pressed, ()): Inicia a função Pressed em uma nova thread.
- Loop Principal - Voltímetro: Executa o voltímetro quando Run é True:
- while True: Loop infinito.
- while (Run == True): Executa enquanto Run é True.
- Total = 0: Inicializa Total.
- for i in range(50): Loop para ler 50 vezes o valor do ADC.
- Volts1 = SensorVolts1.read_u16(): Lê o valor do ADC.
- Total += Volts1: Soma o valor lido ao Total.

- sleep(0.01): Aguarda 0.01 segundos.
- Total = Total / 50: Calcula a média dos valores lidos.
- Volts = Total * (3.3 / 65535): Converte o valor médio para volts.
- Volts = (Volts - 1.623) * 14.9981: Ajusta a leitura de tensão .
- voltimeter(): Chama a função voltimeter para exibir a tensão no display.
- Loop Principal - Amperímetro: Executa o amperímetro quando Run é False:
- while (Run == False): Executa enquanto Run é False.
- Total = 0: Inicializa Total.
- for i in range(50): Loop para ler 50 vezes o valor do ADC.
- Amp1 = SensorCurrent.read_u16(): Lê o valor do ADC.
- Total += Amp1: Soma o valor lido ao Total.
- sleep(0.01): Aguarda 0.01 segundos.
- Total = Total / 50: Calcula a média dos valores lidos.
- Amp = Total * (3.3 / 65535): Converte o valor médio para amperes.
- Amp = (Amp - 1.411) * 0.985: Ajusta a leitura de corrente.
- if Amp < 0: Verifica se a leitura de corrente é negativa.
- Alert(): Chama a função Alert para exibir um alerta se a leitura for negativa.
- else: Caso contrário,
- amperimeter(): Chama a função amperimeter para exibir a corrente no display.

Esse circuito poderá ter um melhor desempenho em uma placa PCB, já que, como vimos, temos quedas de tensão na própria protoboard. Lembrando que temos os dois circuitos, voltímetro e amperímetro, na qual podemos alternar apenas com o botão. Os testes foram realizados no nosso canal de forma mais detalhada, como mostra a Figura 80.

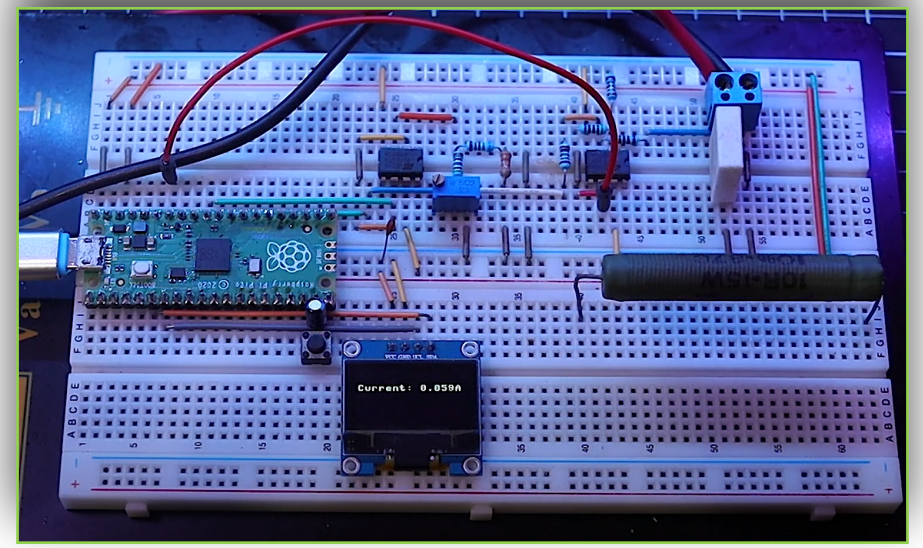


Figura 80: Circuito do amperímetro e voltímetro na bancada.

Aula 22 – Capacímetro usando Arduino IDE

Programar o Raspberry Pi Pico na Arduino IDE é um processo que requer algumas etapas para configurar corretamente o ambiente de desenvolvimento. Seguindo esses passos, estaremos aptos a programar seu Raspberry Pi Pico usando a Arduino IDE. Aqui está um guia passo a passo para fazer isso:

- Passo 1: Instalar a Arduino IDE: Baixe a Arduino IDE do site oficial arduino.cc e instale-a no seu computador.
- Passo 2: Adicionar Suporte ao Raspberry Pi Pico: Abra a Arduino IDE. Vá para File > Preferences (no Windows) ou Arduino > Preferences (no macOS). No campo "Additional Boards Manager URLs", adicione: https://github.com/earlephilhower/arduinopico/releases/download/global/package_rp2040_index.json. Se já houver outros URLs, separe-os por vírgulas. Clique em "OK" para fechar a janela de Preferências.
- Passo 3: Instalar a Placa do Raspberry Pi Pico:Vá para Tools > Board > Boards Manager. Na janela que abrir, digite "Pico" na barra de pesquisa. Encontre a entrada "Raspberry Pi RP2040 Boards" de Earle Philhower e clique em "Install".
- Passo 4: Configurar a Placa e a Porta:
- Conecte seu Raspberry Pi Pico ao computador usando um cabo micro USB. Vá para Tools > Board e selecione "Raspberry Pi Pico". Em Tools > Port, selecione a porta serial correspondente ao seu Raspberry Pi Pico.
- Passo 5: Carregar um Exemplo: Vá para File > Examples > 01.Basics > Blink para abrir o exemplo Blink. Clique no ícone de upload (seta para a direita) para compilar e carregar o código no seu Raspberry Pi Pico.
- Passo 6: Verificar a Conexão: O LED embutido no Raspberry Pi Pico deve começar a piscar, indicando que o código foi carregado com sucesso.

Código Exemplo Blink:

```
void setup() {
pinMode(LED_BUILTIN, OUTPUT);          //Configura o Led como saída
 }

void loop() {
digitalWrite(LED_BUILTIN, HIGH);       // liga o LED
delay(1000);                           // espera por um segundo
digitalWrite(LED_BUILTIN, LOW);        // desliga o LED
delay(1000);                           // espera por um segundo
 }
```

Dicas Adicionais:

- Drivers: Certifique-se de que os drivers USB do seu sistema estejam atualizados para garantir que a IDE reconheça a placa corretamente.
- Documentação: Consulte a documentação oficial e fóruns comunitários para resolver qualquer problema que possa surgir durante o processo de configuração.

22.1 - Constante de tempo de um capacitor

A constante de tempo de um capacitor, geralmente representada pela letra grega τ (tau), é um conceito fundamental em circuitos RC (resistor-capacitor). Ela define a rapidez com que um capacitor carrega ou descarrega através de um resistor. A constante de tempo τ é dada pelo produto da resistência R e da capacitância C no circuito, e representa o tempo necessário para que a tensão no capacitor atinja aproximadamente 63% da tensão da fonte (Vmax.).

$$T=R\times C$$

Onde:

- R é a resistência em ohms (Ω).
- C é a capacitância em farads (F).
- τ é o tempo em segundos(S).

Quando um capacitor descarrega através de um resistor, a tensão V no capacitor como função do tempo. Neste caso, τ representa o tempo necessário para que a tensão no capacitor caia para aproximadamente 37% de Vmax. Após um tempo igual a 5τ(tau), o capacitor está praticamente totalmente carregado ou descarregado. Por exemplo, se você tiver um resistor de 1 kΩ e um capacitor de 1 μF (1 x 10^-6 F), a constante de tempo τ será:

$$\tau=R\times C = 1000\ \Omega \times 1\times 10{-}6\ F = 0{,}001\ s = 1\ ms$$

Isso significa que, em 1 milissegundo, a tensão no capacitor atingirá aproximadamente 63% de sua tensão máxima durante o carregamento, ou cairá para aproximadamente 37% de sua tensão inicial durante o descarregamento.

22.2 – Função millis()

A função millis() no Arduino é uma função de temporização que retorna o número de milissegundos que se passaram desde que o Arduino começou a executar o programa atual. Esta função é extremamente útil para medir intervalos de tempo sem bloquear a execução do código, como ocorre com a função delay(). A função millis() retorna um valor do tipo unsigned long, que representa o tempo em milissegundos desde que a placa Arduino foi ligada ou reiniciada. Esse valor é armazenado em um registro de 32 bits, o que

permite a medição de até aproximadamente 49,71 dias antes de ocorrer um overflow (reinício do contador para 0). A função millis() é frequentemente usada para realizar tarefas de temporização não bloqueante. Aqui temos um exemplo de como você pode usar millis() para piscar um LED sem bloquear a execução do código na Arduino IDE:

```
const int ledPin = 13;                  // Pino do LED
unsigned long previousMillis = 0;  // Armazena a última vez que o LED mudou
de estado
const long interval = 1000;          // Intervalo em milissegundos (1 segundo)
void setup() {
  pinMode(ledPin, OUTPUT);        // Define o pino do LED como saída
}
void loop() {
unsigned long currentMillis = millis();  // Obtém o tempo atual em
milissegundos
if (currentMillis - previousMillis >= interval) {
    previousMillis = currentMillis;                 // Salva o tempo atual
    digitalWrite(ledPin, !digitalRead(ledPin)); // Altera o estado do LED
  }
}
```

Nesse exemplo, o LED no pino 13 pisca a cada 1 segundo. previousMillis armazena o tempo da última vez que o LED mudou de estado. currentMillis armazena o tempo atual. Se o tempo decorrido desde a última mudança de estado (currentMillis - previousMillis) for maior ou igual ao intervalo desejado (1 segundo), o estado do LED é alterado e previousMillis é atualizado para o tempo atual.

22.3 – Circuito do capacímetro

Para medir a capacitância usando nossa placa e a função millis(), vamos usar um circuito RC (resistor-capacitor) e medir o tempo que leva para o capacitor chega em 63% do valor da alimentação. A constante de tempo τ é usada para calcular a capacitância, dada a resistência conhecida. Assim, quando iniciar a carga do capacitor, iremos iniciar um contador com a função millis, vamos ler a tensão pelo ADC e esperar ela chegar em 63% de 3.3 V, que a tensão que iremos utilizar, o que nos dá 2.08 V, o que funcionará bem, já que nosso ADC vai até 3.3 V. Na Arduino IDE iremos trabalhar com 10 bits ou seja, de 0 até 1023, isso porque a biblioteca para Raspberry Pi Pico faz esse ajuste, pois é o padrão do Arduino. Logo, para 2.08 V temos o valor aproximado de 648. Usaremos um resistor de 10kΩ para a carga e um resistor simples de 220Ω para descarga. Assim, nosso circuito ficará como mostra a Figura 81 e na Figura 82 temos os testes realizados no nosso canal no Youtube.

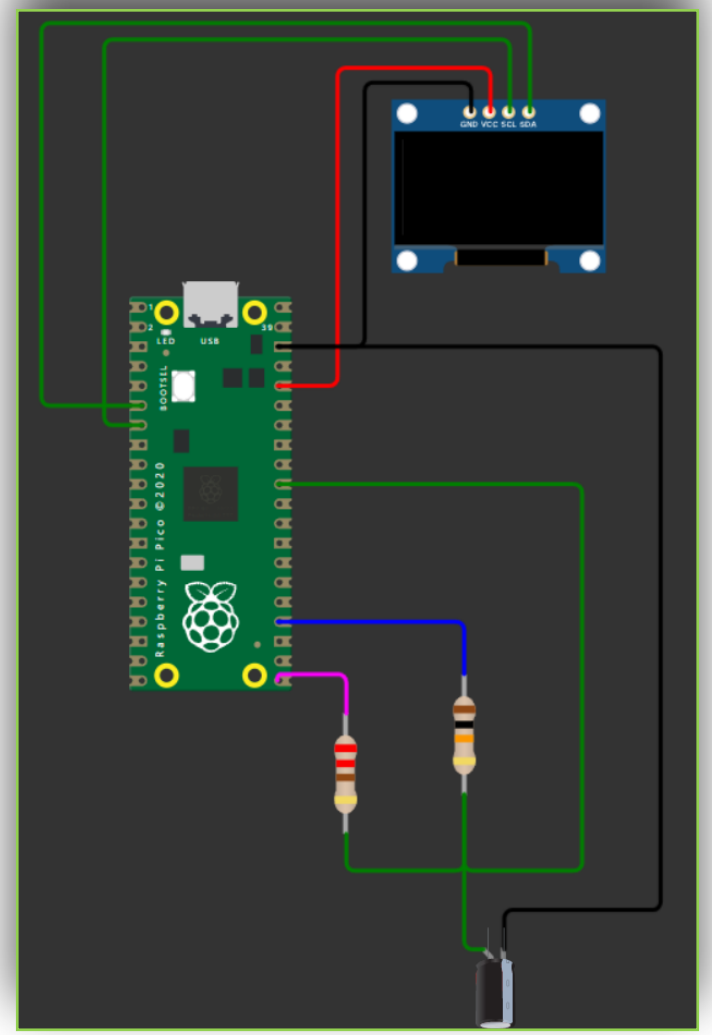


Figura 81: Circuito do capacímetro

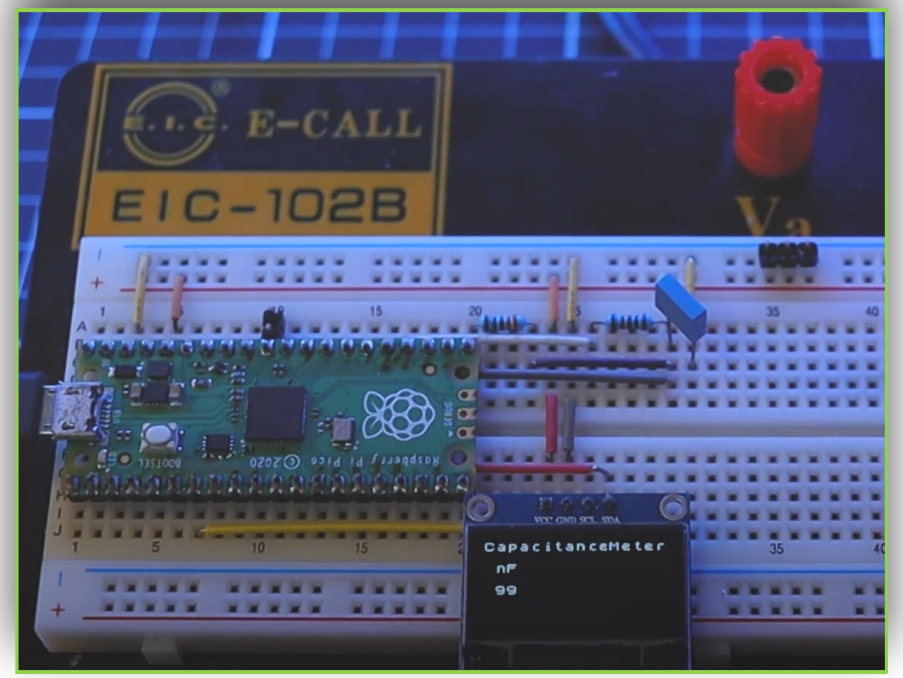


Figura 82: Testes realizamos no canal Engenharia entendida

22.4 – Código principal

Para essa IDE, precisamos usar uma biblioteca para o display, assim como usamos o drive SSD1306, vamos usar a ACROBOTIC_SSD1306.h e Wire.h para comunicação i2c. Basta irmos até library menager, no lado esquerdo, pesquisar o nome da biblioteca e instalar, como mostra a Figura 82.

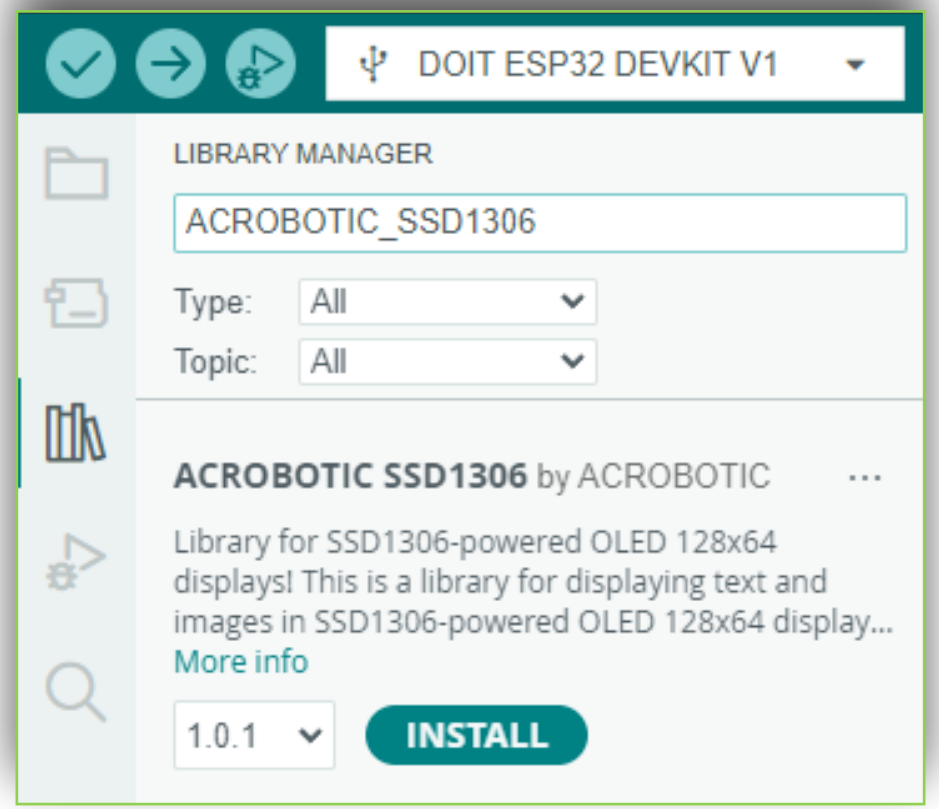


Figura 82: Adicionando bibliotecas na Arduino IDE.

Agora estamos trabalhando com C++, uma linguagem também simples e muito usada em microcontroladores, vamos explicar linha a linha nosso código:

```
#include <Wire.h>
#include "ACROBOTIC_SSD1306.h"
#define CapacitorRead 26
#define Carga 18
#define Descarga 16
int ResistorValue = 10000;
unsigned long StartTime;
unsigned long EndTime;
unsigned long Time;
float Capacitance;
void setup() {
  Wire.begin();
  oled.init();
  oled.clearDisplay();
  oled.setTextXY(0, 0);
```

```
  oled.putString("CapacitanceMeter");
  pinMode(Carga, OUTPUT);
  digitalWrite(Carga, LOW);
}
void loop() {
  digitalWrite(Carga,HIGH);
  StartTime = millis();
  while(analogRead(CapacitorRead) < 648)
    {      //Até que a tensão chegue em 63% de 3.3V.
    }
  EndTime = millis();
  Time = EndTime - StartTime;
  Capacitance = ((float)Time/ResistorValue)*1000;
 if (Capacitance > 1)
 {
  oled.clearDisplay();
  oled.setTextXY(0, 0);
  oled.putString("CapacitanceMeter");
  oled.setTextXY(2, 1);
  oled.putString("uF");
  oled.setTextXY(4, 1);
  oled.putNumber(Capacitance);
  delay(2000);
 }
 else
 {
  Capacitance = Capacitance*1000;
  oled.clearDisplay();
  oled.setTextXY(0, 0);
  oled.putString("CapacitanceMeter");
  oled.setTextXY(2, 1);
  oled.putString("nF");
  oled.setTextXY(4, 1);
  oled.putNumber(Capacitance);
  delay(2000);
 }
  digitalWrite(Carga,LOW);
  pinMode(Descarga,OUTPUT);
  digitalWrite(Descarga,LOW);
  while(analogRead(CapacitorRead) > 0)
    {      //Espera o capacitor descarregar
    }
  pinMode(Descarga, INPUT);
  delay(500);
}
```

Explicando linha a linha:

Inclusão de Bibliotecas:

- As bibliotecas Wire.h e ACROBOTIC_SSD1306.h são incluídas para permitir a comunicação I2C e o controle do display OLED.

Definição dos Pinos:

- CapacitorRead (pino 26) é usado para ler a tensão no capacitor.
- Carga (pino 18) é usado para carregar o capacitor.
- Descarga (pino 16) é usado para descarregar o capacitor.

Declaração de Variáveis:

- ResistorValue é definido como 10k ohms, que é o valor do resistor no circuito.
- StartTime, EndTime, e Time são variáveis usadas para armazenar os tempos de início, fim e o tempo total da medição.
- Capacitance é a variável que armazenará o valor calculado da capacitância.

Função setup():

- Wire.begin(): Inicializa a comunicação I2C.
- oled.init(): Inicializa o display OLED.
- oled.clearDisplay(): Limpa o display OLED.
- oled.setTextXY(0, 0): Define a posição do cursor no display (linha 0, coluna 0).
- oled.putString("CapacitanceMeter"): Exibe "CapacitanceMeter" no display.
- pinMode(Carga, OUTPUT): Define o pino de carga como saída.
- digitalWrite(Carga, LOW): Define o pino de carga como LOW (desligado).

Função loop():

Carregamento do Capacitor:

- digitalWrite(Carga, HIGH): Liga o pino de carga (começa a carregar o capacitor).
- StartTime = millis(): Armazena o tempo de início da carga.
- while (analogRead(CapacitorRead) < 648): Aguarda até que a tensão no capacitor atinja 63% de 3.3V, que dá aproximadamente 2.08V, ou 648 em uma leitura de 10 bits.

Cálculo do Tempo e Capacitância:

- EndTime = millis(): Armazena o tempo atual em milissegundos (ms) desde que o programa começou a ser executado. Este é o tempo de fim do processo de carregamento do capacitor.
- Time = EndTime - StartTime: Calcula o tempo total de carregamento do capacitor subtraindo o tempo de início (StartTime) do tempo de fim (EndTime).
- Capacitance = ((float)Time / ResistorValue) * 1000: Calcula a capacitância do capacitor. O tempo de carregamento (Time) é dividido pelo valor do resistor (ResistorValue) para obter a capacitância em farads. Multiplica-se por 1000 para converter o valor para microfarads (µF).

Exibição do Resultado:

Se a capacitância for maior que 1 (µF):

- oled.clearDisplay();: Limpa o display.
- oled.setTextXY(0, 0);: Define a posição do cursor.
- oled.putString("CapacitanceMeter");: Exibe "CapacitanceMeter".
- oled.setTextXY(2, 1);: Define a posição do cursor.
- oled.putString("uF");: Exibe "uF".
- oled.setTextXY(4, 1);: Define a posição do cursor.
- oled.putNumber(Capacitance);: Exibe a capacitância medida.
- delay(2000);: Aguarda 2 segundos.

Se a capacitância for menor ou igual a 1 (µF):

- Capacitance = Capacitance * 1000;: Converte a capacitância para nF.
- oled.clearDisplay();: Limpa o display.
- oled.setTextXY(0, 0);: Define a posição do cursor.
- oled.putString("CapacitanceMeter");: Exibe "CapacitanceMeter".
- oled.setTextXY(2, 1);: Define a posição do cursor.
- oled.putString("nF");: Exibe "nF".
- oled.setTextXY(4, 1);: Define a posição do cursor.
- oled.putNumber(Capacitance);: Exibe a capacitância medida.
- delay(2000);: Aguarda 2 segundos.

Descarregamento do Capacitor:

- digitalWrite(Carga, LOW);: Desativa a carga do capacitor.
- pinMode(Descarga, OUTPUT);: Define o pino de descarga como saída.
- digitalWrite(Descarga, LOW);: Ativa a descarga do capacitor.
- while (analogRead(CapacitorRead) > 0) {}: Espera até que o capacitor esteja completamente descarregado.
- pinMode(Descarga, INPUT);: Redefine o pino de descarga como entrada.
- delay(500);: Aguarda 500 ms antes de iniciar a próxima medição.

Aula 23 – Indutímetro usando Arduino IDE

Um indutímetro baseado em Arduino IDE usando c++ usa o princípio de um circuito LC (indutor-capacitor) para medir a indutância de um indutor. O circuito LC ressoa em uma frequência específica que depende dos valores do indutor (L) e do capacitor (C). Ao medir essa frequência de ressonância, podemos calcular a indutância do indutor desconhecido. Vamos entender alguns aspectos.

23.1 – Conceito de Ressonância

Imagine que você tem uma criança em um balanço. Quando você empurra a criança no ritmo certo, ela vai mais alto a cada empurrão. Esse ritmo específico em que o balanço oscila é a "frequência de ressonância" do balanço. Se você empurrar no ritmo certo (frequência), a criança balança de

forma suave e alta. Da mesma forma, em um circuito LC, existe uma frequência específica em que ele "balança" ou oscila naturalmente. Quando você retira a alimentação no circuito LC, ele começa a oscilar na sua frequência natural de ressonância, até que a amplitude dessa oscilação chegue a 0, como mostra a Figura 83.

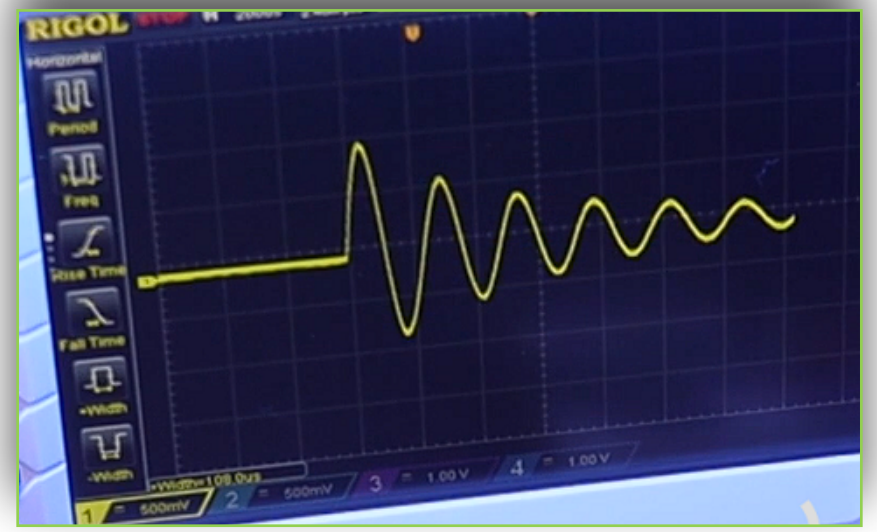


Figura 83: Ressonância do circuito LC.

A frequência de ressonância (f) de um circuito LC é dada pela fórmula:

$$f = \frac{1}{2\pi\sqrt{LC}}$$

Onde:

- f é a frequência de ressonância em Hertz (Hz)
- L é a indutância do indutor em Henrys (H)
- C é a capacitância do capacitor em Farads (F)

A ressonância acontece porque a energia oscila entre os dois componentes de maneira natural. O capacitor (C) e o indutor (L) estão em um estado de perfeita sincronia. A energia elétrica no capacitor é transformada em energia magnética no indutor e vice-versa, de forma cíclica.

Quando o capacitor está totalmente carregado, ele começa a descarregar sua energia elétrica no indutor. O indutor armazena essa energia como um campo magnético e começa a liberar de volta ao capacitor. Este ciclo continua, com a energia oscilando entre o campo elétrico do capacitor e o campo magnético do indutor. A frequência de ressonância é a frequência em que essa transferência de energia acontece de forma mais eficiente. É como empurrar o balanço no ritmo certo, onde ele vai mais alto a cada empurrão. Se tivéssemos um circuito perfeito esse ciclo seria infinito, porém, temos resistências dos componentes, por isso que a amplitude vai diminuindo, até chegar a 0.

23.2 – Calculando a indutância

Para calcular nosso indutor, vamos, primeiro, gera um pulso curto de energia para iniciar a oscilação no circuito LC. Após o pulso inicial, o circuito LC começa a oscilar em sua frequência natural. Nossa placa mede o tempo que o circuito LC leva para completar um ciclo de oscilação, através da função millis(). Esse tempo é dividido em duas partes: tempo em nível alto e tempo em nível baixo, logo, tempo total para um ciclo completo é a soma do tempo em nível alto e do tempo em nível baixo. A frequência é o inverso desse período total. Uma vez que temos a frequência de ressonância, podemos rearranjar a fórmula para encontrar a indutância L:

$$L = \frac{1}{(2\pi f)^2 . C}$$

Por exemplo, se temos um capacitor C=1000 nF = 1000×10^{-9} e um Indutor L que queremos medir. Vamos envia um pulso inicial, e o circuito LC começa a oscilar. O Pi Pico mede os tempos e calcula a frequência de oscilação. Digamos que a frequência medida foi de 5 kHz (5000 Hz), temos:

$$L = \frac{1}{(2\pi 5000)^2 . 1000 \times 10^{-9}}$$

$$L = \frac{1}{39.478}$$

$$L = 25.33\ uH$$

23.3 – Circuito para indutímetro

No nosso circuito, teremos um capacitor conhecido de 1uF, usaremos o lm358 como nosso operacional. O pulso inicial será enviado pelo pino 21, porém, iremos adicionar um diodo 1N4001 para evitar tensões reversas e o pino 16 irá receber o sinal de ressonância. De forma resumida nosso código mede a indutância de um indutor e exibe o valor em um display OLED. Inicialmente, as bibliotecas necessárias para comunicação I2C e controle do display OLED são incluídas, e os pinos para leitura de pulso e carga são definidos. No setup, a comunicação I2C e o display OLED são inicializados, e os pinos são configurados como entrada e saída.

No loop, um pulso é gerado e medido. Com base no tempo do pulso, a frequência é calculada, e a indutância é determinada usando para indutância baseado na frequência e capacitância conhecida. Se a leitura do pulso for válida, o valor da indutância é exibido no display em mH. Caso contrário, uma mensagem solicitando a adição de um indutor é exibida. O display é atualizado a cada segundo para refletir as medições mais recentes. Nosso circuito completo é mostrado na Figura 84.

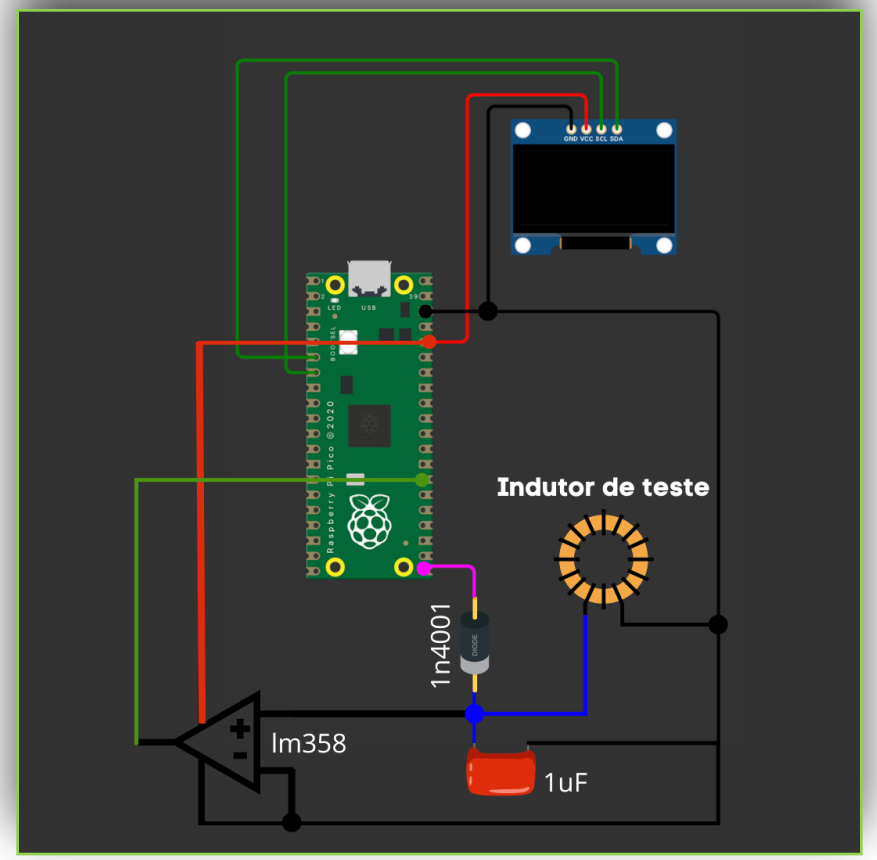


Figura 84: Circuito do indutímetro.

23.4 – Histerese com comparador

A histerese em um comparador é usada para criar uma faixa de tensão dentro da qual o comparador não muda de estado, evitando flutuações rápidas e ruído na saída. Isso é especialmente útil quando se está convertendo um sinal senoidal de ressonância de um circuito LC em uma onda retangular. Histerese é a adição de um intervalo de tensão (faixa de histerese) que define os limites superior e inferior para o ponto de transição do comparador. Esse intervalo ajuda a garantir que o comparador não alterne sua saída com pequenas variações ou ruídos no sinal de entrada, já que o sinal de ressonância, terá vários valores. Histerese é a adição de um intervalo de tensão (faixa de histerese) que define os limites superior e inferior para o ponto de transição do comparador. Esse intervalo ajuda a garantir que o comparador não alterne sua saída com pequenas variações ou ruídos no sinal de entrada. Com histerese, o comparador possui dois pontos de referência diferentes: um para quando o sinal de entrada está subindo (limite superior) e outro para quando o sinal está descendo (limite inferior). Quando o sinal de entrada sobe e ultrapassa o limite superior, a saída do comparador muda para o nível alto. Quando o sinal de entrada desce e cai abaixo do limite inferior, a saída muda para o nível baixo. A diferença entre esses dois limites cria uma faixa onde o comparador não muda de estado, evitando transições rápidas e ruído, como mostra a Figura 85.

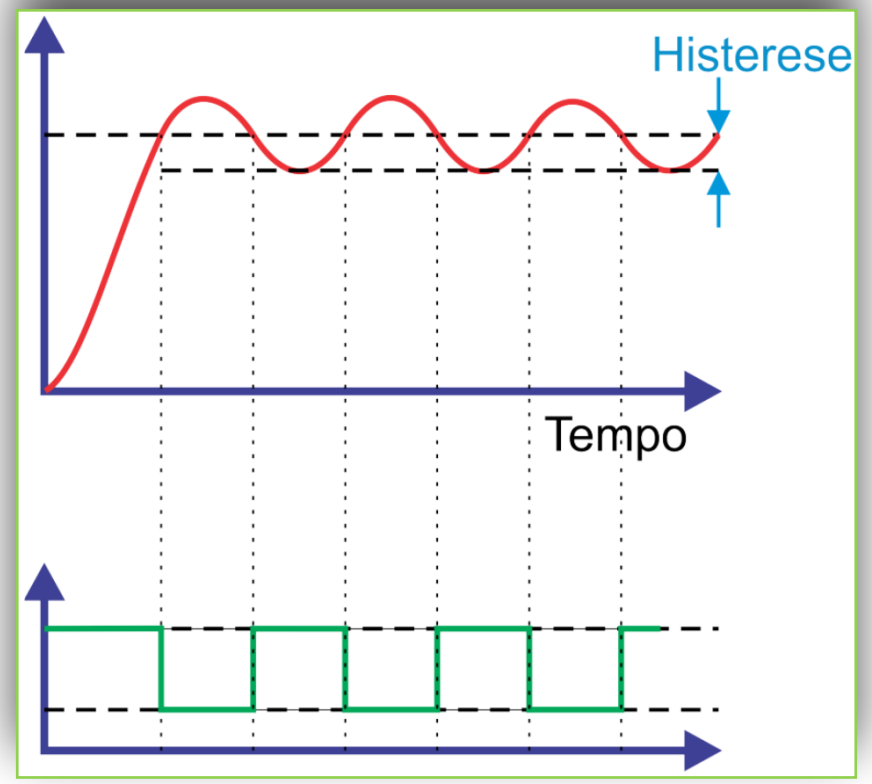


Figura 85: Funcionamento de um comparador com Histerese.

A Histerese garante que a saída do comparador mude de estado apenas quando o sinal de entrada estiver claramente acima ou abaixo dos limites, evitando transições indesejadas. Melhora a forma de onda transformando um sinal senoidal em uma onda retangular, a histerese melhora a clareza e a precisão da onda retangular, especialmente se o sinal senoidal estiver próximo ao ponto de cruzamento e sofrer pequenas variações. Um sinal retangular é lido com nossa placa de forma simples, pois ele está bem definido com essa configuração. Como nosso operacional está referenciado em 0V, teremos apenas uma onda retângular com valores positivos somente.

23.5 - Código do indutímetro

```
#include <Wire.h>
#include "ACROBOTIC_SSD1306.h"
#define PulseRead 21
#define chargePin 16
float Capacitance = 1.0e-6;
float inductance ;
double pulse ;
double frequency ;
void setup() {
  Wire.begin();
  oled.init();
  pinMode(chargePin, OUTPUT);
  pinMode(PulseRead, INPUT);
}
```

```
void loop() {
  digitalWrite(chargePin, HIGH);
  delay(5);
  digitalWrite(chargePin, LOW);
  delayMicroseconds(5);
  pulse = pulseIn(PulseRead,HIGH);
  if(pulse > 0.1){
  frequency = 1.E6/(2*pulse) ;
  inductance=1./(4*Capacitance*pow(frequency , 2 )*pow(3.14159 , 2));
  inductance=inductance*1000;
  oled.clearDisplay();
  oled.setTextXY(0, 1);
  oled.putString("InductorMeter");
  oled.setTextXY(2, 1);
  oled.putString("mH");
  oled.setTextXY(4, 1);
  oled.putFloat(inductance);
  delay(1000);
  }
  else {
  oled.clearDisplay();
  oled.setTextXY(0, 1);
  oled.putString("InductorMeter");
  oled.setTextXY(2, 1);
  oled.putString("Add a inductor");
  delay(1000);
  }
}
```

Vamos Inclui a biblioteca Wire.h, que é necessária para a comunicação I2C e a biblioteca ACROBOTIC_SSD1306.h, que é usada para controlar o display OLED. Definimos PulseRead para leitura do pulso no pino 21 e chargePin para o pino 16, que recebe o sinal de ressonância. Declaramos as variáveis Capacitance para a capacitância (1 uF), inductance para a indutância, pulse para o tempo do pulso medido e frequency para a frequência calculada. Temos então:

- Wire.begin() inicializa a comunicação I2C.
- oled.init() inicializa o display OLED.
- pinMode(chargePin, OUTPUT) configura o pino chargePin como saída.
- pinMode(PulseRead, INPUT) configura o pino PulseRead como entrada.
- digitalWrite(chargePin, HIGH) define o pino chargePin como alto.
- delay(5) espera 5 milissegundos.
- digitalWrite(chargePin, LOW) define o pino chargePin como baixo.
- delayMicroseconds(5) espera 5 microssegundos.

- pulse = pulseIn(PulseRead,HIGH) Usamos a função pulseIn() para medir a duração do pulso em PulseRead enquanto está alto. O valor é armazenado em pulse.
- if(pulse > 0.1) Verifica se o tempo do pulso é maior que 0.1 microssegundo.
- frequency = 1.E6/(2*pulse) Calcula a frequência com base no tempo do pulso. A fórmula é 1E6 / (2 * pulse), onde 1E6 é 1 milhão (para converter microssegundos em segundos).
- inductance = 1. / (4 * Capacitance * pow(frequency , 2) * pow(3.14159 , 2)); inductance = inductance * 1000; Calcula a indutância usando a fórmula para circuitos LC, onde Capacitance é a capacitância e frequency é a frequência. O resultado é convertido de henries para milihenries.

Atualiza o display OLED:

- oled.clearDisplay() limpa o display.
- oled.setTextXY(x, y) define a posição do texto.
- oled.putString("InductorMeter") exibe o título "InductorMeter".
- oled.putString("mH") exibe a unidade "mH" (milihenries).
- oled.putFloat(inductance) exibe o valor da indutância.
- delay(1000) espera 1 segundo antes da próxima atualização.

No nosso else se o tempo do pulso for menor ou igual a 0.1 microssegundo, vamos Limpar o display e exibir a mensagem "Add a inductor" para indicar que um indutor deve ser adicionado e Espera 1 segundo antes de repetir o loop. O circuito em teste da bancada é mostrado na Figura 86.

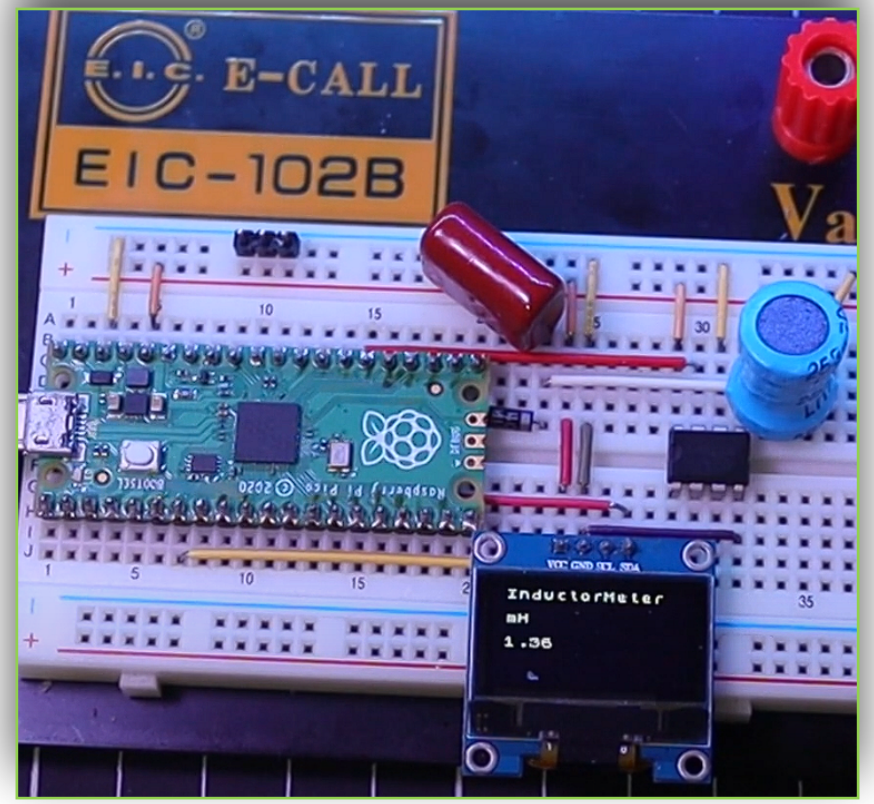


Figura 86: Teste de bancada.

Considerações Finais

Chegamos ao final desta jornada exploratória pelo mundo do Raspberry Pi Pico. Ao longo deste livro, abordamos desde os conceitos básicos, como a introdução ao hardware e o ambiente de desenvolvimento MicroPython, até projetos intermediários mais complexos, como amperímetros, capacímetros e indutímetros.

No Módulo 1, começamos com os fundamentos do Raspberry Pi Pico, explorando seu hardware, configurando o ambiente de desenvolvimento e realizando os primeiros passos com códigos simples. Em seguida, no Módulo 2, mergulhamos em uma variedade de projetos práticos, desde semáforos e sensores até controle de motores e conexões via Wi-Fi. No Módulo 3 em projetos intermediários, ampliamos nosso conhecimento com dispositivos adicionais e técnicas avançadas, desafiando nossa compreensão e habilidades com projetos como o voltímetro e o indutímetro.

Espero que este livro tenha sido uma fonte valiosa de aprendizado e inspiração, fornecendo uma base sólida para seus projetos com o Raspberry Pi Pico e estimulando sua curiosidade para continuar explorando e inovando.

Agradeço profundamente a você, leitor, por acompanhar esta jornada. Sua dedicação e interesse são o que tornam todo o esforço de escrever e compartilhar este conhecimento tão gratificante. Que suas futuras criações sejam ainda mais brilhantes e impactantes.

Com gratidão,

Eng. Dhanuzio A.A.

Tem alguma sugestão ou encontrou algum erro de gramática? Fique à vontade para compartilhar suas observações e ajudar a melhorar este material!

Email: engenhariaentendidacontato@gmail.com

Forte abraço .