embarca tech
Residência Tecnológica em Sistemas Embarcados

Conversores A/D

Unidade 4 | Capítulo 8

Pedro Henrique Almeida Miranda

Executores:
INSTITUTO FEDERAL Piauí
INSTITUTO FEDERAL Rio Grande do Norte
INSTITUTO FEDERAL Maranhão
INSTITUTO FEDERAL Ceará
cepedi
INSTITUTO HARDWARE BR

Coordenação: Softex

Iniciativa: MINISTÉRIO DA CIÊNCIA, TECNOLOGIA E INOVAÇÃO
GOVERNO FEDERAL BRASIL UNIÃO E RECONSTRUÇÃO

Sumário

1. Boas-vindas e introdução

Olá, estudante do EmbarcaTech!
Seja bem-vindo a mais uma etapa do nosso curso de capacitação. Nesta unidade, vamos explorar práticas avançadas sobre o uso do ADC (Conversor Analógico-Digital). O foco desta aula tem como objetivo proporcionar uma compreensão detalhada de como o ADC funciona no Raspberry Pi Pico W. Exploraremos o ADC no RP2040, abordando suas características, exemplos práticos, e como utilizá-lo em diferentes projetos de sistemas embarcados.

Com base nos conhecimentos adquiridos anteriormente sobre interrupções e PWM, estamos prontos para dar mais um passo em direção ao desenvolvimento de sistemas embarcados. Nesta aula, você terá a oportunidade de aplicar a programação em C para realizar projetos práticos e desafiadores. Vamos aprender a realizar conversões analógico-digitais de forma eficiente e precisa. Aproveite este material para aprofundar seus conhecimentos e experimente configurar o ADC no seu Raspberry Pi Pico W. Vamos explorar juntos o potencial desse incrível microcontrolador!

Vamos lá?

2. Objetivos desta Unidade

Ao final desta aula, espera-se que você tenha desenvolvido a capacidade de compreender o processo de conversão A/D, configurar e implementar o módulo ADC e integrá-lo com outros módulos do Raspberry Pi Pico [2] ou com a plataforma educacional BitDogLab[1]. As práticas abordadas neste Ebook permitirão que você desenvolva habilidades essenciais para solucionar problemas comuns em sistemas embarcados, como a leitura precisa de sensores por meio do ADC. Ao dominar essas técnicas, você estará preparado para aplicá-las em projetos práticos, utilizando tanto o Raspberry Pi Pico quanto a BitDogLab. Isso ampliará sua capacidade de criar soluções inovadoras e eficientes para desafios reais no desenvolvimento de sistemas embarcados.

3. O que é A/D?

Olá! Vamos começar entendendo o que é a conversão A/D (Analógico-

Digital) e como ela funciona. Imagine que você tem um potenciômetro que ajusta o volume de um som. Quando você gira o potenciômetro, ele varia a tensão entre 0 e 3,3 volts. O problema é que o microcontrolador, como o RP2040, não entende tensões. Ele só entende números. É aí que entra o ADC!

Conversão A/D é o processo de transformar um sinal analógico (como a variação de tensão do potenciômetro) em um valor digital, que é um número que o microcontrolador pode entender e processar. No caso do RP2040, esse número pode variar de 0 a 4095, porque ele usa um ADC de 12 bits. Isso significa que a tensão de 0 a 3,3V é dividida em 4096 passos discretos.

3.1 Como funciona na prática

- **Sinal de Entrada:** Um sensor, como um potenciômetro, gera um sinal de tensão analógico.
- **Amostragem:** O ADC "tira uma foto" dessa tensão em um determinado instante.
- **Quantização:** A tensão amostrada é mapeada para o valor digital mais próximo, dentro dos 4096 níveis possíveis.
- **Resultado:** Esse valor digital é enviado para o processador, onde ele pode ser usado para tomar decisões ou acionar dispositivos.

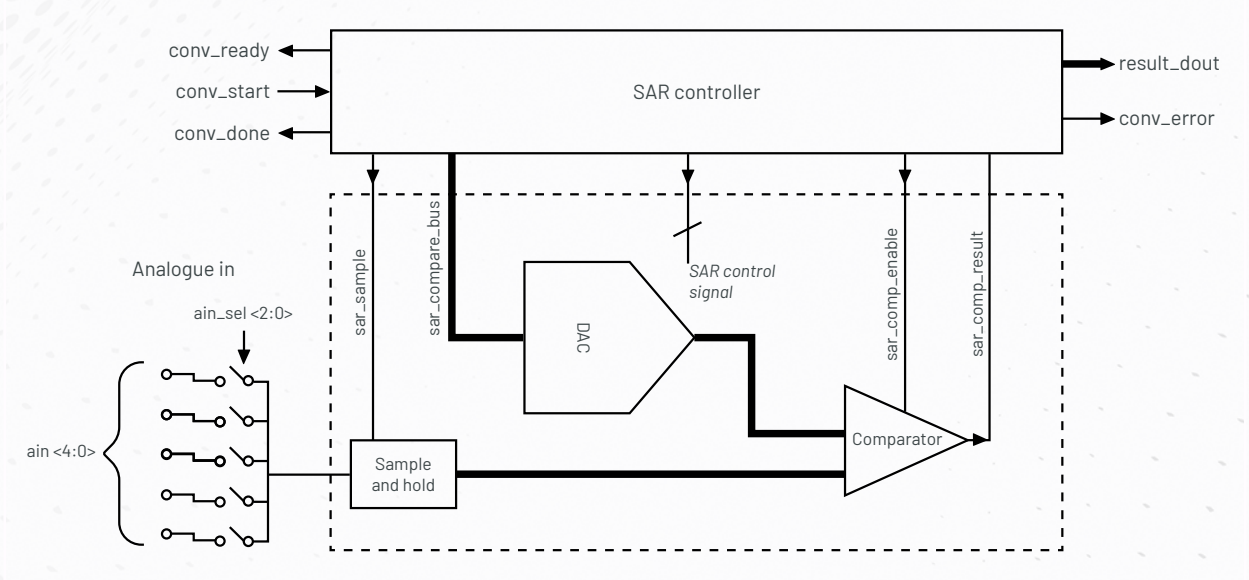
Por exemplo, se o potenciômetro está ajustado para 1,65V, que é metade de 3,3V, o valor digital correspondente seria cerca de 2048 (metade de 4096).

4. Compreender o funcionamento do móduclo A/D

O módulo ADC do RP2040 é o responsável por fazer essa conversão A/D. Ele possui até cinco canais, mas apenas três são diretamente acessíveis pelos pinos GPIO (GP26, GP27 e GP28), enquanto o canal 4 é dedicado ao sensor de temperatura interno.

4.1 Componentes principais:

Amostragem: É o processo de medir e converter um sinal analógico contínuo em uma série de valores digitais discretos. Durante a amostragem, o ADC captura o valor instantâneo do sinal analógico em intervalos de tempo regulares, chamados de «taxa de amostragem». Cada valor medido é transformado em um número binário correspondente, representando a amplitude do sinal naquele momento específico. Esse processo permite que sinais analógicos, como temperatura, som ou tensão, sejam interpretados e processados digitalmente pelo microcontrolador.



- **Quantização:** É o processo pelo qual os valores contínuos de um sinal analógico, capturados durante a amostragem pelo ADC, são convertidos em valores discretos representados por números binários. Durante a quantização, o valor analógico medido é aproximado para o valor mais próximo de um conjunto finito de níveis pré-definidos. Essa aproximação resulta em uma perda de precisão conhecida como "erro de quantização". Em termos práticos, imagine que o ADC divide a faixa total de entrada analógica em vários níveis discretos, dependendo da sua resolução (por exemplo, 8 bits, 10 bits, 12 bits, etc.). Cada nível corresponde a um valor específico de saída digital. Quanto maior a resolução do ADC, mais níveis ele possui, e mais próxima será a representação digital do valor analógico real, reduzindo o erro de quantização.

- **Canais de Entrada:** O ADC possui canais específicos que correspondem

a pinos do microcontrolador. Cada canal pode ser configurado para ler uma entrada analógica diferente. No RP2040, os canais 0, 1 e 2 correspondem aos pinos GP26, GP27 e GP28, respectivamente.

- **Multiplexador:** É um "selecionador" que escolhe qual canal de entrada será convertido no momento. Apenas um canal pode ser convertido por vez.

- **Taxa de amostragem (ou frequência de amostragem):** É o número de vezes por segundo que um sinal analógico é medido e convertido em um valor digital pelo conversor analógico-digital (ADC). Ela é expressa em hertz (Hz), onde 1 Hz corresponde a uma amostra por segundo. Por exemplo, se a taxa de amostragem de um ADC é de 1 kHz, isso significa que o ADC está capturando 1000 amostras do sinal analógico a cada segundo. Quanto maior a taxa de amostragem, mais fielmente o sinal analógico original pode ser reproduzido na forma digital, capturando mudanças rápidas no sinal. A taxa de amostragem deve ser pelo menos duas vezes maior que a frequência máxima do sinal analógico a ser medido, de acordo com o teorema de Nyquist-Shannon. Caso contrário, pode ocorrer o efeito de aliasing, onde as altas frequências do sinal são representadas incorretamente no domínio digital.

- **Registro de Resultado:** Armazena o valor digital convertido, que pode ser lido pelo processador.

- **Interrupção/DMA:** A integração do DMA (Direct Memory Access) com o ADC permite a transferência automática e eficiente de dados de conversões analógicas para a memória, sem intervenção da CPU. Isso é útil em aplicações que requerem amostragem rápida e contínua, como monitoramento de sensores e captura de sinais, pois reduz a carga de processamento, liberando a CPU para outras tarefas. Quando um número específico de amostras é coletado, o DMA gera uma interrupção para notificar que os dados estão prontos para processamento, garantindo eficiência e precisão na coleta de dados.

- **SAR:** O conversor por aproximações sucessivas, ou SAR, é composto por um circuito de amostragem e retenção, chamado de sample *and hold*, que lê a entrada analógica. Ele também conta com um comparador, um conversor digital-analógico (DAC) e um registrador

de aproximação sucessiva. O processo de conversão é iterativo. O DAC gera valores analógicos que são comparados com a amostra. Se o valor gerado for maior que a amostra, é gerado um 1; se for menor, um 0. Esse valor é então usado para ajustar o registrador de aproximação sucessiva, que aciona o DAC novamente, repetindo o processo até que a amostra seja completamente convertida. Ou seja, até que o valor do registrador seja o mais próximo possível da amostra analógica. Esse processo, no entanto, é relativamente lento, levando 96 ciclos de clock para uma única conversão de 12 bits. Além disso, a precisão é limitada, com apenas os 9 bits mais significativos sendo realmente precisos.

4.2 Conhecer os diferentes modos de configuração do módulo conversor A/D:

O ADC do RP2040 oferece dois modos de operação distintos, cada um com características específicas que atendem a diferentes necessidades de aplicação:

4.2.1 Modo One-Shot

No modo One-Shot, o ADC realiza uma única conversão analógica-digital por vez, ou seja, cada vez que uma conversão é desejada, ela deve ser iniciada manualmente pelo software. Esse modo é útil quando precisamos realizar leituras eventuais e precisamos de maior controle sobre quando as conversões acontecem. Para utilizar o modo One-Shot, o processo de configuração segue quatro etapas:

Função	Uso
Inicialização do módulo ADC	A função adc_init() é utilizada para preparar o módulo ADC para operação, garantindo que ele esteja pronto para realizar conversões.
Configuração do GPIO como entrada analógica	Utiliza-se adc_gpio_init(gpio), que configura o pino GPIO correspondente para operar como uma entrada analógica.

Seleção da entrada analógica	Com adc_select_input(channel), escolhe-se qual canal de entrada analógica será utilizado para a próxima conversão. Os canais são numerados de acordo com os pinos analógicos do microcontrolador.
Realização da conversão	A função adc_read() inicia a conversão e retorna o valor digital correspondente ao sinal analógico presente no pino selecionado. Esse valor pode variar de 0 a 4095, correspondendo ao range de 0 a 3,3 V (considerando um ADC de 12 bits).

O modo One-Shot é particularmente útil em situações em que a taxa de amostragem não precisa ser alta e quando há necessidade de controlar precisamente o momento da leitura, como na leitura de sensores que mudam lentamente.

4.2.2 Modo Free-Running

O modo Free-Running é mais automatizado e permite que o ADC realize conversões contínuas de forma automática, sem a necessidade de iniciar cada conversão manualmente. Ele é ideal para aplicações que requerem amostragens rápidas e contínuas, como na captura de sinais analógicos variáveis em tempo real. Neste modo, o ADC funciona da seguinte maneira:

Função	Uso
Inicia novas conversões de forma automática	Assim que o módulo é configurado, ele começa a realizar conversões continuamente, sem precisar de comandos adicionais do software.
Amostragem sequencial das entradas analógicas	O ADC pode ser configurado para amostrar várias entradas analógicas de forma sequencial. Assim, ele alterna automaticamente entre os canais pré-selecionados, armazenando os valores convertidos.

Armazenamento em FIFO	Os valores convertidos são armazenados em uma FIFO (First In, First Out) interna de até oito entradas. Isso permite armazenar múltiplas amostras antes de transferi-las para a memória principal ou processá-las.
Integração com o DMA	Para facilitar o gerenciamento dos dados amostrados, o modo Free-Running pode ser combinado com o DMA (Direct Memory Access). O DMA pode transferir automaticamente os dados da FIFO do ADC para a memória, sem a necessidade de intervenção direta da CPU, garantindo uma captura de dados eficiente e de alta velocidade.

O modo Free-Running é ideal para aplicações onde há necessidade de monitoramento contínuo e de alta taxa de amostragem, como na aquisição de dados de sensores que mudam rapidamente. A integração com o DMA permite reduzir a carga de trabalho da CPU, tornando o processo de coleta de dados mais eficiente.

4.3 Configurar e implementar o uso do módulo A/D

Agora que você entende o funcionamento do ADC, vamos aprender como configurá-lo.

NA PRÁTICA

Passo 1: Inicialização do Módulo ADC

Antes de realizar qualquer leitura analógica, é necessário inicializar o módulo ADC do microcontrolador. No caso do RP2040, utilizamos a função adc_init(). Esse comando configura o módulo para começar a operar, ajustando os registradores internos e preparando o hardware para receber os comandos de conversão.

adc_init(); // Inicializa o módulo ADC do microcontrolador.

Passo 2: Configuração dos Pinos GPIO para Entrada Analógica

O próximo passo é configurar os pinos GPIO que serão utilizados como entradas analógicas. No RP2040, os pinos analógicos correspondem aos canais ADC, que variam de 0 a 3 para os pinos físicos GP26 a GP29. Para configurar um pino como entrada analógica, utilizamos a função adc_gpio_init(gpio).

adc_gpio_init(26); // Configura o pino GP26 como entrada analógica (ADC0).
adc_gpio_init(27); // Configura o pino GP27 como entrada analógica (ADC1).

Passo 3: Seleção do Canal ADC

Após configurar os pinos GPIO, é necessário selecionar qual canal ADC será utilizado para a próxima leitura. Isso é feito com a função adc_select_input(channel), onde channel representa o canal de entrada que queremos ler (0, 1, 2, etc.).

adc_select_input(0); // Seleciona o canal 0 (ADC0) para leitura.

Passo 4: Realização da Conversão Analógica-Digital.

Com o canal selecionado, podemos iniciar a conversão e obter o valor digital correspondente ao sinal analógico presente no pino selecionado. Para isso, usamos a função adc_read(), que retorna um valor de 0 a 4095 (12 bits), correspondente ao range de tensão de 0 a 3,3 V.

uint16_t valor_adc = adc_read(); // Lê o valor do ADC no canal selecionado.

Passo 5: Conversão dos Valores para Grandezas Físicas

O valor digital lido do ADC pode ser convertido em uma grandeza física, como tensão, temperatura ou posição, dependendo do tipo de sensor conectado ao pino analógico. Para isso, utilizamos uma fórmula de conversão baseada no range de entrada do sensor e na resolução do ADC.

```
float tensao = valor_adc * (3.3 / 4095); // Converte o valor ADC para tensão em volts.
```

4.4 Exemplos práticos

Exemplo 00: Esse programa realiza uma leitura contínua da temperatura interna do microcontrolador RP2040 e imprime o valor convertido em graus Celsius no terminal, uma vez por segundo. É um exemplo simples e eficaz de como utilizar o ADC para capturar dados analógicos e interpretá-los em um sistema embarcado.

Clique aqui para ver exemplo 00 (pág 15)

Exemplo 01: Este código realiza a leitura contínua de um sinal analógico no canal 0 do conversor analógico-digital (ADC) do Raspberry Pi Pico, utilizando o pino GPIO 26. O valor lido é impresso na comunicação serial para monitoramento e depuração em tempo real.

Clique aqui para ver exemplo 01 (pág 17)

Exemplo 02: Este código realiza a leitura dos valores analógicos de um joystick conectado aos pinos GP26 e GP27 de um Raspberry Pi Pico, além de ler o estado de um botão conectado ao pino GP22. O código utiliza o ADC (Conversor Analógico-Digital) do microcontrolador para converter os sinais analógicos do joystick em valores digitais, que são então exibidos na interface serial.

Clique aqui para ver exemplo 02 (pág 19)

Exemplo 03: Este código realiza leitura dos valores de um joystick e de um botão, e utiliza esses valores para controlar três LEDs. Ele utiliza o ADC (Conversor Analógico-Digital) para ler as entradas analógicas do joystick nos eixos X e Y, e controla os LEDs com base nesses valores e no estado do botão.

Clique aqui para ver exemplo 03 (pág 22)

Exemplo 04: Este código permite controlar o brilho de um LED usando o eixo X do joystick, com o valor do PWM variando conforme o movimento do joystick. Esse código deve ser utilizado na placa BitDogLab.

Clique aqui para ver exemplo 04 (pág 25)

Exemplo 05: Este código permite controlar o brilho de dois LEDs usando o eixo X e o eixo Y do joystick, com o valor do PWM variando conforme o movimento do joystick. Esse código deve ser utilizado na placa BitDogLab.

Clique aqui para ver exemplo 05 (pág 28)

Exemplo 06: Este código permite controlar a luminosidade dos LEDs, através da função pwm, com base na posição do joystick e no estado do botão, fornecendo uma maneira simples de visualizar e interagir com valores analógicos usando o Raspberry Pi Pico. Esse código deve ser utilizado na placa BitDogLab.

Clique aqui para ver exemplo 06 (pág 32)

4.5 CMakeLists

Para utilizar os programas no VS Code, lembre-se de adicionar a biblioteca padrão do adc, como segue abaixo:

```
# Adiciona as bibliotecas necessárias ao executável
# Inclui a biblioteca padrão do Pico e ADC
target_link_libraries(nome_do_seu_programa
    pico_stdlib          # Biblioteca padrão do Pico
    hardware_adc          # Suporte a hardware ADC (importante
para leitura de sinais analógicos)
)
```

Figura 1: Adicionando à biblioteca padrão do ADC
Fonte: Imagem do autor.

EM SINTESE

Revisando, aprendemos que o ADC (Conversor Analógico-Digital) é uma ferramenta essencial para converter sinais analógicos em valores digitais, permitindo que microcontroladores interpretem dados vindos do mundo físico. No contexto do RP2040 da Raspberry Pi Pico W, exploramos as principais configurações e aplicações do módulo ADC. Através de exemplos práticos, vimos como configurar e utilizar o ADC para capturar sinais analógicos de forma precisa e eficiente. Compreender o funcionamento do ADC, suas características e modos de operação é fundamental para desenvolver projetos robustos e dinâmicos. Isso possibilita a criação de sistemas mais interativos e precisos, que utilizam esses dados para controlar outras funcionalidades, como a modulação de largura de pulso (PWM) e atuadores, tornando o código mais eficiente e o projeto mais escalável.

Não se esqueça de revisar o material e testar os exemplos práticos apresentados neste e-book. Experimente configurar o ADC em seus próprios projetos e explore as diferentes combinações de dispositivos. Para consolidar o conhecimento adquirido, acesse a plataforma de aprendizagem e complete a atividade prática disponível. Mãos à obra!

5. Conclusão

O ADC desempenha um papel fundamental nos microcontroladores, pois permite a conversão de sinais analógicos para o domínio digital, possibilitando que o sistema intérprete dados de sensores e controle funcionalidades como o PWM. Ele é essencial em aplicações que envolvem a leitura de variáveis físicas, como temperatura, luz e posição, fornecidas por sensores analógicos. No RP2040, o módulo ADC pode ser configurado de maneira simples e eficiente, integrando-se facilmente a outros periféricos, como o PWM, para criar sistemas de controle mais sofisticados e dinâmicos. Compreender o funcionamento e as diferentes configurações do ADC é crucial para otimizar o desempenho e a precisão das leituras, permitindo o desenvolvimento de projetos embarcados que interagem de forma precisa e eficaz com o ambiente. Sua participação

e dedicação são essenciais para o sucesso em projetos de sistemas embarcados. Continue explorando, praticando e se desafiando a criar soluções inovadoras. Até a próxima!

Referências

[1] BITDOGLAB. BitDogLab. Disponível em: https://github.com/BitDogLab/BitDogLab. Acesso em: 26 set. 2024.

[2] RASPBERRY PI FOUNDATION. RP2040 Datasheet. Disponível em: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf. Acesso em: 25 set. 2024.

[3] GAY, Warren. Debouncing. In: GAY, Warren. Exploring the Raspberry Pi 2 with C++. 1. ed. New York: Apress, 2015. p. 105-112.

[4] CIRCUIT BASICS. How to set up a keypad on an Arduino. Disponível em: https://www.circuitbasics.com/how-to-set-up-a-keypad-on-an-arduino/. Acesso em: 26 set. 2024.

Exemplo 0

Link do programa comentado:
https://wokwi.com/projects/410323061251060737

A seguir, vamos analisar detalhadamente cada parte do código:

1. Inclusão de Bibliotecas

- #include <stdio.h>: Esta biblioteca é usada para as funções de entrada e saída, como printf, que permite enviar mensagens para o terminal.
- #include "pico/stdlib.h": Contém funções básicas da plataforma Raspberry Pi Pico, como inicialização de GPIO e funções de temporização.
- #include "hardware/adc.h": Fornece funções para controlar o módulo ADC, necessário para converter sinais analógicos em digitais.

2. Definição de Constantes:

- #define ADC_TEMPERATURE_CHANNEL 4: Define o canal do ADC utilizado para a leitura do sensor de temperatura interno. No RP2040, o canal 4 é mapeado para esse sensor.

3. Função adc_to_temperature:

- Converte o valor lido do ADC, que está em formato de 12 bits (0-4095), para uma tensão entre 0 e 3,3 V.
- Em seguida, a função utiliza uma fórmula específica fornecida no datasheet do RP2040 para converter essa tensão em temperatura em graus Celsius.

4. Função main:

- **Inicialização da comunicação serial:**
  - » stdio_init_all(): Prepara a comunicação serial, permitindo o uso da função printf para enviar dados para o console.
- **Inicialização do módulo ADC:**
  - » adc_init(): Prepara o ADC para começar a realizar leituras dos canais

configurados.

- **Habilitação do sensor de temperatura:**
  - » adc_set_temp_sensor_enabled(true): Ativa o sensor de temperatura interno do RP2040.
- **Seleção do canal ADC:**
  - » adc_select_input(ADC_TEMPERATURE_CHANNEL): Configura o ADC para ler o canal 4, onde está conectado o sensor de temperatura.
- **Loop Infinito:**
  - » Realiza continuamente as seguintes ações:
    - **Leitura do ADC:** uint16_t adc_value = adc_read(); Lê o valor atual do canal ADC selecionado.
    - **Conversão para temperatura:** float temperature = adc_to_temperature(adc_value); Converte o valor lido para temperatura em graus Celsius.
    - **Impressão da temperatura:** printf("Temperatura: %.2f °C\n", temperature); Envia a temperatura lida para o console serial.
    - **Atraso de 1 segundo:** sleep_ms(1000); Espera 1 segundo antes de realizar a próxima leitura.

Exemplo 1

Link do programa comentado:
https://wokwi.com/projects/410100384229587969

A seguir, vamos analisar detalhadamente cada parte do código:

1. Inicialização da Comunicação Serial:

- stdio_init_all();: Prepara a comunicação serial via USB, permitindo que possamos usar printf() para enviar mensagens e valores ao console para depuração.

2. Inicialização do ADC:

- adc_init();: Configura o hardware do ADC, preparando-o para realizar leituras analógicas.

3. Configuração do Pino para Leitura Analógica:

- adc_gpio_init(26);: Define o pino GPIO 26 como entrada analógica, associando-o ao canal 0 do ADC.

4. Seleção do Canal de Leitura:

- adc_select_input(0);: Seleciona o canal 0 do ADC para realizar as leituras analógicas a partir do pino 26.

5. Loop Infinito de Leitura:

- O código entra em um loop infinito (while(true){}) para ler continuamente o valor analógico no pino 26.
- adc_read();: Lê o valor do ADC no canal selecionado, retornando um valor de 0 a 4095, que representa a tensão analógica presente no pino 26.
- printf("Valor do ADC: %u\n", adc_value);: Imprime o valor lido na interface serial para que possamos monitorá-lo em tempo real.

6. Atraso Entre Leituras:

- sleep_ms(500);: Introduz um atraso de 500 milissegundos para evitar que as leituras e as mensagens na serial ocorram de forma muito rápida, tornando a visualização mais clara e evitando sobrecarregar a interface.

Exemplo 2

Link do programa comentado:
https://wokwi.com/projects/410311174151494657

A seguir, vamos analisar detalhadamente cada parte do código:

Inclusão das Bibliotecas:

- #include <stdio.h>: Permite o uso da função printf() para enviar dados ao console.
- #include "pico/stdlib.h": Contém funções básicas do Raspberry Pi Pico, como controle de GPIO e temporização.
- #include "hardware/adc.h": Permite o uso do conversor analógico-digital (ADC) para ler valores analógicos.

Definições de Pinos:

- VRX_PIN e VRY_PIN: Representam os pinos GP26 e GP27, respectivamente, que estão conectados aos eixos X e Y do joystick. Esses pinos são configurados para entrada analógica.
- SW_PIN: Representa o pino GP22, que é utilizado para o botão do joystick. Ele está configurado como entrada digital.

Inicialização da Comunicação Serial:

- stdio_init_all();: Habilita a comunicação serial para enviar dados ao console via USB, usando o printf().

Inicialização do ADC:

- adc_init();: Configura o hardware do ADC para ser utilizado, permitindo leituras analógicas.

Configuração dos Pinos ADC:

- adc_gpio_init(VRX_PIN);: Configura o pino GP26 como entrada analógica para leitura do eixo X do joystick.

- adc_gpio_init(VRY_PIN);: Configura o pino GP27 como entrada analógica para leitura do eixo Y do joystick.

Configuração do Botão:

- gpio_init(SW_PIN);: Inicializa o pino GP22 como entrada digital.
- gpio_set_dir(SW_PIN, GPIO_IN);: Define a direção do pino GP22 como entrada.
- gpio_pull_up(SW_PIN);: Ativa o resistor de pull-up interno para garantir que o valor do pino seja "alto" quando o botão não estiver pressionado, evitando leituras instáveis.

Loop Infinito:

- O loop while (true) {} é utilizado para ler continuamente os valores dos eixos do joystick e o estado do botão.

Leitura dos Valores do Joystick:

- adc_select_input(0);: Seleciona o canal 0 do ADC para leitura do pino GP26 (eixo X do joystick).
- uint16_t vrx_value = adc_read();: Lê o valor do eixo X (0 a 4095).
- adc_select_input(1);: Seleciona o canal 1 do ADC para leitura do pino GP27 (eixo Y do joystick).
- uint16_t vry_value = adc_read();: Lê o valor do eixo Y (0 a 4095).

Leitura do Estado do Botão:

- bool sw_value = gpio_get(SW_PIN) == 0;: Lê o estado do botão. Se o valor for 0, o botão está pressionado; caso contrário, não está pressionado.

Impressão dos Valores Lidos:

- printf("VRX: %u, VRY: %u, SW: %d\n", vrx_value, vry_value, sw_value);: Exibe os valores dos eixos do joystick e o estado do botão na interface serial.

Atraso:

- sleep_ms(500);: Pausa a execução do código por 500 milissegundos para evitar leituras muito rápidas e sobrecarregar a comunicação serial.

Resumo do Funcionamento:

O código continua em loop, lendo os valores do joystick e do botão e exibindo os resultados na serial a cada **500ms**. Isso é útil para monitoramento contínuo do estado do joystick e do botão em um projeto de sistemas embarcados.

Sugestões de Melhorias:

- Adicionar **PWM** para controlar a intensidade de LEDs com base nos valores lidos do joystick.
- Implementar **debounce** para o botão, evitando leituras falsas quando pressionado rapidamente.
- Expandir o código para controlar outros dispositivos com base nos valores lidos do joystick.

Exemplo 3

Link do programa comentado:
https://wokwi.com/projects/410311269603373057

A seguir, vamos analisar detalhadamente cada parte do código:

Explicação Detalhada do Código:

Inclusão das Bibliotecas:

- #include <stdio.h>: Permite o uso da função printf() para enviar dados ao console.
- #include "pico/stdlib.h": Contém funções básicas do Raspberry Pi Pico, como controle de GPIO e temporização.
- #include "hardware/adc.h": Permite o uso do conversor analógico-digital (ADC) para ler valores analógicos.

Definições de Pinos:

- VRX_PIN, VRY_PIN: Correspondem aos pinos GP26 e GP27, conectados aos eixos X e Y do joystick. Estes pinos são configurados para entrada analógica (ADC0 e ADC1).
- SW_PIN: Representa o pino GP22, que é utilizado para o botão do joystick. Ele está configurado como entrada digital com pull-up interno.
- LED1_PIN, LED2_PIN, LED3_PIN: Correspondem aos pinos GP11, GP13 e GP12 conectados aos LEDs, que serão controlados com base nos valores dos ADCs e do botão.

Inicialização da Comunicação Serial:

- stdio_init_all();: Habilita a comunicação serial para enviar dados ao console via USB, usando o printf().

Inicialização do ADC:

- adc_init();: Configura o hardware do ADC para ser utilizado, permitindo leituras analógicas.

Configuração dos Pinos ADC:

- adc_gpio_init(VRX_PIN);: Configura o pino GP26 como entrada analógica para leitura do eixo X do joystick.
- adc_gpio_init(VRY_PIN);: Configura o pino GP27 como entrada analógica para leitura do eixo Y do joystick.

Configuração do Botão:

- gpio_init(SW_PIN);: Inicializa o pino GP22 como entrada digital.
- gpio_set_dir(SW_PIN, GPIO_IN);: Define a direção do pino GP22 como entrada.
- gpio_pull_up(SW_PIN);: Ativa o resistor de pull-up interno para garantir que o valor do pino seja "alto" quando o botão não estiver pressionado, evitando leituras instáveis.

Configuração dos LEDs:

- Os LEDs LED1_PIN, LED2_PIN e LED3_PIN são configurados como saídas, e inicialmente apagados utilizando gpio_put() com o valor false.

Loop Infinito:

-
- O loop while (true) {} é utilizado para ler continuamente os valores dos eixos do joystick e o estado do botão.

Leitura dos Valores do Joystick:

- adc_select_input(0);: Seleciona o canal 0 do ADC para leitura do pino GP26 (eixo X do joystick).
- uint16_t vrx_value = adc_read();: Lê o valor do eixo X (0 a 4095).
- adc_select_input(1);: Seleciona o canal 1 do ADC para leitura do pino GP27 (eixo Y do joystick).
- uint16_t vry_value = adc_read();: Lê o valor do eixo Y (0 a 4095).

Leitura do Estado do Botão:

- bool sw_value = gpio_get(SW_PIN) == 0;: Lê o estado do botão. Se o valor

for 0, o botão está pressionado; caso contrário, não está pressionado.

Controle dos LEDs:

- LED1_PIN é controlado pelo valor do eixo X do joystick: Acende se vrx_value > 2100.
- LED2_PIN é controlado pelo valor do eixo Y do joystick: Acende se vry_value > 2100.
- LED3_PIN é controlado pelo estado do botão: Acende se o botão estiver pressionado (sw_value == true).

Impressão dos Valores Lidos:

- printf("VRX: %u, VRY: %u, SW: %d\n", vrx_value, vry_value, sw_value);: Exibe os valores dos eixos do joystick e o estado do botão na interface serial.

Atraso:

- sleep_ms(500);: Pausa a execução do código por 500 milissegundos para evitar leituras muito rápidas e sobrecarregar a comunicação serial.

Resumo do Funcionamento:

- O joystick move-se nos eixos X e Y, e seus valores analógicos são lidos pelo ADC do RP2040.
- Dependendo da posição do joystick (eixo X ou Y) e do estado do botão, diferentes LEDs serão ligados ou desligados.
- As leituras são exibidas no console serial para monitoramento.

Possíveis Melhorias:

- Adicionar **PWM** ao controle dos LEDs para ajustar o brilho com base nos valores do joystick.
- Implementar **debounce** no botão para evitar leituras flutuantes ao pressioná-lo rapidamente.
- Expandir o controle para mais dispositivos ou ajustar o código para outras interações.

Exemplo 4

Link do programa comentado:
https://wokwi.com/projects/410368555907733505

A seguir, vamos analisar detalhadamente cada parte do código:

Inclusão das bibliotecas:

- #include <stdio.h>: Fornece funções padrão como printf() para enviar dados ao console.
- #include "pico/stdlib.h": Contém funções básicas de controle de GPIO, temporização e outras operações essenciais no Raspberry Pi Pico.
- #include "hardware/adc.h": Permite a leitura de valores analógicos usando o ADC (Conversor Analógico-Digital) do Pico.
- #include "hardware/pwm.h": Permite o controle de sinais PWM (Pulse Width Modulation) para o controle de dispositivos como LEDs.

Definição de pinos:

- VRX_PIN: Representa o pino GP26, utilizado para ler o valor do eixo X do joystick através do ADC.
- LED_PIN: Representa o pino GP12, onde será conectado o LED controlado via **PWM**.

Função pwm_init_gpio:

- Esta função inicializa o PWM no pino definido, configurando o pino para ser usado como saída PWM e definindo o valor máximo (wrap) do contador do PWM.
- Ela retorna o número do "slice" do PWM, que será usado posteriormente para controlar o PWM.

Função main:

- Inicializa a comunicação serial para enviar dados ao console e permite monitorar as informações durante a execução do programa.
- O ADC é configurado para leitura de valores analógicos no pino VRX_PIN.

- O PWM é configurado para o pino LED_PIN com uma resolução de 4096 (valor máximo do contador do PWM).
- O loop principal do programa lê o valor do joystick (via ADC) e ajusta o duty cycle do LED de acordo com o valor lido. O duty cycle varia proporcionalmente ao valor do joystick (0 a 4095).
- A cada 1 segundo, o valor lido e o duty cycle calculado são impressos no console.

Controle do PWM:

- O valor lido do joystick (vrx_value) é usado para ajustar o nível de PWM no pino LED_PIN, que controla o brilho do LED.
- O valor do ADC lido varia de 0 a 4095, e esse valor é mapeado diretamente para o duty cycle do PWM.

Impressão na serial:

- A cada 1 segundo, o valor lido do joystick e o duty cycle calculado são impressos no console, permitindo acompanhar a variação do PWM conforme o movimento do joystick.

Atraso de 100ms:

- O sleep_ms(100) introduz um pequeno atraso no loop principal, para evitar que o código rode muito rápido e sobrecarregue o sistema com leituras e impressões em excesso.

Resumo do Funcionamento:

- **Joystick e ADC:** O programa lê continuamente o valor analógico do eixo X de um joystick, utilizando o ADC do Raspberry Pi Pico.
- **Controle do LED com PWM:** O valor lido do joystick é utilizado para ajustar o duty cycle do PWM, controlando o brilho do LED conectado ao pino GP12.
- **Monitoramento via Serial:** O valor lido e o duty cycle correspondente são impressos no console serial a cada 1 segundo.
- **Ajuste do PWM:** O LED brilha mais intensamente à medida que o joystick é movido para uma posição de valor mais alto, e diminui o brilho para valores mais baixos.

Aplicações:

- **Controle de Brilho:** Pode ser usado para controlar o brilho de um LED ou outro dispositivo através de um joystick ou outro sensor analógico.
- **Interfaces de Controle:** Adapte o código para usar diferentes entradas analógicas, como potenciômetros, para controlar o PWM de dispositivos de saída.
- **Sistemas de Iluminação Inteligente:** Esse código pode ser a base para um sistema de controle de iluminação inteligente, onde o brilho das luzes é ajustado automaticamente com base em entradas analógicas.

Possíveis Melhorias:

- **Adicionar mais Eixos:** Expanda o código para incluir o eixo Y do joystick e controlar um segundo LED.
- **Botões de Controle:** Adicionar botões para ligar/desligar o controle do LED ou ajustar manualmente o duty cycle.
- **Feedback Visual:** Adicionar uma forma de feedback visual, como uma barra de LEDs, para mostrar o duty cycle em tempo real.
- **Integração com Outros Dispositivos:** Utilizar outros dispositivos, como displays LCD ou sensores adicionais, para melhorar a interface.

Exemplo 5

Link do programa comentado:
https://wokwi.com/projects/410311612618810369

A seguir, vamos analisar detalhadamente cada parte do código:

Inclusão das Bibliotecas:

- #include <stdio.h>: Permite o uso da função printf() para enviar dados ao console, facilitando a depuração e a visualização dos valores lidos e processados pelo programa.
- #include "pico/stdlib.h": Fornece funções básicas específicas do Raspberry Pi Pico, como controle de GPIO, temporização e comunicação serial, essenciais para o funcionamento do microcontrolador.
- #include "hardware/adc.h": Habilita o uso do conversor analógico-digital (ADC) para ler valores analógicos dos pinos configurados, como o joystick.

Definições de Pinos:

- **VRX_PIN, VRY_PIN:** Correspondem aos pinos GP26 e GP27, conectados aos eixos X e Y do joystick. Esses pinos são configurados como entradas analógicas nos canais ADC0 e ADC1 respectivamente.
- **SW_PIN:** Corresponde ao pino GP22, utilizado para o botão do joystick. Esse pino está configurado como entrada digital com um resistor de pull-up interno, garantindo leituras estáveis.
- **LED1_PIN, LED2_PIN, LED3_PIN:** Correspondem aos pinos GP11, GP13 e GP12, conectados aos LEDs que serão controlados com base nos valores dos ADCs e no estado do botão.

Inicialização da Comunicação Serial:

- **stdio_init_all();:** Inicializa a comunicação serial via USB, permitindo que o programa envie mensagens para o console usando a função printf(). Isso é útil para depuração e monitoramento do comportamento do código.

Inicialização do ADC:

- **adc_init();:** Configura e habilita o módulo ADC do microcontrolador, permitindo leituras de sinais analógicos dos pinos GPIO configurados. Isso prepara o hardware para iniciar a conversão de sinais analógicos para digitais.

Configuração dos Pinos ADC:

- **adc_gpio_init(VRX_PIN);:** Configura o pino GP26 como entrada analógica para leitura do eixo X do joystick.
- **adc_gpio_init(VRY_PIN);:** Configura o pino GP27 como entrada analógica para leitura do eixo Y do joystick.

Configuração do Botão:

- **gpio_init(SW_PIN);:** Inicializa o pino GP22 como entrada digital.
- **gpio_set_dir(SW_PIN, GPIO_IN);:** Define o pino GP22 como entrada, configurando-o para ler o estado do botão.
- **gpio_pull_up(SW_PIN);:** Ativa um resistor de pull-up interno no pino GP22, garantindo que a leitura do botão seja "alta" quando não está pressionado. Isso previne leituras incorretas e interferências.

Configuração dos LEDs:

- **Configuração inicial dos LEDs:**
  - » Cada LED (LED1, LED2, LED3) é configurado como saída, e o comando gpio_put() com valor false apaga todos os LEDs inicialmente.

Loop Infinito:

- O loop while (true) {} é usado para ler continuamente os valores dos eixos do joystick e o estado do botão, controlando os LEDs com base nesses valores.

Leitura dos Valores do Joystick:

- **adc_select_input(0);:** Seleciona o canal 0 do ADC, correspondente ao pino GP26 (eixo X do joystick).

- **uint16_t vrx_value = adc_read();:** Lê o valor analógico do eixo X, que varia de 0 a 4095.
- **adc_select_input(1);:** Seleciona o canal 1 do ADC, correspondente ao pino GP27 (eixo Y do joystick).
- **uint16_t vry_value = adc_read();:** Lê o valor analógico do eixo Y, que varia de 0 a 4095.

Leitura do Estado do Botão:

- **bool sw_value = gpio_get(SW_PIN) == 0;:** Lê o estado do botão. Se o valor for 0, significa que o botão está pressionado; caso contrário, ele não está pressionado.

Controle dos LEDs:

- **Controle do LED1:**
  - » Se o valor do eixo X (vrx_value) for maior que 2100, o LED1 é ligado (gpio_put(LED1_PIN, true);), caso contrário, ele é desligado.
- **Controle do LED2:**
  - » Se o valor do eixo Y (vry_value) for maior que 2100, o LED2 é ligado, caso contrário, ele é desligado.
- **Controle do LED3:**
  - » O LED3 é controlado pelo estado do botão (SW_PIN). Se o botão estiver pressionado, o LED3 é ligado, caso contrário, é desligado.

Impressão dos Valores Lidos:

- **printf("VRX: %u, VRY: %u, SW: %d\n", vrx_value, vry_value, sw_value);:** Exibe os valores lidos dos eixos do joystick e o estado do botão na interface serial, facilitando a visualização e depuração.

Atraso:

- **sleep_ms(500);:** Pausa a execução do código por 500 milissegundos para evitar que as leituras e a impressão dos valores sejam feitas rapidamente, sobrecarregando a comunicação serial e facilitando a visualização dos dados.

Resumo do Funcionamento:

- **Joystick:** Lê continuamente os valores dos eixos X e Y, variando de 0 a 4095.
- **Botões A e B:** Controlam a ativação dos LEDs azul e verde, respectivamente.
  - » **Botão A:** Quando pressionado, permite que o LED azul seja controlado pelo eixo X.
  - » **Botão B:** Quando pressionado, permite que o LED verde seja controlado pelo eixo Y.
- **PWM nos LEDs:** O brilho dos LEDs é ajustado via PWM, onde o duty cycle é proporcional ao valor lido do joystick.
- **Debounce:** Implementado nos botões A e B para evitar leituras falsas.
- **Impressão na Serial:** Exibe informações úteis para monitoramento e depuração a cada 1 segundo.

Possíveis Aplicações e Extensões:

- **Controle de Brilho:** Este código pode ser utilizado para controlar o brilho de LEDs em um projeto de iluminação ou indicadores visuais.
- **Interface Homem-Máquina:** O joystick e os botões podem ser usados para navegar em menus ou ajustar parâmetros em um dispositivo.
- **Jogos e Simulações:** Poderia ser adaptado para controlar elementos em um jogo simples ou simulação, utilizando o joystick como entrada.

Desafios e Melhorias:

- **Adicionar Funções ao Botão do Joystick (SW_PIN):** Implementar funcionalidades adicionais, como um modo de calibração ou reset.
- **Feedback Sonoro:** Integrar um buzzer que emite sons conforme o movimento do joystick ou pressionamento dos botões.
- **Display LCD:** Adicionar um display para mostrar as leituras em tempo real ou informações adicionais.
- **Melhorar o Debounce:** Refinar o algoritmo de debounce para lidar com múltiplos botões simultaneamente.

Exemplo 6

Link do programa comentado:
https://wokwi.com/projects/410414274413048833

A seguir, vamos analisar detalhadamente cada parte do código:

Inclusão de Bibliotecas:

- **stdio.h:** Biblioteca padrão de entrada e saída para uso de funções como printf().
- **hardware/adc.h:** Biblioteca usada para configurar e controlar o módulo ADC (Conversor Analógico-Digital).
- **hardware/pwm.h:** Biblioteca usada para configurar e controlar o PWM (Pulse Width Modulation).
- **pico/stdlib.h:** Biblioteca padrão do Raspberry Pi Pico para manipulação de GPIO e temporização.

Definição de Pinos e Variáveis:

- **VRX, VRY, SW:** Pinos do joystick (VRX e VRY) para leitura dos eixos X e Y. O pino **SW** é para leitura do botão do joystick.
- **LED_B, LED_R:** Pinos para controle dos LEDs azul e vermelho, respectivamente, via PWM.
- **DIVIDER_PWM:** Define o divisor de clock para o PWM, controlando a frequência.
- **PERIOD:** Define o período do PWM (número máximo de contagem), sendo 4096.
- **led_b_level, led_r_level:** Variáveis para armazenar o nível de PWM (duty cycle) dos LEDs.
- **slice_led_b, slice_led_r:** Variáveis que armazenam os "slices" de PWM correspondentes aos LEDs. Cada slice é uma unidade de controle de PWM.

Função setup_joystick():

Esta função inicializa o ADC e o pino do botão do joystick.

- **adc_init():** Inicializa o módulo ADC.

- **adc_gpio_init():** Configura os pinos para leitura analógica.
- **gpio_init() e gpio_set_dir():** Inicializa o pino do botão como entrada.
- **gpio_pull_up():** Ativa o pull-up interno do botão para evitar leituras instáveis.

Função setup_pwm_led():

Esta função configura o PWM para os LEDs.

- **gpio_set_function():** Configura o pino do LED para funcionar como saída PWM.
- **pwm_gpio_to_slice_num():** Obtém o número do slice PWM associado ao pino.
- **pwm_set_clkdiv():** Define o divisor do clock do PWM.
- **pwm_set_wrap():** Define o valor máximo do contador do PWM, que controla o período.
- **pwm_set_gpio_level():** Define o nível inicial do PWM (duty cycle) para o LED.
- **pwm_set_enabled():** Habilita o PWM no slice correspondente.

Função setup():

Inicializa as funções essenciais do sistema.

- **stdio_init_all():** Inicializa a porta serial para permitir a comunicação.
- Chama as funções setup_joystick() e setup_pwm_led() para configurar o joystick e os LEDs via PWM.

Função joystick_read_axis():

Lê os valores analógicos dos eixos X e Y do joystick.

- **adc_select_input():** Seleciona o canal do ADC para ler a entrada analógica.
- **adc_read():** Lê o valor analógico convertido para digital (0-4095).

Função Principal main():

O loop principal do programa:

- **joystick_read_axis( ):** Lê os valores dos eixos X e Y.
- **pwm_set_gpio_level( ):** Ajusta os níveis PWM dos LEDs de acordo com os valores lidos dos eixos do joystick.
- **sleep_ms(100):** Introduz uma pausa de 100ms antes de repetir o ciclo.

Explicação do Funcionamento:

- O joystick é usado para controlar os níveis de brilho dos LEDs azul e vermelho via PWM. O eixo X controla o brilho do LED azul e o eixo Y controla o LED vermelho.
- O código está continuamente lendo os valores dos eixos do joystick e ajustando o brilho dos LEDs de forma proporcional.
- O botão **SW** do joystick não é usado neste exemplo, mas está configurado e disponível para futuras expansões do projeto.

Atividade Proposta:

- **Modifique o código** para que o botão **SW** controle o estado dos LEDs (liga/desliga) enquanto o joystick ajusta o brilho.
- **Adicione um terceiro LED** e utilize o eixo Y para controlar seu brilho.


embarca
tech
Residência
Tecnológica
em Sistemas
Embarcados