embarca tech
Residência Tecnológica em Sistemas Embarcados

Guia de Interfaces de Comunicação com RP2040 / Raspberry Pi Pico W

Unidade 4 | Capítulo 6

Jorge Wattes

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 3
2. OBJETIVOS EDUCACIONAIS 3
3. Entendendo as Interfaces de Comunicação Serial 3
3.1. Por Que Usar Comunicação Serial? 4
4. UART - Universal Asynchronous Receiver/Transmitter 4
4.1. Funcionamento da UART 4
4.2. Configuração Básica da UART no RP2040 5
4.3. Aplicações Comuns da UART 8
5. SPI - Serial Peripheral Interface 8
5.1. Como Funciona a SPI? 8
5.2. Implementação Básica da SPI no RP2040 9
5.3. Aplicações Comuns da SPI 11
6.1. Funcionamento da I2C 12
6.2. Implementação Básica da I2C no RP2040 13
6.3. Aplicações Comuns da I2C 15
8. Aplicações Práticas e Projetos 17
8.1. Projeto Integrado: Monitoramento Ambiental 17
CONCLUSÃO 20
REFERÊNCIAS 20

Unidade 4

Capítulo 6

1. BOAS-VINDAS E INTRODUÇÃO

Olá! Seja bem-vindo ao Guia Completo de Interfaces de Comunicação com Raspberry Pi Pico W. Este e-book foi desenvolvido para fornecer um entendimento aprofundado e prático das principais interfaces de comunicação serial utilizadas no microcontrolador RP2040, presente na placa Raspberry Pi Pico W. Exploraremos os protocolos UART, SPI e I2C, abordando suas características, exemplos práticos, e como utilizá-los em diferentes projetos de sistemas embarcados.
A capacidade de se comunicar com diversos dispositivos é essencial em projetos de eletrônica e automação, permitindo a integração de sensores, atuadores e módulos de comunicação. Neste guia, você encontrará tudo o que precisa para implementar essas interfaces com eficiência e simplicidade.

2. OBJETIVOS EDUCACIONAIS

Após a leitura deste Ebook você será capaz de:

- Compreender os fundamentos das interfaces de comunicação serial.
- Implementar e configurar UART, SPI e I2C no RP2040.
- Aplicar essas interfaces em projetos reais com exemplos de código e esquemas.
- Identificar as melhores práticas e armadilhas comuns durante a implementação.

3. Entendendo as Interfaces de Comunicação Serial

As interfaces de comunicação serial são utilizadas para transmitir dados entre dispositivos de forma eficiente e com uso mínimo de cabos. Em contraste com a comunicação paralela, onde vários bits são transmitidos simultaneamente, a comunicação serial envia os dados bit a bit, tornando a implementação mais simples e barata.

3.1. Por Que Usar Comunicação Serial?

A comunicação serial é amplamente utilizada em sistemas embarcados por diversos motivos:

- **Eficiência:** Menor quantidade de cabos e conexões.
- **Flexibilidade:** Pode ser usada em longas distâncias e em ambientes com alta interferência.
- **Simplicidade:** Configuração e implementação diretas.

SAIBA MAIS

Dica de Leitura: Para uma visão mais detalhada sobre os fundamentos da comunicação serial, confira o artigo UART Explained da Digilent.

IMPORTANTE

Observação Importante: Certifique-se de sempre adequar o tipo de interface serial ao seu projeto. A escolha correta pode simplificar muito o design do circuito e a programação.

4. UART - Universal Asynchronous Receiver/Transmitter

A UART é uma interface de comunicação assíncrona que permite a troca de dados entre dois dispositivos. Ela é frequentemente utilizada em módulos Bluetooth, GPS e para depuração de código.

4.1. Funcionamento da UART

A UART utiliza apenas dois fios principais: TX (transmissor) e RX (receptor), além do fio de GND (terra). A comunicação é assíncrona, o que significa que não utiliza um sinal de clock compartilhado. Isso simplifica a implementação, mas requer que os dispositivos estejam configurados com a mesma taxa de transmissão, conhecida como baud rate.

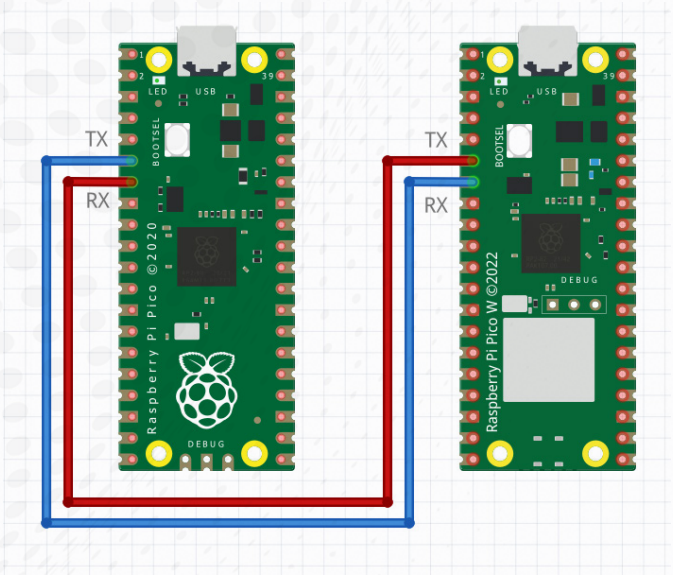

Figura 1. Placas com dispositivos com a mesma taxa de recepção.
Fonte: imagem do autor

A imagem mostra dois microcontroladores Raspberry Pi Pico conectados entre si por meio de fios, representados por linhas coloridas, em um diagrama esquemático. Os microcontroladores estão dispostos lado a lado, com os pinos de conexão virados para fora.
O Raspberry Pi Pico à esquerda tem dois fios conectados aos pinos:
O fio azul sai do pino RX (recepção) e se conecta ao pino TX (transmissão) do Raspberry Pi Pico à direita.
O fio vermelho sai do pino TX (transmissão) e se conecta ao pino RX (recepção) do Raspberry Pi Pico à direita.
Esse tipo de conexão geralmente é usado para comunicação serial entre dois dispositivos, permitindo que eles troquem dados. A ordem de cores e pinos é importante para garantir que a transmissão de um dispositivo chegue à recepção do outro.

IMPORTANTE

Certifique-se de que os dispositivos conectados compartilhem o mesmo baud rate. Erros nessa configuração podem resultar em falhas na comunicação ou perda de dados.

4.2. Configuração Básica da UART no RP2040

Para utilizar a UART no RP2040, precisamos configurar os pinos GPIO para as funções de transmissão (TX) e recepção (RX). Abaixo, um exemplo de código em C/C++:

```c
#include "pico/stdlib.h"
#include "hardware/uart.h"

int main() {
    uart_init(uart0, 115200); // Inicializa a UART0 com baud rate de 115200
    gpio_set_function(0, GPIO_FUNC_UART); // Configura pino 0 como TX
    gpio_set_function(1, GPIO_FUNC_UART); // Configura pino 1 como RX




    while (true) {
        uart_puts(uart0, "Hello abcdefg\n"); // Envia uma mensagem via UART
        sleep_ms(1000); // Pausa de 1 segundo
    }
}
```

Figura 2 – Exemplo de código em C/C++.
Fonte: imagem do autor

FALTA TEXTO PARA CEGO VER.

Já para a recepção de dados via UART no RP2040, podemos configurar o microcontrolador de duas formas, a primeira, envolve utilizar a função uart_is_readable() retorna verdadeiro caso exista um valor a ser lido no periférico UART.

```c
#include "pico/stdlib.h"
#include "hardware/uart.h"

int main() {
    uart_init(uart0, 115200); // Inicializa a UART0 com baud rate de 115200
    gpio_set_function(0, GPIO_FUNC_UART); // Configura pino 0 como TX
    gpio_set_function(1, GPIO_FUNC_UART); // Configura pino 1 como RX

    // Habilita o FIFO para evitar sobrecarga de buffer
    uart_set_fifo_enabled(uart0, true);

    // Configurações adicionais (opcional)
    uart_set_hw_flow(uart0, false, false); // Desabilita controle de fluxo
    uart_set_format(uart0, 8, 1, UART_PARITY_NONE); // Configura formato (8N1)

    while (true) {
        // Verifica se há dados disponíveis na UART
        if (uart_is_readable(uart0)) {
            // Lê um caractere da UART
            char c = uart_getc(uart0);
            // Envia o caractere de volta (echo) para verificação
            uart_putc(uart0, c);
        }
    }
}
```

Figura 3 – Exemplo 2 de código em C/C++.
Fonte: imagem do autor

O código começa incluindo as bibliotecas necessárias: pico/stdlib.h para funções básicas e hardware/uart.h para funções relacionadas à UART.
Em seguida, a função main() é definida, sendo o ponto de partida do programa.
Dentro da função main(), o código:
Inicializa a UART: uart_init(uart0, 115200); - Configura a UART com uma taxa de baud de 115200 bits por segundo.
Configura os pinos:
gpio_set_function(0, GPIO_FUNC_UART); - Configura o pino 0 como saída para transmissão (TX).
gpio_set_function(1, GPIO_FUNC_UART); - Configura o pino 1 como entrada para recepção (RX).
Habilita o FIFO: uart_set_fifo_enabled(uart0, true); - Habilita o buffer FIFO para evitar perda de dados durante a comunicação.
Configurações opcionais:
uart_set_hw_flow(uart0, false, false); - Desabilita o controle de fluxo de hardware.
uart_set_format(uart0, 8, 1, UART_PARITY_NONE); - Configura o formato de dados para 8 bits, 1 bit de parada, sem paridade.
O código entra em um loop infinito (while(true)):
Verifica dados disponíveis: if (uart_is_readable(uart0)) { ... } - Verifica se há dados disponíveis para leitura na UART.
Lê um caractere: char c = uart_getc(uart0); - Lê um caractere da UART.
Envia o caractere de volta (echo): uart_putc(uart0, c); - Envia o caractere lido de volta para o dispositivo de origem.
Este código implementa um eco básico, onde qualquer caractere recebido pela UART é enviado de volta para o dispositivo de origem.
Em resumo, o código demonstra o uso da UART para comunicação serial, configurando os pinos, habilitando o FIFO e recebendo e transmitindo dados.

A código anterior permite realizar a leitura do UART de forma simples, contudo, ele consome o processador no processo de verificação de dados disponíveis. Uma alternativa mais eficiente do ponto de vista de sistemas embarcados é utilizar a interrupção nativa do periférico de UART. No código a seguir faremos isso.

```c
#include "pico/stdlib.h"
#include "hardware/uart.h"
#include "hardware/irq.h"

// Função de callback que será chamada quando a interrupção ocorrer
void on_uart_rx() {
    // Enquanto houver dados para ler na UART
    while (uart_is_readable(uart0)) {
        // Lê o caractere recebido
        char c = uart_getc(uart0);
        // Envia o caractere de volta (echo) para verificação
        uart_putc(uart0, c);
    }
}

int main() {
    // Inicializa a UART0 com baud rate de 115200
    uart_init(uart0, 115200);
    // Configura pino 0 como TX
    gpio_set_function(0, GPIO_FUNC_UART);
    // Configura pino 1 como RX
    gpio_set_function(1, GPIO_FUNC_UART);

    // Habilita o FIFO para evitar sobrecarga de buffer
    uart_set_fifo_enabled(uart0, true);

    // Configura a interrupção para a UART0
    // Define a função de callback para a interrupção de recepção
    irq_set_exclusive_handler(UART0_IRQ, on_uart_rx);
    // Habilita a interrupção na UART0
    irq_set_enabled(UART0_IRQ, true);
    // Habilita a interrupção de recepção de dados (RX) na UART
    uart_set_irq_enables(uart0, true, false);

    while (true) {
    }
}
```

Figura 4 – Exemplo 2 de código em C/C++.
Fonte: imagem do autor

O código apresentado define uma função de callback chamada on_uart_rx(), que será executada quando uma interrupção ocorrer na UART (Universal Asynchronous Receiver/Transmitter).
O código começa incluindo as bibliotecas necessárias:
pico/stdlib.h para funções básicas
hardware/uart.h para funções relacionadas à UART
hardware/irq.h para funções relacionadas a interrupções
A função on_uart_rx() tem a função de lidar com os dados recebidos pela UART.
Dentro da função on_uart_rx():
É iniciado um loop while (uart_is_readable(uart0)) { ... }, que continua a iterar enquanto houver dados disponíveis para serem lidos na UART.
A função uart_is_readable(uart0) verifica se há dados para leitura na UART.
O código dentro do loop while é responsável por ler os dados recebidos e processá-los. A função uart_getc(uart0) lê um caractere da UART e o processa conforme o código dentro do loop while.
Em resumo, este código define uma função de callback que será chamada quando uma interrupção ocorrer na UART. A função lida com os dados recebidos e realiza o processamento necessário.
As linhas de código que incluem bibliotecas e definem a função de callback são importantes para a comunicação serial usando interrupções, permitindo que o microcontrolador responda a eventos de recepção de dados de forma mais eficiente.

Observação: Utilize a UART para depuração durante o desenvolvimento de firmware. Isso permite identificar erros no código sem a necessidade de um debugger complexo.

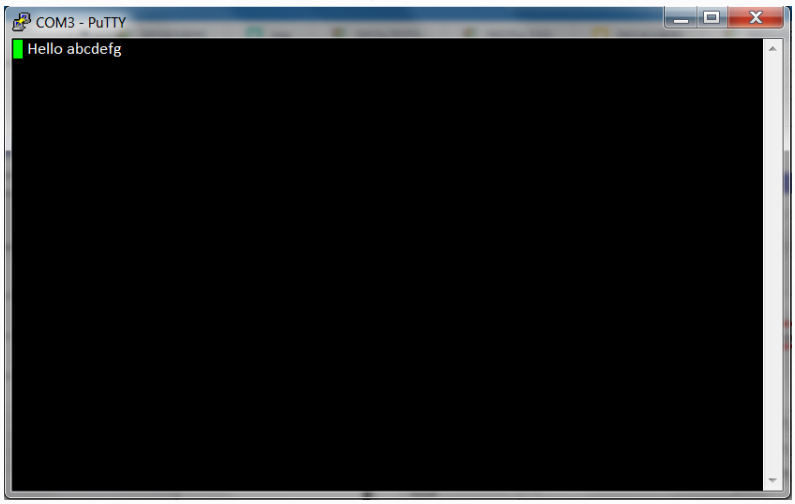

Figura 4 – Erro de código.
Fonte: imagem do autor

Imagem: Uma janela de um programa de computador chamado PuTTY, exibindo um terminal de texto.
Elementos:
Título da janela: "COM3 - PuTTY". Isso indica que a janela está conectada a uma porta serial chamada COM3 e que o programa PuTTY está sendo usado para essa conexão.
Texto na janela: "Hello abcdefg". Este é o texto visível na parte superior da janela, indicando que alguma mensagem foi enviada ou recebida pela porta serial.
Fundo preto: A maior parte da janela é preenchida com um fundo preto, típico de terminais de texto.
Cursor verde: Um cursor verde piscante está localizado na linha abaixo do texto, indicando onde a próxima entrada do usuário será inserida.
Barra de título: Na parte superior da janela, há uma barra de título azul com os botões padrão de minimizar, maximizar e fechar uma janela.
Interpretação:
A imagem mostra uma interface de linha de comando, onde os usuários podem interagir com um dispositivo ou sistema através de comandos de texto. O texto "Hello abcdefg" sugere que o dispositivo conectado à porta COM3 enviou essa mensagem como uma saudação ou como parte de um processo de inicialização.
Em resumo: A imagem mostra uma janela de um programa de computador usado para se conectar a dispositivos seriais. A janela está exibindo uma mensagem de texto e um prompt de comando, indicando que o usuário pode interagir com o dispositivo conectado.
Sugestões para descrições mais detalhadas:
Para usuários com baixa visão: "A janela é predominantemente preta, com um texto verde brilhante na parte superior. O cursor verde pisca, indicando onde você pode digitar."
Para usuários cegos: "Imagine uma tela de computador com um fundo escuro como a noite. Na parte superior, há uma linha de luz verde com as palavras 'Hello abcdefg'. Abaixo dessa linha, há um ponto verde piscante, como um cursor, indicando onde você pode digitar comandos."

4.3. Aplicações Comuns da UART

- **Comunicação com Módulos Bluetooth:** Enviar e receber dados sem fio.
- **Módulos GPS:** Receber coordenadas e dados de localização.
- **Depuração:** Enviar mensagens de log para monitorar o comportamento do código.

SAIBA MAIS

Indicação de Vídeo: Assista ao vídeo de introdução à UART para entender melhor as nuances desta interface.

5. SPI - Serial Peripheral Interface

A SPI é uma interface síncrona e de alta velocidade, ideal para a comunicação com sensores, displays e memórias externas.

5.1. Como Funciona a SPI?

A SPI utiliza um sinal de clock compartilhado entre o mestre e os escravos, garantindo uma comunicação rápida e confiável. O mestre controla o clock e envia comandos aos dispositivos escravos.
Esquema de comunicação SPI com o mestre e vários escravos

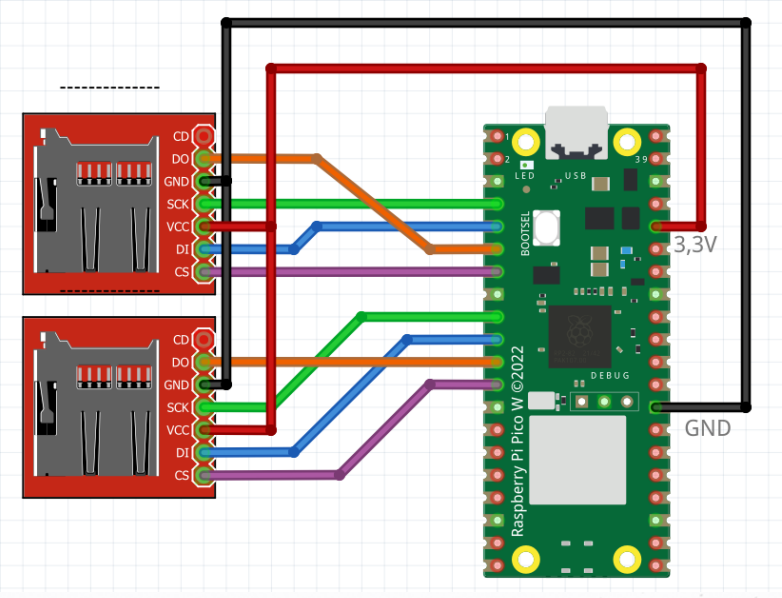

Figura 5: Diagrama de Conexão do Raspberry Pi Pico W com dois Leitores de Cartões SD
Fonte: imagem do autor

Descrição da Imagem: Diagrama de Conexão do Raspberry Pi Pico W com Dois Leitores de Cartões SD.
Imagine um circuito eletrônico. Ele representa uma conexão entre um pequeno computador chamado Raspberry Pi Pico W e dois leitores de cartões SD. Pense nisso como se você estivesse conectando dois pendrives muito pequenos a um computador de mesa.
O Raspberry Pi Pico W é a peça central. É uma placa verde, retangular, com diversos componentes eletrônicos. Ele serve como o "cérebro" do circuito, controlando tudo o que acontece.
Os leitores de cartões SD são representados por dois retângulos cinza. Eles são os dispositivos que permitem que o Raspberry Pi leia os dados armazenados nos cartões SD. Cada leitor possui várias conexões, representadas por linhas finas, que servem para transmitir dados e energia entre o leitor e o Raspberry Pi.
Conexões:
Fios coloridos: Linhas coloridas conectam os leitores de cartões SD ao Raspberry Pi. Cada cor representa um tipo diferente de conexão:
Vermelho: Geralmente indica a conexão de energia positiva (VCC).
Preto: Indica a conexão de terra (GND), que é o ponto de referência de voltagem.
Outras cores (verde, azul, etc.): Representam as conexões de dados, que permitem a transferência de informações entre os dispositivos.
Pinos: As conexões se encaixam em pinos específicos do Raspberry Pi. Esses pinos são pequenos pontos metálicos onde os fios são conectados.

IMPORTANTE

Lembre-se de conectar corretamente os pinos MOSI, MISO, SCLK e CS para garantir que os dispositivos respondam adequadamente ao mestre.

5.2. Implementação Básica da SPI no RP2040

A seguir, um exemplo de configuração básica para a SPI em C/C++:

```
#include "pico/stdlib.h"
#include "hardware/spi.h"

int main() {
    spi_init(spi0, 500000); // Inicializa SPI0 com velocidade de 500kHz
    gpio_set_function(2, GPIO_FUNC_SPI); // Configura pino 2 como SCLK
    gpio_set_function(3, GPIO_FUNC_SPI); // Configura pino 3 como MOSI
    gpio_set_function(4, GPIO_FUNC_SPI); // Configura pino 4 como MISO
    gpio_set_function(5, GPIO_FUNC_SPI); // Configura pino 5 como CS

    uint8_t data_out = 0x55; // Dados para enviar
    uint8_t data_in = 0; // Dados recebidos
    while (true) {
        gpio_put(5, 0); // Seleciona o dispositivo
        spi_write_read_blocking(spi0, &data_out, &data_in, 1); // Envia e recebe 1
byte
        gpio_put(5, 1); // Libera o dispositivo
        sleep_ms(1000); // Pausa de 1 segundo
    }
}
```

Figura 6: exemplo de configuração básica para a SPI em C/C++
Fonte: imagem do autor

Descrição da Imagem: Código em Linguagem C para Configuração SPI no Raspberry Pi Pico.
Imagine um pequeno computador, como o Raspberry Pi Pico, se comunicando com um dispositivo externo. Essa comunicação é como uma conversa, onde cada dispositivo envia e recebe informações. Para facilitar essa conversa, usamos um protocolo específico chamado SPI (Serial Peripheral Interface). O código que você está vendo é como um manual de instruções que ensina ao Raspberry Pi como configurar essa comunicação SPI.
Vamos analisar o código passo a passo:
Inclusão de bibliotecas: As primeiras linhas, #include "pico/stdlib.h" e #include "hardware/spi.h", são como se estivéssemos pegando ferramentas de uma caixa de ferramentas. Essas bibliotecas fornecem funções prontas que facilitam a programação do Raspberry Pi.
Função principal: A linha int main() { marca o início do programa principal. É como dizer: "Vamos começar a executar o código aqui".
Inicialização do SPI: A linha spi_init(spi0, 500000); configura a interface SPI (SPI0) para funcionar a uma velocidade de 500 kHz. Isso define a rapidez com que os dados serão transmitidos entre os dispositivos.
Configuração dos pinos: As próximas quatro linhas configuram os pinos do Raspberry Pi para serem utilizados pela interface SPI:
gpio_set_function(2, GPIO_FUNC_SPI);: Configura o pino 2 como o relógio (SCLK) da interface SPI. O relógio controla o ritmo da comunicação.
gpio_set_function(3, GPIO_FUNC_SPI);: Configura o pino 3 como o dado de saída mestre (MOSI). É por esse pino que o Raspberry Pi envia dados para o outro dispositivo.
gpio_set_function(4, GPIO_FUNC_SPI);: Configura o pino 4 como o dado de entrada mestre (MISO). É por esse pino que o Raspberry Pi recebe dados do outro dispositivo.
gpio_set_function(5, GPIO_FUNC_SPI);: Configura o pino 5 como a seleção de chip (CS). Esse pino é usado para selecionar qual dispositivo está sendo comunicado naquele momento.
Em resumo:
Esse código prepara o Raspberry Pi Pico para se comunicar com outros dispositivos usando o protocolo SPI. Ele configura a velocidade da comunicação e define quais pinos do Raspberry Pi serão utilizados para enviar e receber dados.

A seguir apresentamos um código um pouco mais complexo que leva em consideração os dados a serem recebidos através da comunicação SPI, utilizando a interrupção do periférico após a solicitação de dados.

```c
#include "pico/stdlib.h"
#include "hardware/spi.h"
#include "hardware/irq.h"

// Buffer para armazenar dados recebidos
#define BUFFER_SIZE 10
uint8_t rx_buffer[BUFFER_SIZE];
volatile int rx_index = 0;

// Função de callback para tratar interrupção de SPI
void on_spi_rx() {
    while (spi_is_readable(spi0)) {
        // Lê dados da SPI e armazena no buffer
        uint8_t received = spi_get_hw(spi0)->dr;
        if (rx_index < BUFFER_SIZE) {
            rx_buffer[rx_index++] = received;
        }
    }
}

int main() {
    // Inicializa SPI0 com velocidade de 500kHz, modo 0 (CPOL = 0, CPHA = 0)
    spi_init(spi0, 500 * 1000);

    // Configura pinos de SPI
    gpio_set_function(2, GPIO_FUNC_SPI); // Configura pino 2 como SCLK
    gpio_set_function(3, GPIO_FUNC_SPI); // Configura pino 3 como MOSI
    gpio_set_function(4, GPIO_FUNC_SPI); // Configura pino 4 como MISO
    gpio_set_function(5, GPIO_FUNC_SPI); // Configura pino 5 como CS

    // Habilita interrupções na SPI0
    irq_set_exclusive_handler(SPI0_IRQ, on_spi_rx);
    irq_set_enabled(SPI0_IRQ, true);
    spi_get_hw(spi0)->imsc = SPI_IMSC_RXIM_BITS;
    // Habilita interrupção de recepção

    uint8_t data_out = 0x55; // Dados para enviar

    while (true) {
        // Reseta índice do buffer
        rx_index = 0;

        // Seleciona o dispositivo
        gpio_put(5, 0);

        // Envia dados via SPI
        spi_write_blocking(spi0, &data_out, 1);

        // Aguarda até que todos os dados sejam recebidos
        sleep_ms(100);

        // Libera o dispositivo
        gpio_put(5, 1);

        // Processa os dados recebidos
        if (rx_index > 0) {
            printf("Dados recebidos: ");
            for (int i = 0; i < rx_index; i++) {
                printf("%02x ", rx_buffer[i]);
            }
            printf("\n");
        }
        sleep_ms(1000); // Pausa de 1 segundo antes do próximo ciclo
    }
}
```

Figura 6: exemplo de configuração básica para a SPI em C/C++
Fonte: imagem do autor

Descrição Detalhada do Código para Pessoas com Deficiência Visual Contexto: Imagine o Raspberry Pi Pico como um pequeno computador. Este código, escrito na linguagem C, ensina ao Pico como se comunicar com outros dispositivos eletrônicos usando um protocolo chamado SPI. É como ensinar ao Pico a falar uma língua específica para se comunicar com outros dispositivos. Explicação do Código Linha a Linha: Inclusão de Bibliotecas: #include "pico/stdlib.h" e #include "hardware/spi.h": Essas linhas são como importar ferramentas de uma caixa de ferramentas. Elas fornecem funções prontas que facilitam a programação do Pico, como funções para configurar a comunicação SPI e controlar os pinos. Definição de Buffer: #define BUFFER_SIZE 10: Define uma constante chamada BUFFER_SIZE com o valor 10. Essa constante será usada para determinar o tamanho máximo de um buffer, que é uma área de memória usada para armazenar temporariamente os dados recebidos. uint8_t rx_buffer[BUFFER_SIZE];: Cria um array (uma lista) de bytes chamado rx_buffer com o tamanho definido por BUFFER_SIZE. Esse array será usado para armazenar os dados recebidos pela interface SPI. volatile int rx_index = 0;: Cria uma variável chamada rx_index que será usada para acompanhar o próximo local livre no array rx_buffer onde um novo dado pode ser armazenado. A palavra-chave volatile indica que o valor dessa variável pode ser modificado por outras partes do programa, como interrupções. Função de Interrupção: void on_spi_rx(): Define uma função chamada on_spi_rx. Essa função será chamada automaticamente quando novos dados estiverem disponíveis na interface SPI. É como um alarme que avisa o Pico quando há algo novo para ser lido. O código dentro da função lê os dados disponíveis na interface SPI e os armazena no buffer rx_buffer. Função Principal: int main(): É o ponto de entrada do programa. Todo programa em C tem uma função main que é executada primeiro. spi_init(spi0, 500 * 1000);: Inicializa a interface SPI com uma velocidade de 500 kHz. Isso define a rapidez com que os dados serão transmitidos. As linhas seguintes configuram os pinos do Pico para serem usados pela interface SPI: gpio_set_function(2, GPIO_FUNC_SPI);: Configura o pino 2 como o relógio (SCLK). gpio_set_function(3, GPIO_FUNC_SPI);: Configura o pino 3 como o dado de saída mestre (MOSI). gpio_set_function(4, GPIO_FUNC_SPI);: Configura o pino 4 como o dado de entrada mestre (MISO). gpio_set_function(5, GPIO_FUNC_SPI);: Configura o pino 5 como a seleção de chip (CS). Em resumo: Este código configura o Raspberry Pi Pico para receber dados de outro dispositivo através da interface SPI. Quando novos dados chegam, uma interrupção é gerada e a função on_spi_rx é chamada para armazenar os dados em um buffer. Esse código é fundamental para diversas aplicações, como comunicação com sensores, controle de displays e troca de dados com outros microcontroladores.

A forma de onda a seguir ilustra um frame de comunicação da interface SPI.

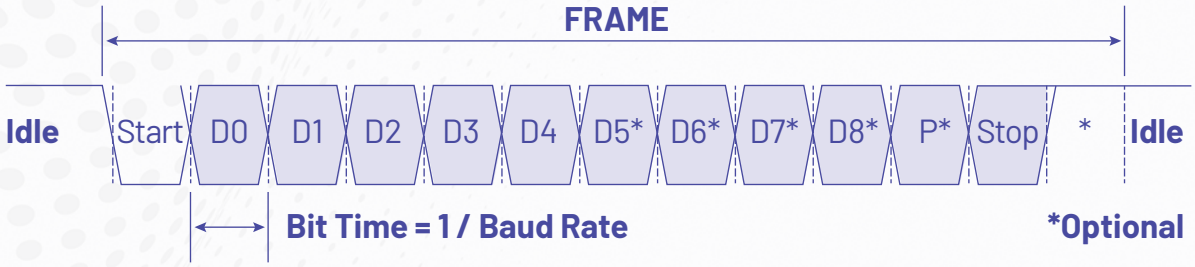

Figura 7 - Quadro de Comunicação Serial
Fonte: imagem do autor

Descrição da Imagem: Diagrama de um Quadro de Comunicação Serial.
Imagine uma linha horizontal que representa um fio por onde passam sinais elétricos. Essa linha é dividida em várias seções, cada uma representando um momento diferente na transmissão de dados.
O que cada parte representa:
Idle (inativo): O início e o fim da transmissão, onde não há dados sendo enviados. É representado por uma linha reta.
Start: Um pulso que indica o início da transmissão de um novo byte de dados. É como um sinal de "atenção" para o receptor.
D0, D1, D2, ...: Representam os bits individuais que compõem o byte de dados. Cada bit pode ter dois valores: 0 ou 1, representados por diferentes níveis de tensão no sinal.
P: Bit de paridade. É um bit adicional usado para verificar se houve erros durante a transmissão. Não está presente em todas as comunicações.
Stop: Um pulso que indica o fim da transmissão do byte.
Bit Time: A duração de cada bit, medida em tempo. A taxa de transmissão, ou baud rate, é o número de bits transmitidos por segundo.
O que o diagrama mostra:
O diagrama mostra a estrutura básica de um quadro de comunicação serial. Um quadro é uma unidade de dados que contém um byte de informação, juntamente com os bits de start e stop. A duração de cada bit é constante e determina a taxa de transmissão.

SAIBA MAIS

Dica de Leitura: Para detalhes avançados sobre a implementação de SPI, confira o datasheet do RP2040.

5.3. Aplicações Comuns da SPI

- **Displays Gráficos:** Controle de telas com alta taxa de atualização.
- **Sensores de Alta Precisão:** Coleta de dados em alta velocidade.
- **Memórias Flash:** Leitura e gravação rápidas de dados.

SAIBA MAIS

Indicação de Vídeo: Veja este tutorial prático sobre SPI para aplicar o conhecimento em um projeto real.

6. I2C - Inter-Integrated Circuit

A I2C é uma interface síncrona que utiliza apenas dois fios para

comunicação, facilitando a conexão de vários dispositivos em um único barramento.

6.1. Funcionamento da I2C

A I2C utiliza dois fios: SDA (dados) e SCL (clock). Cada dispositivo tem um endereço único e o mestre utiliza esse endereço para se comunicar com um escravo específico.

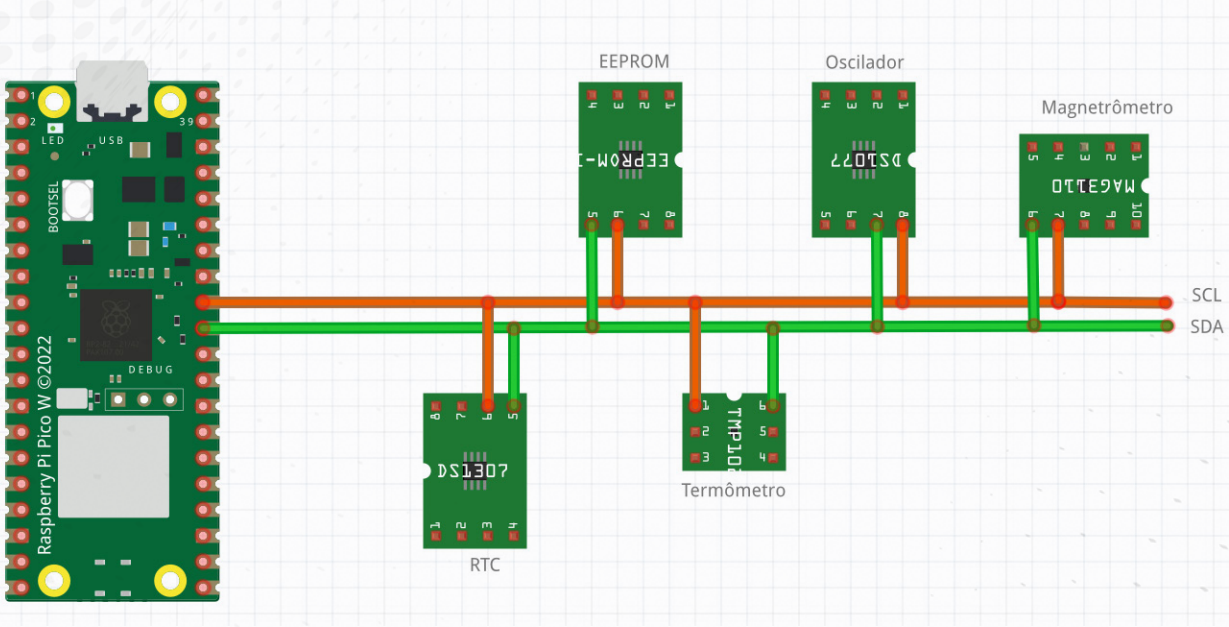

Figura 8 - Esquema de comunicação I2C com um mestre e vários escravos
Fonte: imagem do autor

Descrição da Imagem: Diagrama de Conexão do Raspberry Pi Pico W com Diversos Sensores
Imagine uma placa de circuito eletrônico, o Raspberry Pi Pico W, que é como um pequeno computador. Essa placa é o centro de um sistema que coleta informações do ambiente.
Conectado ao Raspberry Pi, há vários componentes menores, cada um com uma função específica:
EEPROM: É como uma pequena memória que armazena dados de forma permanente, mesmo quando o dispositivo é desligado. Imagine-a como um bloco de notas onde você pode escrever informações importantes e consultá-las mais tarde.
Oscilador: É um componente que gera um sinal elétrico oscilante, ou seja, um sinal que varia periodicamente entre dois valores. Esse sinal é essencial para o funcionamento de muitos circuitos eletrônicos.
Magnetômetro: É um sensor que mede a intensidade e a direção do campo magnético. Ele pode ser usado para detectar a presença de objetos metálicos ou para criar uma bússola digital.
DS1307: É um chip de relógio em tempo real (RTC, Real Time Clock). Ele mantém a hora e a data mesmo quando o dispositivo é desligado.
Termômetro: É um sensor que mede a temperatura. Ele pode ser usado para monitorar a temperatura ambiente ou a temperatura de outros componentes.
Conexões:
Todos esses componentes estão conectados ao Raspberry Pi por meio de fios coloridos. As cores dos fios geralmente indicam o tipo de conexão:
Verde: Normalmente indica uma conexão de dados chamada SCL (Clock), que é usada para sincronizar a comunicação entre os dispositivos.
Laranja: Geralmente indica uma conexão de dados chamada SDA (Data), que é usada para transmitir os dados reais entre os dispositivos

IMPORTANTE

Sempre use resistores de pull-up nos pinos SDA e SCL para garantir uma comunicação estável, sejam internos ou externos.

6.2. Implementação Básica da I2C no RP2040

A seguir é apresentado um exemplo simples de como configurar e utilizar a interface I2C em C/C++:

```c
#include "pico/stdlib.h"
#include "hardware/i2c.h"

#define I2C_ADDR 0x48 // Endereço do dispositivo I2C

int main() {
    i2c_init(i2c0, 100 * 1000); // Inicializa I2C0 a 100kHz
    gpio_set_function(4, GPIO_FUNC_I2C); // SDA
    gpio_set_function(5, GPIO_FUNC_I2C); // SCL
    gpio_pull_up(4); // Habilita pull-up para SDA
    gpio_pull_up(5); // Habilita pull-up para SCL

    uint8_t buffer[2];
    while (true) {
        i2c_read_blocking(i2c0, I2C_ADDR, buffer, 2, false); // Lê 2 bytes do
sensor
        uint16_t valor = (buffer[0] << 8) | buffer[1]; // Combina os dois bytes
        printf("Valor lido: %d\n", valor); // Exibe o valor lido
        sleep_ms(1000); // Pausa de 1 segundo
    }
}
```

Figura 9: Exemplo de código C/C++.
Fonte: imagem do autor

Descrição Detalhada do Código para Pessoas com Deficiência Visual
Imagine o Raspberry Pi Pico como um pequeno computador conectado a diversos sensores. Este código, escrito na linguagem C, ensina ao Pico como se comunicar com esses sensores utilizando um protocolo chamado I2C. É como se o Pico estivesse tendo uma conversa com cada sensor, pedindo informações e recebendo respostas.
Vamos analisar o código passo a passo:
Inclusão de Bibliotecas:
#include "pico/stdlib.h" e #include "hardware/i2c.h": Essas linhas são como importar ferramentas de uma caixa de ferramentas. Elas fornecem funções prontas que facilitam a programação do Pico, como funções para configurar a comunicação I2C e controlar os pinos.
Definição do Endereço I2C:
#define I2C_ADDR 0x48: Essa linha define o endereço do sensor que o Pico irá se comunicar. É como dar um nome para a casa de um amigo para que você possa encontrá-lo.
Função Principal:
int main(): É o ponto de entrada do programa. Todo programa em C tem uma função main que é executada primeiro.
i2c_init(i2c0, 100 * 1000);: Inicializa a interface I2C com uma velocidade de 100 kHz. Isso define a rapidez com que os dados serão transmitidos entre o Pico e o sensor.
As próximas linhas configuram os pinos do Pico para serem usados pela interface I2C:
gpio_set_function(4, GPIO_FUNC_I2C);: Configura o pino 4 como o dado (SDA).
gpio_set_function(5, GPIO_FUNC_I2C);: Configura o pino 5 como o clock (SCL).
gpio_pull_up(4); e gpio_pull_up(5);: Habilitam os resistores de pull-up nos pinos SDA e SCL, o que é necessário para a comunicação I2C.
Leitura dos Dados do Sensor:
uint8_t buffer[2];: Cria um array (uma lista) de 2 bytes chamado buffer para armazenar os dados recebidos do sensor.
while (true) { ... }: Cria um loop infinito, fazendo com que o código seja executado repetidamente.
i2c_read_blocking(i2c0, I2C_ADDR, buffer, 2, false);: Essa linha lê 2 bytes de dados do sensor e armazena no array buffer.
uint16_t valor = (buffer[0] << 8) | buffer[1];: Combina os dois bytes lidos em um único valor de 16 bits.
printf("Valor lido: %d\n", valor);: Imprime o valor lido na tela.
sleep_ms(1000);: Faz uma pausa de 1 segundo antes de ler os dados novamente.

Para realizar a leitura de dados através da comunicação I2C através de interrupção, deve-se configurar a função de call-back e também o processo de recepção e armazenamento dos dados em um Buffer.

```c
#include "pico/stdlib.h"
#include "hardware/i2c.h"
#include "hardware/irq.h"

// Endereço I2C do dispositivo slave
#define I2C_SLAVE_ADDRESS 0x42
// Buffer para armazenar os dados recebidos
uint8_t rx_buffer[16];
volatile int rx_index = 0;

// Função de callback para a interrupção I2C
void i2c0_irq_handler() {
    // Enquanto houver dados para ler na I2C
    while (i2c_get_read_available(i2c0)) {
        // Lê dados da I2C e armazena no buffer
        if (rx_index < sizeof(rx_buffer)) {
            rx_buffer[rx_index++] = i2c_read_raw_blocking(i2c0, &rx_buffer[rx_index], 1);
        }
    }
}
int main() {
    // Inicializa a I2C0 com frequência de 100 kHz
    i2c_init(i2c0, 100 * 1000);
    // Configura pinos 4 e 5 para a I2C (SDA e SCL, respectivamente)
    gpio_set_function(4, GPIO_FUNC_I2C);
    gpio_set_function(5, GPIO_FUNC_I2C);
    gpio_pull_up(4);
    gpio_pull_up(5);
    // Configura o Pico como dispositivo I2C slave
    i2c_set_slave_mode(i2c0, true, I2C_SLAVE_ADDRESS);
    // Configura o manipulador de interrupção e habilita a interrupção na I2C0
    irq_set_exclusive_handler(I2C0_IRQ, i2c0_irq_handler);
    irq_set_enabled(I2C0_IRQ, true);
    // Habilita interrupções de recebimento (RX) para o modo slave
    i2c0->hw->intr_mask = I2C_IC_INTR_MASK_M_RX_FULL_BITS;
    rx_index = 0; // Limpa o buffer antes de começar

    while (true) {
    }
}
```

Figura 10: Exemplo 2 de código C/C++.
Fonte: imagem do autor

Descrição Detalhada do Código para Pessoas com Deficiência Visual
Imagine o Raspberry Pi Pico como um pequeno computador conectado a diversos sensores. Este código, escrito na linguagem C, ensina ao Pico como se comunicar com esses sensores utilizando um protocolo chamado I2C, mas desta vez, de uma forma mais avançada, utilizando interrupções. É como se o Pico estivesse sempre atento, esperando por novos dados dos sensores, sem precisar ficar verificando constantemente se há algo novo.
Vamos analisar o código passo a passo:
Inclusão de Bibliotecas:
#include "pico/stdlib.h" e #include "hardware/i2c.h": Essas linhas são como importar ferramentas de uma caixa de ferramentas. Elas fornecem funções prontas que facilitam a programação do Pico, como funções para configurar a comunicação I2C e controlar os pinos.
Definição do Endereço I2C:
#define I2C_SLAVE_ADDRESS 0x42: Essa linha define o endereço do sensor que o Pico irá se comunicar. É como dar um nome para a casa de um amigo para que você possa encontrá-lo.
Definição do Buffer:
uint8_t rx_buffer[16];: Cria um array (uma lista) de 16 bytes chamado rx_buffer para armazenar os dados recebidos do sensor.
volatile int rx_index = 0;: Cria uma variável para acompanhar qual posição do buffer está sendo utilizada.
Função de Interrupção:
void i2c0_irq_handler(): Essa função é chamada automaticamente quando novos dados estão disponíveis na interface I2C. É como um alarme que avisa o Pico quando há algo novo para ser lido.
O código dentro da função lê os dados disponíveis na interface I2C e os armazena no buffer rx_buffer.
Função Principal:
int main(): É o ponto de entrada do programa. Todo programa em C tem uma função main que é executada primeiro.
Inicialização da I2C: Configura a interface I2C, os pinos SDA e SCL, e habilita os resistores de pull-up.
Configuração do Modo Slave: Configura o Pico para operar no modo escravo, ou seja, para receber dados de outros dispositivos.
Configuração da Interrupção: Configura a função i2c0_irq_handler para ser chamada quando houver novos dados disponíveis na interface I2C e habilita a interrupção.
Limpeza do Buffer: Zera o índice do buffer antes de começar a receber dados.
Loop Principal:
while (true) { ... }: Cria um loop infinito, fazendo com que o código seja executado repetidamente.

As formas de onda típicas de uma comunicação I2C são apresentadas a seguir.

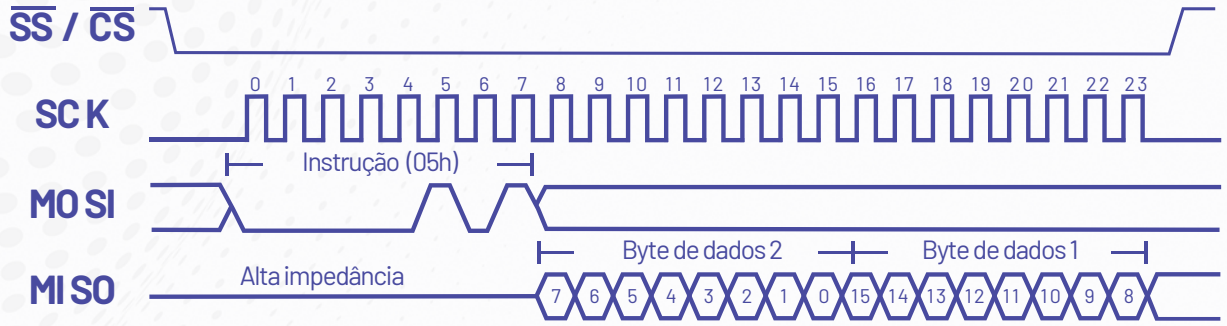

Figura 11 - Onda típicas de uma comunicação I2C.
Fonte: imagem do autor

Descrição Detalhada da Imagem: Diagrama de Tempo de uma Transação SPI.
Imagine uma tela de um osciloscópio, onde podemos visualizar sinais elétricos ao longo do tempo. A imagem que você descreveu representa um tipo específico de comunicação entre dispositivos eletrônicos, conhecido como SPI (Serial Peripheral Interface).
O diagrama mostra três sinais:
S/CS (Chip Select):* Essa linha controla se o dispositivo escravo está selecionado ou não para receber ou enviar dados. No início da transmissão, ela é ativada (nível lógico baixo) para selecionar o dispositivo. No final, ela é desativada (nível lógico alto) para desselecioná-lo.
SCK (Clock): Este sinal é um relógio que sincroniza a comunicação entre os dispositivos. Ele gera pulsos regulares que determinam o tempo em que cada bit de dado é transmitido ou recebido.
MOSI (Master Out, Slave In): Essa linha é usada para enviar dados do mestre para o escravo. Os dados são transmitidos bit a bit, sincronizados com os pulsos do relógio.
MISO (Master In, Slave Out): Essa linha é usada para enviar dados do escravo para o mestre. Os dados são transmitidos bit a bit, sincronizados com os pulsos do relógio.
Como funciona a comunicação:
Seleção do Dispositivo: O sinal S*/CS é ativado para selecionar o dispositivo escravo.
Transmissão da Instrução: O mestre envia uma instrução para o escravo, bit a bit, através da linha MOSI, sincronizado com os pulsos do relógio SCK. A instrução indica ao escravo o tipo de operação que ele deve realizar (por exemplo, ler ou escrever dados em um determinado endereço de memória).
Transmissão de Dados: Após a instrução, o mestre pode enviar dados para o escravo (se a operação for uma escrita) ou receber dados do escravo (se a operação for uma leitura). Os dados são transmitidos ou recebidos bit a bit, sempre sincronizados com os pulsos do relógio.
Desseleção do Dispositivo: Ao final da transação, o sinal S*/CS é desativado, desselecionando o dispositivo escravo.
Características da Comunicação SPI:
Síncrona: A transmissão de dados é sincronizada por um sinal de clock comum aos dois dispositivos.
Full-duplex: Permite a transmissão e recepção de dados simultaneamente.
Multi-master: Permite que múltiplos dispositivos atuem como mestres em um mesmo barramento.
Simples: A implementação da interface SPI é relativamente simples, o que a torna popular em diversas aplicações.

IMPORTANTE

Ao utilizar vários dispositivos I2C no mesmo barramento, garanta que cada dispositivo tenha um endereço único para evitar conflitos.

6.3. Aplicações Comuns da I2C

- **Sensores de Temperatura e Pressão:** Leitura de dados ambientais.
- **Displays OLED:** Exibição de informações em pequenos painéis.
- **RTC (Relógios de Tempo Real):** Controle de tempo e data em projetos embarcados.

SAIBA MAIS

Indicação de Vídeo: Confira o vídeo I2C Basics para uma introdução prática à interface.

Comparação das Interfaces

Cada interface possui características específicas que se adequam a diferentes tipos de aplicação. A tabela abaixo apresenta uma comparação entre UART, SPI e I2C.

Tabela comparativa das interfaces UART, SPI e I2C

Interface	UART	I2C	SPI
Complexidade	Simples	Fácil de comandar vários dispositivos	Complexidade aumenta com o número de dispositivos.
Velocidade	Slowest	Mais rápida que UART	Mais rápida das três
Bits por segundo (bps)	Até 115.2k	Até 400k	Até 10M
Número de dispositivos	Up to 2 devices	Up to 127	Alguns
Número de fios	3	3	4+1 por dispositivo
Nº de Masters e Slaves	Um para um	Múltiplos Masters e Slaves	1 Master, múltiplos Slaves

Tabela 2 - Onda típicas de uma comunicação I2C.
Fonte: Tabela produzida pelo autor

Esta tabela compara três interfaces de comunicação serial: UART, I2C e SPI.
A tabela tem seis linhas e quatro colunas.
A primeira linha é o cabeçalho da tabela, que contém o nome da interface.
A segunda linha é a complexidade da interface.
O UART é simples, o I2C é fácil de comandar vários dispositivos, e o SPI aumenta em complexidade com o número de dispositivos.
A terceira linha é a velocidade da interface.
O UART é o mais lento, o I2C é mais rápido que o UART, e o SPI é o mais rápido dos três.
A quarta linha é a taxa de bits da interface.
O UART tem uma taxa de bits de até 115.2k bits por segundo, o I2C tem uma taxa de bits de até 400k bits por segundo, e o SPI tem uma taxa de bits de até 10M bits por segundo.
A quinta linha é o número de dispositivos que podem ser conectados à interface.
O UART pode suportar até dois dispositivos, o I2C pode suportar até 127 dispositivos, e o SPI pode suportar alguns dispositivos.
A sexta linha é o número de fios necessários para a interface.
O UART precisa de 3 fios, o I2C também precisa de 3 fios, e o SPI precisa de 4 + 1 fios por dispositivo.
A última linha é o número de masters e slaves que a interface pode suportar.
O UART suporta um master para um slave, o I2C suporta múltiplos masters e slaves, e o SPI suporta 1 master para múltiplos slaves.

IMPORTANTE

A escolha da interface correta depende das exigências do projeto, como velocidade de comunicação, número de dispositivos e simplicidade do design.

SAIBA MAIS

Dica de Leitura: Para uma discussão detalhada sobre as vantagens e desvantagens de cada interface, consulte o livro Raspberry Pi Pico & The RP2040.

8. Aplicações Práticas e Projetos

Uma das grandes vantagens do RP2040 é a capacidade de utilizar várias interfaces simultaneamente, o que o torna ideal para projetos complexos.

8.1. Projeto Integrado: Monitoramento Ambiental

Neste projeto, utilizaremos todas as três interfaces de comunicação para coletar e exibir dados ambientais.

- **UART:** Envia mensagens de depuração para o computador.
- **SPI:** Controla um display gráfico que mostra os dados coletados.
- **I2C:** Lê dados de sensores de temperatura e pressão.

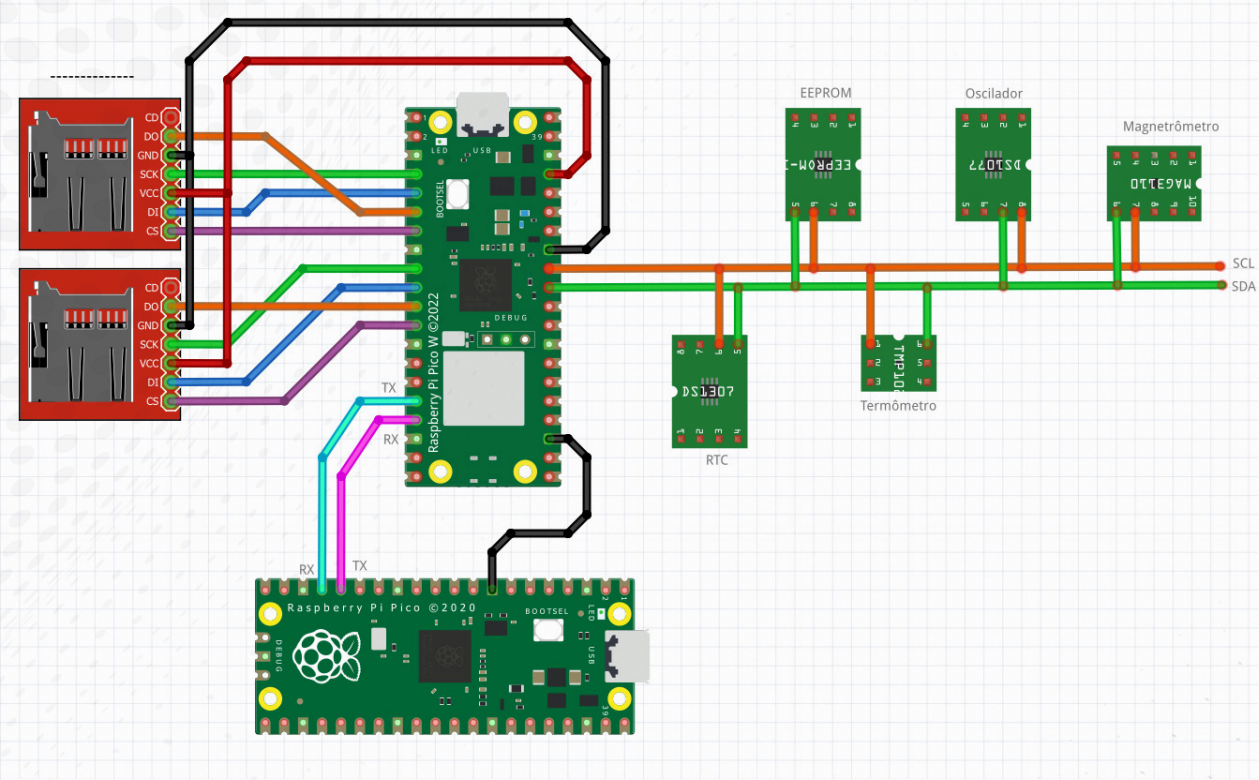

FIGURA 12 - Esquema de um projeto utilizando UART, SPI e I2C simultaneamente.
Fonte: imagem do autor

Descrição Detalhada da Imagem: Diagrama de Conexão do Raspberry Pi Pico W com Diversos Sensores
Imagine um circuito eletrônico, como um quebra-cabeça visual. No centro desse quebra-cabeça, temos o Raspberry Pi Pico W, que é como um pequeno computador. Ele está conectado a vários componentes menores, cada um com uma função específica, através de fios coloridos.
Vamos explorar cada parte:
Raspberry Pi Pico W: É a peça central do circuito. É uma placa de desenvolvimento pequena e poderosa, usada para criar diversos projetos eletrônicos.
Fios coloridos: Conectam o Raspberry Pi aos outros componentes. Cada cor geralmente indica um tipo de conexão diferente.
EEPROM: É como uma pequena memória que armazena dados de forma permanente, mesmo quando o dispositivo é desligado. Imagine-a como um bloco de notas onde você pode escrever informações importantes e consultá-las mais tarde.
Oscilador: É um componente que gera um sinal elétrico oscilante, ou seja, um sinal que varia periodicamente entre dois valores. Esse sinal é essencial para o funcionamento de muitos circuitos eletrônicos.
Magnetômetro: É um sensor que mede a intensidade e a direção do campo magnético. Ele pode ser usado para detectar a presença de objetos metálicos ou para criar uma bússola digital.
DS1307: É um chip de relógio em tempo real (RTC, Real Time Clock). Ele mantém a hora e a data mesmo quando o dispositivo é desligado.
Termômetro: É um sensor que mede a temperatura. Ele pode ser usado para monitorar a temperatura ambiente ou a temperatura de outros componentes.
Como tudo se conecta:
Os fios coloridos conectam os pinos do Raspberry Pi aos pinos dos componentes. As cores dos fios geralmente indicam o tipo de conexão:
Verde: Normalmente indica uma conexão de dados chamada SCL (Clock), que é usada para sincronizar a comunicação entre os dispositivos.
Laranja: Geralmente indica uma conexão de dados chamada SDA (Data), que é usada para transmitir os dados reais entre os dispositivos.
O que esse circuito faz:
O Raspberry Pi envia comandos através dos fios SCL e SDA para cada um dos componentes. Os componentes, por sua vez, enviam dados de volta para o Raspberry Pi através dos mesmos fios. O Raspberry Pi então processa esses dados e pode realizar diversas ações, como exibir os valores em um display, armazená-los em um arquivo ou controlar outros dispositivos.
Para que serve:
Esse tipo de circuito pode ser usado para criar diversos dispositivos, como:
Estação meteorológica: Coletar dados de temperatura, umidade e pressão atmosférica para monitorar as condições climáticas.
Robô móvel: Usar o magnetômetro para determinar a direção e o termômetro para evitar superaquecimento.
Sistema de monitoramento ambiental: Monitorar a qualidade do ar, a luminosidade e outros parâmetros ambientais

Passo a Passo:

1. Configure a UART para enviar mensagens de depuração a cada leitura de sensor.
2. Utilize a SPI para atualizar o display com os dados coletados.
3. Implemente a I2C para ler os valores dos sensores a cada segundo.

IMPORTANTE

Observação: A combinação de interfaces permite uma maior flexibilidade e expansão de projetos. Não hesite em experimentar diferentes combinações e testar a performance do seu sistema.

SINTESE

Revisando, aprendemos que as interfaces de comunicação serial são essenciais para conectar e integrar diferentes dispositivos em sistemas embarcados. Exploramos as três principais interfaces disponíveis no RP2040 da Raspberry Pi Pico W: UART, SPI e I2C. A UART, com sua simplicidade, é ideal para comunicações ponto a ponto e depuração de código. A SPI se destaca pela velocidade e controle de múltiplos dispositivos simultaneamente, enquanto a I2C permite conectar diversos periféricos com apenas dois fios, tornando o design do circuito mais simples. Através de exemplos práticos, vimos como configurar e programar cada interface em C/C++, garantindo a comunicação eficiente entre dispositivos como sensores, displays e módulos de comunicação. Ao compreender suas características e aplicações, você estará mais preparado para desenvolver projetos robustos e escaláveis.

Não se esqueça de revisar o material e testar os exemplos práticos apresentados neste e-book. Experimente configurar as interfaces de comunicação 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!

CONCLUSÃO

Na próxima etapa, vamos aprofundar nossos conhecimentos sobre o desenvolvimento de sistemas embarcados, explorando o periférico de modulação por largura de pulso, PWM. Pense sobre as seguintes perguntas: Como você poderia utilizar UART, SPI e I2C simultaneamente em um único projeto? Que tipo de desafios você poderia enfrentar ao conectar múltiplos dispositivos ao RP2040?
Obrigado por acompanhar este guia sobre interfaces de comunicação com o RP2040. 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. Documentação oficial do RP2040.
2. Exemplos de código para Raspberry Pi Pico.
3. UART Explained.
4. Raspberry Pi Pico & The RP2040.
5. I2C Basics.


embarca
tech
Residência
Tecnológica
em Sistemas
Embarcados