embarca tech | Residência Tecnológica em Sistemas Embarcados

Interrupções

Unidade 4 | Capítulo 4

Otávio Alcântara de Lima Jr.

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

Unidade 4

Capítulo 4

1. Boas-vindas e Interrupções

Olá, estudantes! Sejam bem-vindos ao material de apoio sobre "Interrupções". Este material complementa o Capítulo 4 da Unidade 4 sobre Microcontroladores, oferecendo apoio e conteúdo extra para aprofundar seu aprendizado.

Sabemos que os microcontroladores possuem uma ampla variedade de periféricos, dos mais simples aos mais complexos. Esses periféricos permitem que os microcontroladores interajam com o mundo físico e lidem com temporizações, essenciais para o desenvolvimento de aplicações embarcadas. Os eventos gerados por esses dispositivos precisam ser tratados pelo microcontrolador. Até o momento, você trabalhou com o módulo GPIO, monitorando os pinos para identificar alterações. No entanto, com o mecanismo de interrupções, podemos delegar essa tarefa de monitoramento ao hardware, liberando o processador para outras atividades.

As **interrupções** são uma ferramenta fundamental no desenvolvimento de sistemas embarcados com microcontroladores. Elas permitem um uso mais eficiente do recurso mais valioso: o tempo do processador. Ao invés de ocupar o processador com o monitoramento contínuo dos periféricos, o hardware pode assumir essa função, gerando uma interrupção somente quando um evento ocorrer.

O tratamento de interrupções envolve tanto hardware quanto software. O software é responsável por configurar um periférico para identificar e gerar interrupções em determinados eventos. Os periféricos já possuem circuitos dedicados a essa função. Quando um evento acontece, o periférico envia um sinal ao controlador de interrupções que, caso a interrupção esteja habilitada, interrompe a execução do processador. O hardware do processador, automaticamente, salva parte do seu contexto de execução, como o contador de programa e o registrador de status, e redireciona a execução para um endereço na memória, onde se encontra a **rotina de tratamento de interrupção.**

Neste ponto, o software do sistema assume, salvando os demais registradores importantes do contexto de execução. A seguir, a rotina de tratamento da interrupção é executada. Após a sua conclusão, o processador restaura o contexto salvo e retoma a execução do ponto onde havia sido interrompido.

Embora pareça complexo, o mecanismo de interrupções é relativamente simples de configurar quando usamos o SDK do Raspberry Pi Pico [1], já que ele se encarrega de boa parte dos detalhes do hardware. Neste material de apoio, você encontrará conteúdo teórico sobre as técnicas de tratamento de entrada e saída, com ênfase na entrada e saída controlada por interrupções. Vamos explorar como o SDK facilita a configuração de interrupções, começando com o módulo GPIO. Neste primeiro momento, focaremos nas interrupções desse módulo, mas, ao longo do curso, você terá a oportunidade de aplicar o mesmo conceito a outros periféricos.

Agora, vamos ver como o conteúdo deste documento está organizado. Começamos com uma breve descrição comparativa das técnicas de entrada e saída. Isso é importante para que você compreenda as vantagens e desvantagens do uso de interrupções. Em seguida, voltamos nossa atenção para o funcionamento do mecanismo de interrupções no RP2040 [2], o microcontrolador da nossa placa BitDogLab. Na seção seguinte, discutimos como utilizar as interrupções do módulo GPIO, empregando as funções do SDK. Por fim, concluímos o material com um resumo das informações apresentadas e os próximos passos.

2. Técnicas de Entrada e Saída

Os periféricos em um sistema embarcado são essenciais para interagir com o mundo físico, seja através de interfaces de comunicação, pinos digitais, entradas analógicas ou para gerar temporizações nas nossas ações. Em um sistema microcontrolado, existe uma grande variedade de dispositivos periféricos que formam o sistema de entrada e saída. Esses dispositivos têm diferentes demandas de transmissão de dados e eventos. Alguns são mais simples, como o módulo GPIO, enquanto outros são mais complexos, como o PIO ou o controlador USB.

De modo geral, nosso código precisa interagir com os periféricos para configurá-los, tratar seus eventos e utilizar os recursos que eles oferecem. Grande parte do desenvolvimento de software para sistemas

microcontrolados envolve o gerenciamento desses componentes. Por isso, é fundamental compreender as técnicas de entrada e saída e saber como aplicá-las corretamente em nossas aplicações.

Parte do trabalho de configuração dos periféricos é facilitada pelas bibliotecas do SDK, que nos poupam do esforço de configurar manualmente os registradores desses dispositivos. No entanto, uma parte crucial do processo — tratar os eventos gerados pelos periféricos — fica a cargo do nosso código, com o auxílio do SDK. E é exatamente essa parte que vamos explorar nesta seção.

Vamos apresentar e comparar três técnicas de entrada e saída: **E/S programada, E/S controlada por interrupção e DMA.**

A primeira técnica, **E/S programada**, é a mais simples e você vem utilizando desde o início desta unidade. Nessa abordagem, o processador tem controle total sobre as operações de entrada e saída. O código que você desenvolve é responsável por monitorar o estado do periférico, tratar os eventos e realizar a transferência de dados. Em resumo, todo o processo é gerenciado diretamente pelo processador.

Nessa técnica, o processador fica ocioso, aguardando a conclusão das operações de entrada e saída. Isso leva a um desperdício do nosso recurso mais valioso: o tempo do processador. Além disso, consome energia de forma desnecessária. Um exemplo prático que você já utilizou em aulas anteriores foi a leitura das entradas dos pinos GPIO. Para realizar essa leitura, o processador precisa monitorar constantemente o estado do pino **(poolling)**, verificando se houve alguma mudança. Esse tempo de espera pode se estender e impactar outras atividades que o processador poderia estar executando.

A segunda técnica é chamada de **E/S controlada por interrupção**. Nesse modelo, o periférico gera uma interrupção sempre que um evento de entrada ou saída acontece. Nesse momento, a execução do processador é interrompida para atender à solicitação, por meio de uma rotina de tratamento de interrupções. Com essa abordagem, o processador não precisa esperar pela conclusão das operações de entrada e saída; ele é notificado assim que essas operações são finalizadas.

Essa estratégia permite um uso mais eficiente do tempo do processador, já que ele fica livre para realizar outras tarefas enquanto aguarda o evento. No entanto, é importante ter cuidado com o uso de interrupções, pois elas introduzem concorrência no sistema. Isso significa que múltiplas linhas de execução podem ser disparadas de forma assíncrona, dependendo dos eventos gerados pelos periféricos. Esse cenário pode levar a problemas clássicos de programação concorrente [3], como **condições de disputa** (*race conditions*) e **starvation**.

Outra questão a considerar é que interrupções funcionam bem para eventos que ocorrem em **baixa frequência**. Se as interrupções forem disparadas com muita frequência, e o tempo de execução das rotinas de interrupção for alto, o laço principal da função main() poderá ser interrompido constantemente. Isso pode comprometer a execução correta do programa, já que o processador passará grande parte do tempo lidando com interrupções, em vez de executar as demais tarefas.

Para evitar esses problemas, recomendamos que você tome algumas precauções. Primeiro, tenha cuidado com as variáveis e estruturas de dados que são compartilhadas entre uma rotina de interrupção e o programa principal. Se essas variáveis forem modificadas tanto na rotina de interrupção quanto no fluxo normal do programa, é necessário garantir que a alteração seja **atômica [3]**, ou seja, que a operação não possa ser interrompida no meio.

Outro ponto importante é que as rotinas de interrupção devem ser leves, pequenas e simples. Evite adicionar complexidade a essas funções. Quando precisar realizar tarefas mais complexas na ocorrência de um evento, prefira usar uma variável **flag** para sinalizar no laço principal que o evento precisa ser tratado. Ao longo do curso, iremos apresentar técnicas que permitem lidar com várias tarefas em um mesmo programa, sem a necessidade de usar um **RTOS** (Sistema Operacional de Tempo Real).

A terceira técnica é o **acesso direto à memória (DMA)**, que auxilia o processador no tratamento de periféricos que geram grandes volumes de dados. O DMA funciona como um co-processador especializado na movimentação de dados entre a memória e os periféricos. O processador apenas configura o DMA para operar com um determinado periférico, como um conversor analógico-digital (ADC).

Por exemplo, se em seu projeto você precisar atender a um requisito de taxa de amostragem, o processador pode configurar o DMA para trabalhar junto com o ADC, de modo que as amostras sejam salvas diretamente em um buffer na memória principal. O DMA notificará o processador, por meio de uma interrupção, quando todas as amostras tiverem sido armazenadas. Dessa forma, o processador não precisa se preocupar em mover os dados entre o periférico e a memória, liberando tempo para realizar outras tarefas mais complexas.

Por fim, apresentaremos um **quadro comparativo [4]** com as **vantagens e desvantagens** de cada uma das técnicas de entrada e saída discutidas. É importante notar que, em uma aplicação real, podemos combinar uma ou mais dessas técnicas, dependendo da complexidade dos periféricos envolvidos e das **restrições de tempo real** do projeto. Por isso, é essencial dominar cada uma dessas abordagens. Nesta aula, o foco será na **entrada e saída controlada por interrupção**, explorando seus benefícios e aplicações práticas.

	Técnica	Vantagens	Desvantagens
1	E/S Programada	Simplicidade na implementação; Controle direto do processador sobre os periféricos.	Processador sobrecarregado; Ineficiente para sistemas com muitos periféricos ou dados frequentes; Baixa escalabilidade.
2	E/S Controlada por Interrupção	Melhor uso do processador; Processador não precisa verificar constantemente os periféricos; Maior eficiência.	Maior complexidade de implementação; Pode exigir mais cuidado com sincronização e gerenciamento de eventos.
3	DMA	Descarrega a transferência de dados do processador; Muito eficiente em transferências grandes de dados; Processador livre para outras tarefas.	Maior complexidade no setup; Limitado a transferências específicas; Nem todos os periféricos suportam DMA.

Fig.1. Comparativo com as vantagens e desvantagens de cada uma das técnicas de entrada e saída.
Fonte: imagem gerada pelo autor

3. Interrupções no RP2040

O microcontrolador **RP2040** possui dois núcleos de processamento **ARM Cortex-M0+**, e cada um deles conta com um módulo **NVIC** (Nested Vectored Interrupt Controller) para auxiliar no tratamento de interrupções. Isso significa que qualquer um dos núcleos pode tratar as interrupções dos periféricos. No entanto, nas aplicações que desenvolveremos ao longo deste curso, utilizaremos apenas um núcleo de processamento.

O módulo **NVIC** permite gerenciar até **32 fontes de interrupção**, mas, no caso do RP2040, apenas **26 vetores de interrupção** são realmente utilizados. O quadro a seguir apresentará os endereços das **IRQ** (Interrupt Request), assim como os periféricos responsáveis por gerar essas interrupções. Note que todos os periféricos do microcontrolador possuem, no mínimo, um vetor de interrupção associado.

Um periférico que merece destaque é o **módulo GPIO**, que possui apenas um vetor de interrupção compartilhado para eventos em todos os pinos. Na próxima seção, vamos nos aprofundar um pouco mais sobre as interrupções desse módulo. Já os outros periféricos serão estudados de forma mais detalhada ao longo do curso.

IRQ	Fonte	IRQ	Fonte	IRQ	Fonte	IRQ	Fonte	IRQ	Fonte
0	TIMER_IRQ_0	6	XIP_IRQ	12	DMA_IRQ_1	18	SPI0_IRQ	24	I2C1_IRQ
1	TIMER_IRQ_1	7	PIO0_IRQ_0	13	IO_IRQ_BANK0	19	SPI1_IRQ	25	RTC_IRQ
2	TIMER_IRQ_2	8	PIO0_IRQ_1	14	IO_IRQ_QSPI	20	UART0_IRQ		
3	TIMER_IRQ_3	9	PIO1_IRQ_0	15	SIO_IRQ_PROC0	21	UART1_IRQ		
4	PWM_IRQ_WRAP	10	PIO1_IRQ_1	16	SIO_IRQ_PROC1	22	ADC_IRQ_FIFO		
5	USBCTRL_IRQ	11	DMA_IRQ_0	17	CLOCKS_IRQ	23	I2C0_IRQ		

Fig.2. Lista de interrupções que podem ocorrer em um sistema eletrônico.
Fonte: imagem gerada pelo autor

4. Interrupções do GPIO

Nesta seção, trataremos das interrupções do módulo GPIO. De forma simples, os eventos que geram uma interrupção no **GPIO** estão relacionados a mudanças no nível ou borda de um sinal de entrada. Em vez de utilizar a técnica de **polling** para monitorar continuamente o estado da entrada, podemos configurar o módulo de interrupções para nos alertar automaticamente quando ocorrerem essas mudanças. O quadro a seguir apresentará os quatro eventos principais que podem gerar interrupções neste periférico.

Evento	Descrição
GPIO_IRQ_LEVEL_LOW	Nível baixo em um pino
GPIO_IRQ_LEVEL_HIGH	Nível alto em um pino
GPIO_IRQ_EDGE_FALL	Borda de descida, ou seja, mudança de alto para baixo.
GPIO_IRQ_EDGE_RISE	Borda de subida, ou seja, mudança de baixo para alto.

Fig.3. Interrupções do módulo GPIO
Fonte: imagem gerada pelo autor

Desses quatro eventos, os dois últimos são mais utilizados. Eles permitem identificar se houve um evento de borda em um dos pinos, podendo ser usados em várias aplicações da leitura de botões a medição da largura de um pulso digital.

Para configurar a interrupção do módulo **GPIO**, utilizaremos a estrutura fornecida pelo **SDK**, que nos permite abstrair os detalhes de configuração dos registradores e focar no código para tratar o evento. A configuração é realizada por meio da função **gpio_set_irq_enable_with_callback**, que recebe quatro parâmetros.

O primeiro parâmetro é um inteiro que representa o número do pino onde desejamos habilitar a interrupção. O segundo é uma **máscara de bits** que especifica quais eventos iremos monitorar. Podemos combinar múltiplos eventos utilizando a operação **OU bit-a-bit**. Por exemplo, ao combinar **GPIO_IRQ_EDGE_RISE | GPIO_IRQ_EDGE_FALL,** indicamos que queremos detectar tanto borda de subida quanto borda de descida.

O terceiro parâmetro é utilizado para **habilitar** ou **desabilitar** a interrupção. O quarto e último parâmetro é um **ponteiro para a função callback**, que

será chamada quando o SDK detectar e tratar a interrupção no GPIO. Essa função **callback** é implementada pelo programador e será responsável por tratar a interrupção de acordo com a lógica da aplicação. Essa função de callback precisa ter dois parâmetros. O primeiro é um inteiro que identifica qual pino do GPIO causou a interrupção. O segundo é uma máscara que identifica qual evento foi gerado, seguindo o padrão da máscara usada na configuração da interrupção.

O SDK oferece uma base sólida para o tratamento de interrupções, permitindo que o programador não precise se preocupar com os detalhes de configuração de hardware ou gerenciamento de vetores de interrupção. Dessa forma, o foco pode ser direcionado para o tratamento do evento que gerou a interrupção. Usar o SDK é recomendado, pois ele fornece um código validado, adequado para a maioria das aplicações. É como se o SDK fizesse toda a "arrumação da casa", simplificando o processo.

No entanto, mesmo com esse suporte, é essencial ter cuidado ao escrever as funções de callback para interrupções. Como mencionado anteriormente, interrupções podem introduzir desafios relacionados à programação concorrente. Esses desafios se tornam ainda mais complexos ao lidar com múltiplas interrupções ou interrupções aninhadas. Embora a programação concorrente não seja o foco deste material, uma dica importante é evitar que uma variável ou estrutura de dados seja modificada simultaneamente pela interrupção e pelo programa principal. Se for necessário, assegure-se de que a modificação seja feita por meio de uma operação atômica.

Outro aspecto crucial na escrita de funções de callback é mantê-las simples, rápidas e concisas. Funções longas ou complexas podem dificultar a depuração de erros, aumentando as chances de falhas. Além disso, evite chamar funções de entrada e saída, como printf ou similares, dentro das interrupções. Se o evento exigir um tratamento mais elaborado, é melhor que isso seja feito no laço principal (super loop da função main).

Outro cuidado importante é declarar as variáveis modificadas dentro de uma interrupção utilizando o qualificador volatile. Esse qualificador informa ao compilador que os valores da variável podem ser alterados de forma assíncrona, ou seja, fora do fluxo normal do programa, como é o caso de interrupções. Se não utilizarmos volatile, o compilador pode

aplicar otimizações que, embora melhorem o desempenho do código, podem levar a resultados inesperados e incorretos, pois o compilador não saberá que a variável pode ser modificada em outro contexto, como em uma interrupção. Portanto, ao declarar uma variável que pode ser alterada dentro de uma interrupção, sempre utilize o volatile para garantir o comportamento correto do programa.

O quadro abaixo mostra um passo a passo para a configuração de um evento de borda de descida em um pino do GPIO. Basicamente, você precisa configurar um pino como entrada e habilitar a interrupção designando uma função de callback.

Etapa	Código	Observações
Configuração do pino como entrada, pull-up.	gpio_init(button_a); gpio_set_ dir(button_a,GPIO_IN); gpio_pullup(button_a);	Nessa seção de código, fazemos a inicialização de um pino endereçado pela constante button_a como entrada com resistor de pull-up.
Configuração da interrupção	gpio_set_irq_ enabled(button_a, GPIO_ IRQ_EDGE_FALL, true, gpio_irq_callback);	Esta função configura a interrupção de borda de descida para o pino endereçado por button_a
Função de callback	void gpio_irq_callback(int gpio, uint32_t events){...}	Esta é a declaração da rotina de callback que é chamada quando ocorrer o evento de borda de descida no pino button_a. Lembre-se que há apenas uma rotina dessas para todos os pinos, por isso, você precisa usar as entradas gpio para identificar qual pino gerou a interrupção e events para saber qual foi o evento.

Fig.4. Passo a passo para a configuração de um evento de borda de descida em um pino do GPIO
Fonte: imagem gerada pelo autor

Glossário

Vetor de Interrupções:
O vetor de interrupções é uma estrutura de dados utilizada por sistemas baseados em microcontroladores para gerenciar interrupções. Ele consiste em uma tabela que armazena os endereços de memória das rotinas de tratamento de interrupções (ISR – Interrupt Service Routines). Cada posição do vetor corresponde a uma interrupção específica, e, quando um evento de interrupção ocorre, o processador consulta essa tabela para localizar e executar a rotina apropriada. Isso permite que o processador responda rapidamente a eventos externos sem a necessidade de verificar constantemente o estado dos periféricos, otimizando o uso de recursos e melhorando o desempenho geral do sistema.

NVIC:

O **NVIC** (Nested Vectored Interrupt Controller) é um módulo presente em microcontroladores baseados na arquitetura ARM Cortex, responsável por gerenciar as interrupções. Ele permite que o processador responda a eventos externos de forma eficiente, priorizando as interrupções com diferentes níveis de importância. O NVIC oferece suporte a interrupções aninhadas, ou seja, ele permite que uma interrupção de maior prioridade interrompa outra de menor prioridade que já está sendo tratada. Além disso, o NVIC reduz o tempo de latência, pois possibilita que o processador identifique rapidamente a interrupção ativa e inicie a execução da rotina de tratamento correspondente. No caso do RP2040, o NVIC é capaz de gerenciar até 32 fontes de interrupção, sendo fundamental para a implementação de sistemas com múltiplos periféricos e eventos.

IRQ (Interrupt Request):

IRQ, ou **Interrupt Request**, é um sinal gerado por um dispositivo periférico para alertar o processador de que uma interrupção

deve ser tratada. Quando um periférico precisa de atenção — por exemplo, quando um botão é pressionado ou um dado é recebido por uma porta de comunicação — ele envia uma solicitação de interrupção (IRQ) ao processador. Ao receber essa solicitação, o processador pausa temporariamente a execução do programa principal e chama a rotina de tratamento de interrupção associada ao IRQ em questão. Depois de resolver o evento, o processador retorna à execução normal do programa, minimizando o tempo gasto com operações de entrada e saída.

Poolling:

Poolling é uma técnica de entrada e saída em que o processador verifica constantemente o estado de um periférico para detectar eventos, como mudanças de estado em um sensor ou a chegada de dados em uma interface de comunicação. Nesse método, o processador executa um loop de repetição onde, periodicamente, consulta o periférico para saber se ele precisa de atenção. Embora seja simples de implementar, o pooling pode ser ineficiente, pois o processador desperdiça tempo de execução monitorando ativamente os periféricos, em vez de realizar outras tarefas. Isso pode resultar em um uso ineficiente de recursos, especialmente em sistemas que exigem alta performance ou tempo real, onde outras técnicas, como interrupções, podem ser mais adequadas.

Condição de Disputa (Race Condition):

Uma condição de disputa ocorre quando dois ou mais trechos de código acessam e modificam a mesma variável ou recurso compartilhado de maneira concorrente. Se não houver sincronização adequada, o resultado final pode depender da ordem em que as operações ocorrem, levando a comportamentos inesperados e bugs difíceis de rastrear. Esse problema é comum em sistemas com múltiplas interrupções ou em ambientes multitarefa.

Starvation:

Starvation acontece quando uma tarefa ou processo em um sistema é impedido de obter os recursos de que precisa para continuar a execução, pois outros processos estão constantemente ocupando esses recursos. Em sistemas com interrupções de alta prioridade, por exemplo, tarefas de menor prioridade podem ser deixadas de lado por tempo indefinido, causando problemas de desempenho e respostas inadequadas do sistema.

Operação Atômica:

Uma operação atômica é uma operação que é executada de forma completa, sem a possibilidade de ser interrompida no meio do processo. Em sistemas concorrentes, como aqueles que utilizam interrupções, garantir a atomicidade de certas operações é crucial para evitar problemas como condições de disputa. Um exemplo de operação atômica é a leitura ou escrita de uma variável em uma única instrução de máquina.

RTOS (Real-Time Operating System):

Um RTOS, ou Sistema Operacional de Tempo Real, é um tipo de sistema operacional projetado para gerenciar tarefas e recursos em sistemas embarcados de forma previsível e com tempos de resposta garantidos. Ele é utilizado em aplicações que exigem alta confiabilidade e precisão no tratamento de eventos em tempo real, como sistemas de controle industrial, automação e dispositivos médicos.

Função de callback:

Uma **função de callback** é uma função definida pelo programador que é passada como parâmetro para outra função e chamada automaticamente quando ocorre um determinado evento. Em sistemas embarcados, as funções de callback são amplamente utilizadas no tratamento de interrupções. Quando configuramos uma interrupção, associamos uma função de callback que será executada assim que o evento de interrupção ocorrer. Essa

abordagem permite separar a lógica do evento (por exemplo, uma mudança em um pino de entrada) da lógica de tratamento (o que fazer quando esse evento acontece). Isso facilita a manutenção do código, melhora a modularidade e torna o sistema mais responsivo, já que a função de callback é chamada de forma imediata assim que o evento ocorre, sem necessidade de monitoramento contínuo.

Volatile:

O qualificador volatile é utilizado em C para indicar ao compilador que o valor de uma variável pode ser alterado por eventos externos ao fluxo normal do programa, como interrupções ou hardware. Isso significa que o compilador não deve aplicar otimizações que assumam que o valor da variável permanecerá constante entre acessos. Sem o uso de volatile, o compilador poderia, por exemplo, armazenar o valor da variável em um registrador e ignorar mudanças feitas externamente, levando a comportamentos incorretos em sistemas embarcados.

Static:

O qualificador static tem duas funções principais em C. Quando aplicado a uma variável dentro de uma função, ele preserva o valor da variável entre múltiplas execuções da função, permitindo que a variável mantenha seu estado. Além disso, quando usado em variáveis globais ou funções, static limita o escopo ao arquivo onde a variável ou função foi declarada, impedindo que outras partes do código acessem diretamente esses elementos. Isso é útil para encapsular o comportamento e evitar conflitos de nome em projetos maiores.
/Glossário

Conclusão

O tratamento de eventos de entrada e saída é uma das atividades centrais em programas embarcados. O uso de interrupções simplifica esse processo e permite um uso mais eficiente do processador, já que ele pode executar outras tarefas enquanto aguarda um evento. No caso do Raspberry Pi Pico, o SDK oferece uma base de código que facilita o desenvolvimento, permitindo que o programador foque no tratamento do

evento em si, sem a necessidade de se preocupar com os detalhes de hardware. Isso acelera o desenvolvimento e torna o código mais simples de gerenciar.

No entanto, apesar de serem extremamente úteis, as interrupções podem trazer uma camada de complexidade. Elas são ideais para eventos de baixa frequência, pois em situações com muitos eventos em sequência, ou de alta frequência, o gerenciamento das interrupções pode se tornar mais complicado. Nessas circunstâncias, é necessário avaliar cuidadosamente qual técnica de entrada e saída é mais adequada, considerando-se interrupções, poolling ou DMA para o caso específico.

Aproveite esta oportunidade e experimente replicar o exemplo de código apresentado na aula em sua placa. Espero que este material de apoio seja útil e enriquecedor para o seu aprendizado. Por favor, aprofunde seus estudos por meio das referências deste texto, que servem como material suplementar. Vamos explorar juntos o fascinante mundo dos microcontroladores!

REFERÊNCIAS

[1] RASPBERRY PI LTD. Raspberry Pi Pico C/C++ SDK. 2024. Disponível em: https://datasheets.raspberrypi.com/pico/raspberry-pi-pico-c-sdk.pdf. Acesso em: 03 set. 2024.

[2] RASPBERRY PI LTD. RP2040 Datasheet. 2024. Disponível em: https://datasheets.raspberrypi.com/rp2040/rp2040-datasheet.pdf. Acesso em: 03 set. 2024.

[3] BitDogLab. Manual BitDogLab, Disponível em: https://github.com/BitDogLab/BitDogLab/tree/main/doc. Acesso em: 03 set. 2024.

[4] TANENBAUM, Andrew S.; BOS, Herbert. Sistemas Operacionais Modernos. 4. ed. São Paulo: Pearson, 2016.

[5] STALLINGS, William. Arquitetura e Organização de Computadores. 9. ed. São Paulo: Pearson, 2016.


embarca
tech
Residência
Tecnológica
em Sistemas
Embarcados