# Roteiro de Experimento: Transferência DMA com Interrupção na BitDogLab

## 1. Título do Experimento

Transferência de Dados usando DMA e Interrupção no RP2040 (Projeto: isr\_dma\_bitdoglab)

## 2. Objetivo

- Entender o funcionamento básico do DMA (Direct Memory Access) no RP2040.  
- Configurar transferências automáticas de dados entre áreas de memória.  
- Utilizar interrupções para detectar a conclusão de uma transferência.

## 3. Materiais Necessários

- Placa BitDogLab.  
- LED integrado no GPIO11.  
- Computador com VSCode e Pico SDK configurado.

## 4. Diagrama de Montagem

• LED conectado ao GPIO11.  
• (A BitDogLab já possui esse LED integrado, sem necessidade de montagem adicional.)

## 5. Procedimento

### 5.1. Inicialização do projeto

• Criar um novo projeto no VSCode com suporte ao Pico SDK.  
• Nome sugerido: isr\_dma\_bitdoglab.

### 5.2. Código Fonte Básico

#include <stdio.h>  
#include "pico/stdlib.h"  
#include "hardware/dma.h"  
#include "hardware/irq.h"  
  
#define LED\_PIN 11  
#define TAMANHO 16  
  
uint8\_t origem[TAMANHO] = {1,2,3,4,5,6,7,8,9,10,11,12,13,14,15,16};  
uint8\_t destino[TAMANHO];  
  
int canal\_dma;  
  
void dma\_isr() {  
 dma\_hw->ints0 = 1u << canal\_dma;  
 gpio\_put(LED\_PIN, 1);  
 printf("✅ Transferência DMA finalizada!\n");  
}  
  
int main() {  
 stdio\_init\_all();  
 sleep\_ms(2000);  
 printf("Iniciando exemplo de DMA com interrupção...\n");  
  
 gpio\_init(LED\_PIN);  
 gpio\_set\_dir(LED\_PIN, GPIO\_OUT);  
 gpio\_put(LED\_PIN, 0);  
  
 canal\_dma = dma\_claim\_unused\_channel(true);  
 dma\_channel\_config config = dma\_channel\_get\_default\_config(canal\_dma);  
  
 channel\_config\_set\_transfer\_data\_size(&config, DMA\_SIZE\_8);  
 channel\_config\_set\_read\_increment(&config, true);  
 channel\_config\_set\_write\_increment(&config, true);  
  
 dma\_channel\_set\_irq0\_enabled(canal\_dma, true);  
 irq\_set\_exclusive\_handler(DMA\_IRQ\_0, dma\_isr);  
 irq\_set\_enabled(DMA\_IRQ\_0, true);  
  
 dma\_channel\_configure(  
 canal\_dma,  
 &config,  
 destino,  
 origem,  
 TAMANHO,  
 true  
 );  
  
 while (true) {  
 tight\_loop\_contents();  
 }  
}

## 6. Análises e Observações

- O DMA transfere automaticamente dados do vetor origem para destino.  
- Nenhuma ação explícita da CPU é necessária após iniciar a transferência.  
- Ao final da transferência, o LED no GPIO11 é aceso e uma mensagem é exibida no terminal.  
- Discuta:  
 • Vantagens do uso do DMA para liberar a CPU.  
 • O que poderia acontecer se acessássemos destino antes da transferência terminar.

## 7. Extensões e Desafios

- Fazer múltiplas transferências sequenciais, com LEDs diferentes.  
- Modificar o tamanho da transferência e observar o comportamento.  
- Implementar transferência circular (modo contínuo).  
- Usar DMA para transferir dados para periféricos como SPI, UART ou PIO.

## 8. Importante

- Sempre limpar a interrupção do DMA dentro do handler.  
- Ajustar corretamente as configurações de incremento e tamanho de dados.  
- O uso de tight\_loop\_contents() mantém o sistema em estado de espera eficiente.

## 9. Aplicações Reais

- O DMA é essencial em sistemas embarcados que exigem alta taxa de transferência de dados.  
- Bons exemplos de aplicação:  
 • Transferência de buffers de áudio para DACs.  
 • Transferência de imagens de sensores para memória.  
 • Escrita de dados em cartões SD via SPI.  
 • Transmissão de pacotes via UART ou Ethernet.  
Essas aplicações reduzem drasticamente o uso da CPU e aumentam a eficiência do sistema.