

Tarefa Unidade 4 Capítulo 5

Nome: Matheus Alves do Nascimento

Exercício 1:

Elabore um programa para acionar um LED quando o botão A for pressionado 5 vezes, utilizando o temporizador como contador. Quando o valor da contagem atingir 5 vezes, um LED deve ser piscar por 10 segundos na frequência de 10 Hz.

Exercício 2:

Na questão anterior, implemente o botão B, para mudar a frequência do LED para 1 Hz.

Link do Wokwi: <https://wokwi.com/projects/421087757922612225>

Explicação: Foi feito em um código esse exercício e próximo. O funcionamento desse exercício representa essa parte do código. Primeiramente a respeito de como foi feito a parte do contador para o botão

Contador de 5 Vezes

```
if (gpio_get(PINO_BOTAO)) {  
    if (!botao_ja_pressionado) {  
        contagem_pressoes_botao++;  
        botao_ja_pressionado = true;  
    }  
} else {  
    botao_ja_pressionado = false;  
}  
  
if (contagem_pressoes_botao >= 5) {  
    contagem_pressoes_botao = 0;  
}
```

1. Verificar Pressionamento do Botão A:

- `gpio_get(PINO_BOTAO)` verifica se o botão A está pressionado.
- Se o botão A foi pressionado (`gpio_get(PINO_BOTAO)` retorna true) e não estava pressionado anteriormente (`!botao_ja_pressionado`), incrementa o contador `contagem_pressoes_botao` e define `botao_ja_pressionado` como true para evitar contagens múltiplas durante a mesma pressão.
- Se o botão A não está pressionado, `botao_ja_pressionado` é redefinido para false.

2. Verificar se o Botão A foi Pressionado 5 Vezes:

- Se contagem_pressoes_botao é maior ou igual a 5, o contador é resetado (contagem_pressoes_botao = 0), indicando que o botão A foi pressionado 5 vezes.

LED Piscando por 10 Segundos a 10 Hz

```
// Pisca o LED por 10 segundos a 10 Hz
for (int i = 0; i < 100; i++) {
  // Verifica se o botão B foi pressionado
  if (gpio_get(PINO_BOTAOB)) {
    if (!botaob_ja_pressionado) {
      Serial.println("Botão B pressionado! Frequência do LED: 1 Hz");
      botaob_ja_pressionado = true;
    }
    gpio_put(PINO_LED, 1);
    sleep_ms(500);
    gpio_put(PINO_LED, 0);
    sleep_ms(500);
  } else {
    botaob_ja_pressionado = false;
    gpio_put(PINO_LED, 1);
    sleep_ms(50);
    gpio_put(PINO_LED, 0);
    sleep_ms(50);
  }
}
```

1. Piscar o LED:

- Um loop for é executado 100 vezes ($i < 100$), o que resulta em 100 ciclos de piscamento.

2. Verificar Pressionamento do Botão B:

- Dentro do loop, verifica-se se o botão B está pressionado (gpio_get(PINO BOTAOB)).
- Se o botão B está pressionado e não estava pressionado anteriormente (!botaob_ja_pressionado), uma mensagem é impressa na comunicação serial indicando que o botão B foi pressionado e a frequência do LED muda para 1 Hz.
- O LED então pisca a uma frequência de 1 Hz (1 segundo ligado, 1 segundo desligado) usando sleep_ms(500) para criar um atraso de 500 milissegundos entre os estados ligado/desligado do LED.

3. Piscar o LED a 10 Hz:

- Se o botão B não está pressionado, botaob_ja_pressionado é redefinido para false.

- O LED pisca a uma frequência de 10 Hz (0.1 segundo ligado, 0.1 segundo desligado) usando `sleep_ms(50)` para criar um atraso de 50 milissegundos entre os estados ligado/desligado do LED.

Exercício 3:

Elabore um código utilizando as interfaces UART0 e conecte os fios TX e RX atribuídos à essa interface entre. Essa estrutura envia dados e recebe os dados na mesma interface, apenas para verificar seu funcionamento. Utilize a função `scanf` da biblioteca `stdio` para enviar via console um dado à placa, em seguida, transmita da UART0 para a UART1, e por fim, transmita o dado recebido para o console utilizando o `printf`.

Link do WOKWI: <https://wokwi.com/projects/421361408306550785>

Este código é um programa em C para o microcontrolador Raspberry Pi Pico que utiliza duas interfaces UART (UART0 e UART1) para comunicação serial. Aqui está um resumo do que o código faz:

- 1. Inclusão de bibliotecas:**
 - `stdio.h` para entrada e saída padrão.
 - `pico/stdlib.h` para inicialização padrão do Raspberry Pi Pico.
 - `hardware/uart.h` para utilizar funções de UART.
- 2. Definição dos pinos UART:**
 - Define os pinos GPIO para transmissão (TX) e recepção (RX) para as duas UARTs.
- 3. Inicialização do UART:**
 - Inicializa a entrada e saída padrão (`stdio`).
 - Inicializa UART0 e UART1 com uma taxa de transmissão de 9600 baud.
 - Configura os pinos GPIO para as funções de UART correspondentes.
- 4. Loop principal:**
 - Lê um caractere do console usando `scanf`.
 - Envia o caractere lido para UART0 usando `uart_putc`.
 - Imprime o caractere enviado no console.
 - Verifica se há dados disponíveis na UART0 usando `uart_is_readable`.
 - Se houver dados, lê o dado da UART1 usando `uart_getc`.
 - Imprime o dado recebido no console.

Este programa permite a comunicação serial entre duas interfaces UART, enviando e recebendo caracteres através do console e exibindo-os.

Exercício 4

Já para a comunicação I2C, iremos utilizar o DS1307, que é um Real Time Clock – RTC disponível no simulador Wokwi. O endereço I2C do DS1307 é 0x68. Um RTC é um hardware que garante a contagem de tempo na unidade de segundos. Muitos microcontroladores possuem RTC internos, mas alguns fazem uso de hardware externos. Para ler os valores, é necessário inicialmente configurar um valor de data e hora que deve, por exemplo, ser configurado manualmente pelo usuário. Nessa questão você deverá configurar o RTC para 24/09/2024 –13:27:00 e em seguida, realizar a leitura do mesmo a cada 5 segundos, e imprimir na tela do console (Serial USB) o valor lido. Na tabela a seguirão apresentados os principais endereços do RTC DS1307.

Link do WOKWI: <https://wokwi.com/projects/421372018445126657>

Este código é um programa em C para o microcontrolador Raspberry Pi Pico que utiliza a interface I2C para se comunicar com um módulo RTC (Relógio de Tempo Real) DS1307. Aqui está um resumo do que o código faz:

1. **Inclusão de Bibliotecas:**
 - stdio.h para entrada e saída padrão.
 - pico/stdlib.h para funções padrão do Raspberry Pi Pico.
 - pico/cyw43_arch.h e hardware/i2c.h para utilizar funções de I2C.
2. **Definição de IOs:**
 - Define a porta I2C a ser utilizada (i2c0).
 - Define o endereço I2C do módulo DS1307 (0x68).
3. **Funções de Leitura e Escrita I2C:**
 - i2c_write_byte: Escreve um byte de dado em um registro específico do DS1307.
 - i2c_read_byte: Lê um byte de dado de um registro específico do DS1307.
4. **Funções de Conversão:**
 - dec_to_bcd: Converte um valor decimal para BCD (Binary Coded Decimal).
 - bcd_to_dec: Converte um valor BCD para decimal.
5. **Função para Configurar Data e Hora:**
 - set_time: Escreve a data e hora no DS1307, convertendo os valores de decimal para BCD antes de escrever.
6. **Função para Ler Data e Hora:**
 - get_time: Lê a data e hora do DS1307 e converte os valores de BCD para decimal.
7. **Função Principal (main):**

- Inicializa a interface I2C e os pinos GPIO.
- Configura a data e hora no DS1307 (no exemplo, 24/09/2024 - 13:27:00).
- Entra em um loop infinito onde lê a data e hora do DS1307 a cada 5 segundos e imprime no console.

Exercicio 5 A e B:

Modifique o exemplo de código apresentado na videoaula (reproduzido abaixo) para controlar os LEDs RGB da placa BitDogLab usando o módulo PWM e interrupções, seguindo as orientações a seguir:

- a) O LED vermelho deve ser acionado com um PWM de 1kHz.
- b) O LED verde deve ser acionado com um PWM de 10kHz.

Link do WOKWI: <https://wokwi.com/projects/421723149083453441>

Aqui está um resumo do que o código faz:

1. Inclusão de Bibliotecas:

- stdio.h para entrada e saída padrão.
- pico/stdlib.h para funções padrão do Raspberry Pi Pico.
- hardware/pwm.h para utilizar funções de PWM.

2. Definição de Constantes e Variáveis:

- Define os pinos GPIO para os LEDs vermelho e verde.
- Define o período do PWM e o divisor de clock para obter as frequências desejadas (1kHz para o LED vermelho e 10kHz para o LED verde).
- Define o nível inicial do PWM (duty cycle) para os LEDs.

3. Função setup_pwm_red:

- Configura o pino do LED vermelho para a função PWM.
- Obtém o slice de PWM associado ao pino do LED vermelho.
- Define o divisor de clock do PWM para obter a frequência de 1kHz.
- Configura o valor máximo do contador (período do PWM) para o LED vermelho.
- Define o nível inicial do PWM para o LED vermelho.
- Habilita o PWM no slice correspondente para o LED vermelho.

4. Função setup_pwm_green:

- Configura o pino do LED verde para a função PWM.
- Obtém o slice de PWM associado ao pino do LED verde.
- Define o divisor de clock do PWM para obter a frequência de 10kHz.
- Configura o valor máximo do contador (período do PWM) para o LED verde.
- Define o nível inicial do PWM para o LED verde.
- Habilita o PWM no slice correspondente para o LED verde.

5. Função Principal (main):

- Inicializa o sistema padrão de entrada/saída.
- Chama as funções `setup_pwm_red` e `setup_pwm_green` para configurar os PWMs dos LEDs.
- Entra em um loop infinito (`while (true)`) para manter o programa em execução.

Exercício 5 c:

O Duty Cycle deve ser iniciado em 5% e atualizado a cada 2 segundos em incrementos de 5%. Quando atingir o valor máximo, deve retornar a 5% e continuar a incrementando. - Fazer isso para ambos os LEDs: azul e vermelho.

Link do WOKWI: <https://wokwi.com/projects/421742525850435585>

Aqui está um resumo do que o código faz:

1. Inclusão de Bibliotecas:

- `stdio.h` para entrada e saída padrão.
- `pico/stdlib.h` para funções padrão do Raspberry Pi Pico.
- `hardware/pwm.h` para utilizar funções de PWM.

2. Definição de Constantes e Variáveis:

- Define os pinos GPIO para os LEDs vermelho e azul.
- Define o passo de incremento/decremento do duty cycle do LED.
- Define o nível inicial do PWM (duty cycle) para os LEDs.
- Define o período do PWM e o divisor de clock para obter a frequência de 1kHz.

3. Funções `setup_pwm_red` e `setup_pwm_blue`:

- Configuram os pinos dos LEDs para a função PWM.
- Obtêm os slices de PWM associados aos pinos dos LEDs.
- Definem o divisor de clock do PWM para obter a frequência de 1kHz.
- Configuram o valor máximo do contador (período do PWM) para os LEDs.
- Definem o nível inicial do PWM para os LEDs.
- Habilitam o PWM nos slices correspondentes para os LEDs.

4. Função Principal (`main`):

- Inicializa o sistema padrão de entrada/saída.
- Chama as funções `setup_pwm_blue` e `setup_pwm_red` para configurar os PWMs dos LEDs.
- Entra em um loop infinito onde:
 - Define o nível atual do PWM (duty cycle) para os LEDs.
 - Espera um pequeno atraso (50 ms).
 - Incrementa ou decrementa o nível do LED baseado na variável `up_down`.
 - Muda a direção do incremento/decremento quando o nível atinge os valores máximos ou mínimos.

Exercício 5 d:

Tente controlar frequência e o Duty Cycle do LED azul de forma independente do que fez nos LEDs vermelho e verde. Você consegue? Por que não?

Resposta: Não, pois o PWM do RP2040 cada pino PWM está associado a um slice e cada slice possui dois canais (A e B), como estamos querendo mudar a frequência e o Duty cycle de três canais independentes isso não é possível pois só temos canais com slices A e B. Portanto, ao tentar controlar a frequência e o duty cycle de três LEDs (vermelho, azul e verde) de forma independente, enfrentamos uma limitação de hardware.

Exercício 6: Refaça o programa prático 01 presente no Ebook do Capítulo de ADC, mude a unidade de medida da temperatura de Celsius para Fahrenheit.

Link do WOKWI: <https://wokwi.com/projects/421910506807391233>

Explicação:

1. **Bibliotecas:**

- o `stdio.h` para entrada e saída.
- o `pico/stdlib.h` para funções básicas do Pico.
- o `hardware/adc.h` para controle do ADC.

2. **Definições:**

- o `ADC_TEMPERATURE_CHANNEL` define o canal ADC para o sensor de temperatura interno.

3. **Funções:**

- o `adc_to_temperature_celsius(uint16_t adc_value_c)`: Converte o valor ADC para graus Celsius.
- o `adc_to_temperature_fahrenheit(uint16_t adc_value_F)`: Converte o valor ADC para graus Fahrenheit.

4. **Função Principal (main):**

- o Inicializa o sistema e o ADC.
- o Seleciona o canal do sensor de temperatura.
- o Em um loop infinito, lê o valor do ADC, converte para Celsius e Fahrenheit, e imprime os valores.

Foi usado um valor padrão para o valor de ADC = 891 para fim de teste. No datasheet do RP2040 diz que quando o valor do adc já está em 891 a temperatura correta é de 20,1 celsius então foi predefinido um valor. E para a conversão de celsius para Fahrenheit na equação para cálculo do adc foi usado a proporção $((27\text{ }^{\circ}\text{C} \times 9/5) + 32 = 80,6\text{ }^{\circ}\text{F})$. Equação que foi fornecida pelo datasheet.