

PWM

Unidade 4 | Capítulo 7



Executores:



Coordenação:



Iniciativa:



Sumário

- Objetivos
- Revisão
- Visão Geral
- PWM no RP2040
- Configurações
- Exemplos de Código
- Principais Pontos
- Conclusão

Objetivos

- Entender o processo de modulação por largura de pulso e seu funcionamento.
- Configurar a utilizar os diferentes modos de configuração do módulo PWM.
- Aplicar o módulo PWM para controle de intensidade luminosa de LED.



Revisão

- Nas últimas aulas você aprendeu:
 - » Utilizar os pinos de entrada e saída
 - » Configurar o clocke temporizadores
 - » Utilizar interrupções para tratar eventos do periféricos



Visão Geral

PWM (Pulse Width Modulation)

- Técnica para controlar a potência entregue a dispositivos eletrônicos
- Permite variar a largura de pulso mantendo uma frequência constante
 - » Período
 - » Duty Cycle

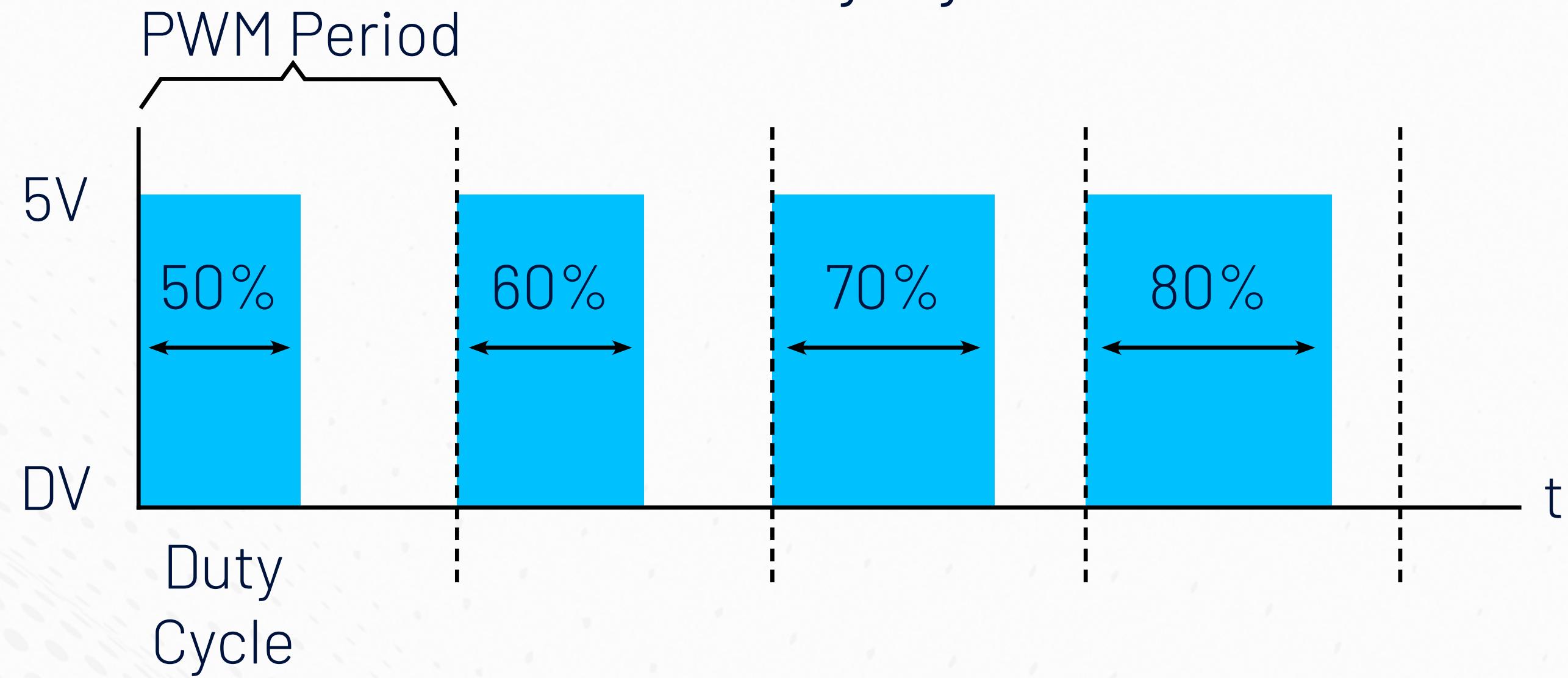
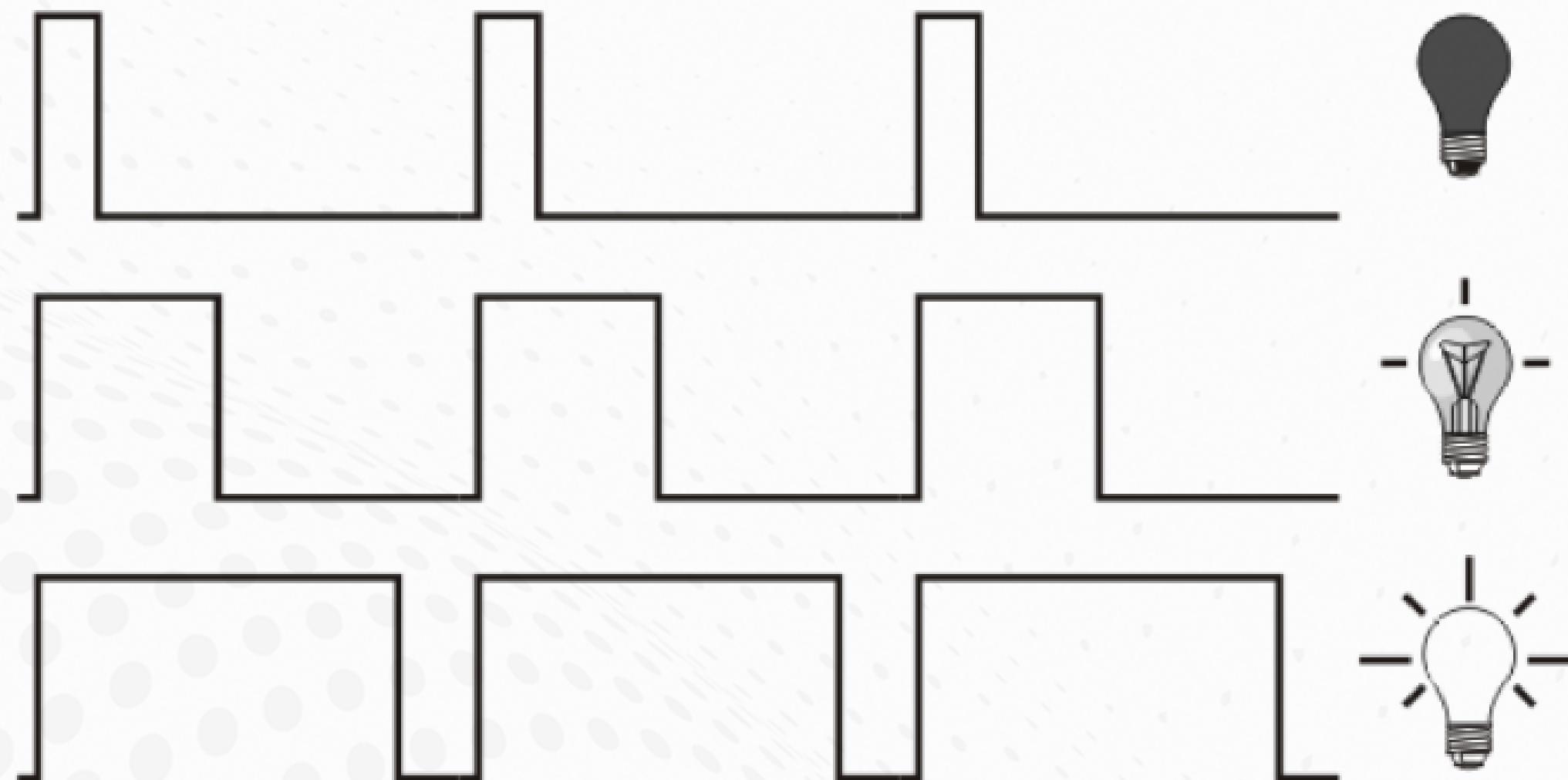


Fig.1
Fonte: imagem do autor.

Visão Geral

- Aplicações do PWM
 - » Controle do brilho de LEDs
 - » Geração de sinais de áudio
 - » Controle da velocidade motores
 - » Controle de conversores de potência
- Vantagens do PWM
 - » Fácil de implementar em microcontroladores
 - » Precisão no controle



PWM no RP2040

Bloco de PWM

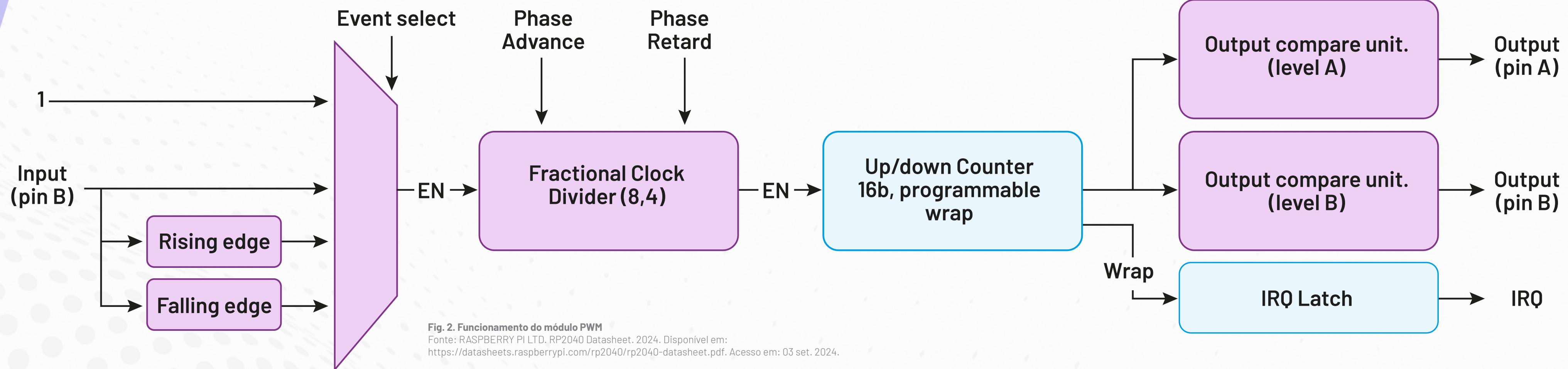
- 8 'slices' idênticos
 - » Cada um gera dois sinais de PWM
 - * Saída PWM
 - * Entrada
 - Medição de período ou ciclo ativo
- 16 saídas de PWM
 - » Qualquer GPIO pode ser uma saída PWM



PWM no RP2040

Características

- Contador de 16 bits
 - » Contagem crescente
 - * Contagem crescente/decrescente
 - * Divisor de clockfracional
 - 2 canais de saída independentes
 - » Detecção de eventos de borda
 - » Medição de período
 - Detecção de nível
 - » Medição de dutycycle
 - Configuração do wrap (período) e level (duty cycle)
 - Interrupção e DMA



A imagem apresenta um diagrama de blocos representando o funcionamento de um sistema digital que processa entradas e gera saídas. Aqui está uma descrição detalhada:
Na parte esquerda, há um bloco denominado "Input (pin B)", representando a entrada do sistema, que pode detectar a borda de subida ("Rising edge") ou borda de descida ("Falling edge") de um sinal. Essas bordas passam para um seletor de eventos ("Event Select"), que também recebe um sinal constante (representado pelo número 1). O seletor de eventos está conectado a um "Divisor de clock fracionado" ("Fractional Clock Divider"), que tem entradas para avançar a fase ("Phase Advance") ou retardar a fase ("Phase Retard"). O divisor de clock fracionado, por sua vez, está conectado a um contador de subida/descida programável de 16 bits ("Up/down Counter 16b, programmable wrap"), que também pode ser "envelopado" (wrap). O contador de subida/descida envia informações para três unidades: duas unidades de comparação de saída (uma para o nível A e outra para o nível B), que geram saídas correspondentes em pinos (pin A e pin B), e um bloco de interrupção (IRQ Latch), que gera uma interrupção (IRQ). Esse diagrama descreve o fluxo de sinal de entrada, passando por divisores de clock e contadores, até que as saídas sejam comparadas e enviadas, além de fornecer uma capacidade de gerar uma interrupção (IRQ).

PWM no RP2040

Todos os 30 pinos de GPIO
podem ser usados para PWM

GPIO	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
PMW Channel	0A	0A	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B	7A	7B
GPIO	16	17	18	19	20	21	22	23	24	25	26	27	28	29		
PWM Channel	0A	08	1A	1B	2A	2B	3A	3B	4A	4B	5A	5B	6A	6B		

Fig. 3. pinos de GPIO

Fonte: https://www.codrey.com/raspberry-pi/raspberry-pi-pico-pwm-primer/#google_vignette.

PWM no RP2040

Funcionamento do módulo PWM

- O circuito do PWM compara um contador livre de 16 bits com uma entrada
 - » Geração de um sinal de saída que fica alternando entre 0 e 1
 - * O tempo em alto é proporcional ao valor da entrada

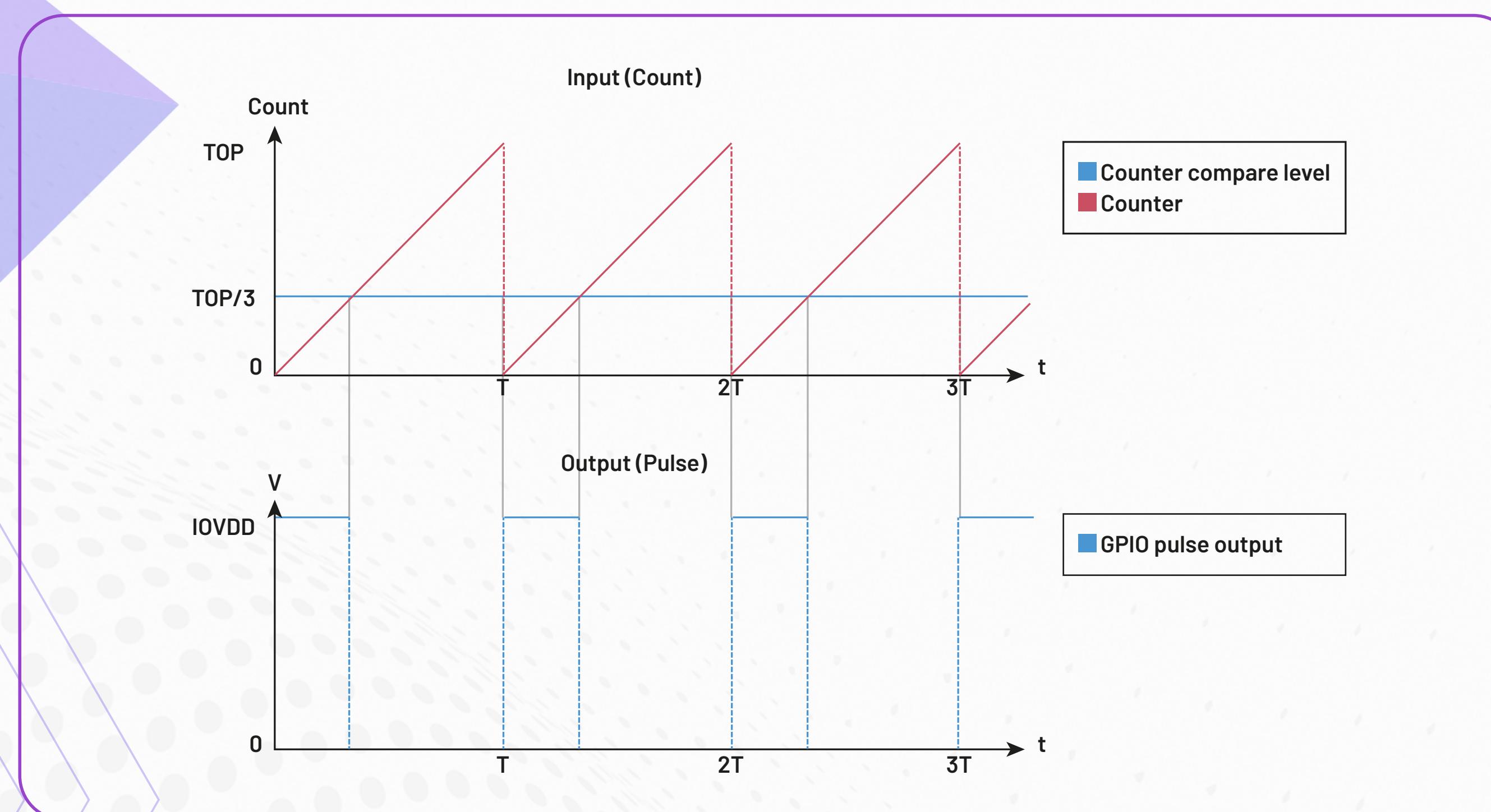


Fig. 4.

A imagem mostra dois gráficos que explicam como um contador controla a geração de pulsos:
Gráfico superior (Contador):
O contador aumenta e diminui em forma de triângulo, indo de 0 até um valor máximo ("TOP") e voltando para 0. Há uma linha horizontal azul em um terço do valor máximo ("TOP/3"). Quando o contador atinge esse ponto, ele aciona a geração de um pulso.
Gráfico inferior (Pulso de saída):
Cada vez que o contador chega a "TOP/3", um pulso é gerado, que sobe até um valor alto (IOVDD) e depois volta para 0. Esses pulsos acontecem regularmente no tempo (T , $2T$, $3T$), com um intervalo constante entre eles. Resumindo, o contador controla a saída de pulsos, que ocorrem sempre que ele atinge um terço de seu valor máximo. Esse diagrama descreve o fluxo de sinais de entrada, passando por divisores de clock e contadores, até que as saídas sejam comparadas e enviadas, além de fornecer uma capacidade de gerar uma interrupção (IRQ).

PWM no RP2040

Funcionamento do módulo PWM

- No modo de ajuste de fase, o contador é crescente/decrescente
 - » O centro do duty cycle é sempre o mesmo

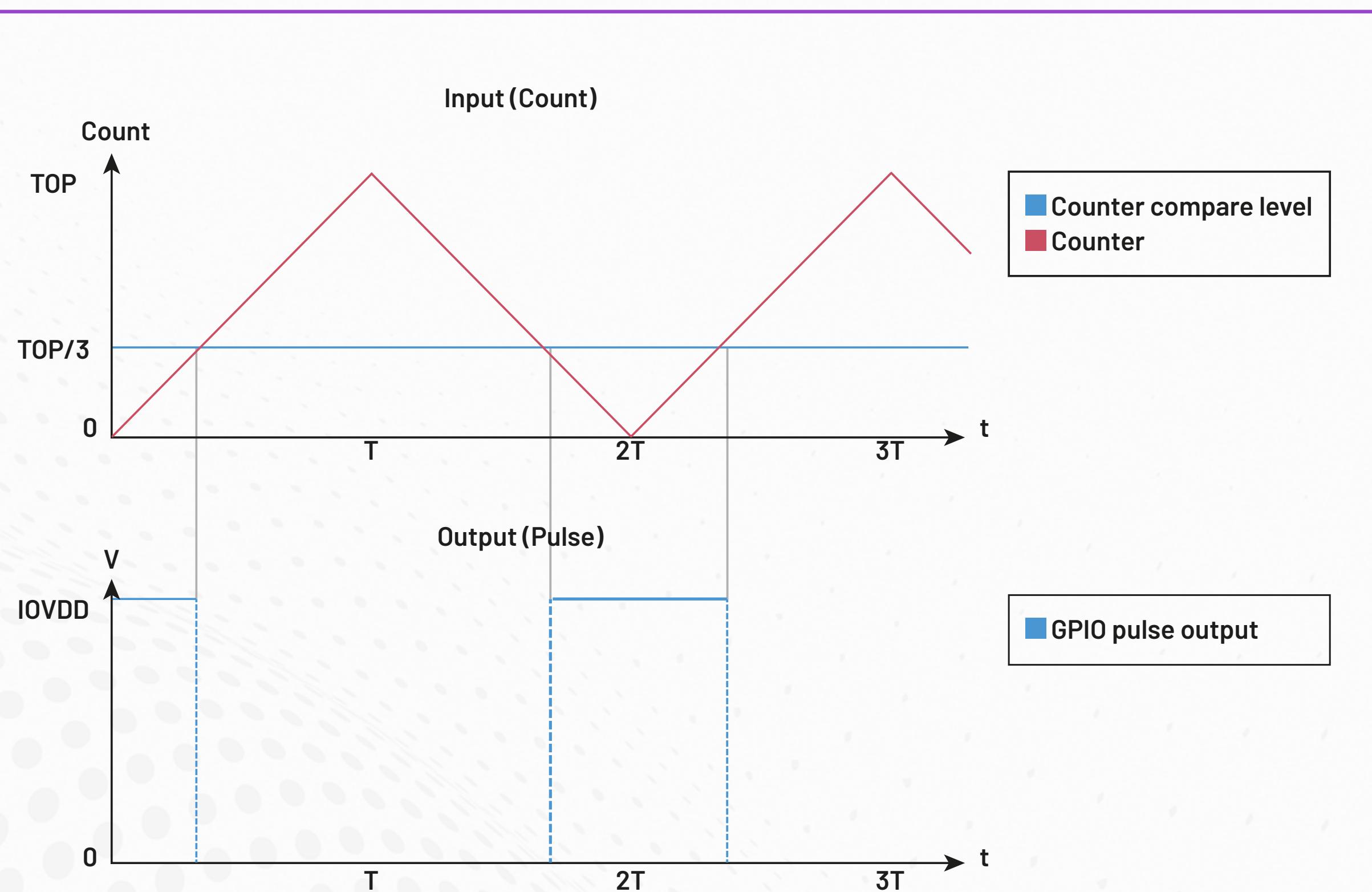


Fig. 5.

A imagem apresenta dois gráficos que ilustram o funcionamento de um contador digital.

Gráfico superior:
Eixo horizontal: Representa o tempo, marcado em unidades de "t" (por exemplo, segundos).

Eixo vertical: Representa o valor de uma contagem, com um limite superior denominado "TOP". Há uma linha horizontal que marca o valor "TOP/3".

Duas curvas:
Curva em vermelho: Representa o valor da contagem ao longo do tempo. Essa contagem aumenta e diminui de forma cíclica.

Linha azul: Representa um nível de comparação para a contagem. Quando a contagem em vermelho cruza essa linha, ocorre um evento no sistema.

Gráfico inferior:
Eixo horizontal: Mesma escala de tempo do gráfico superior.

Eixo vertical: Representa um sinal de saída digital, com dois níveis: alto (Vdd) e baixo (0).
Pulso: Cada vez que a contagem no gráfico superior cruza a linha de comparação azul, um pulso (sinal curto) é gerado no gráfico inferior.

Configurações

Etapas de Configuração usando o SDK

- Configuração do GPIO
- Configuração do Sinal de Saída
 - » Período
 - » Divisor de Clock
 - » Duty Cycle
- Habilitação da saída PWM
- Ajustando o Duty Cycle

Configurações

Configuração do GPIO

- `gpio_set_function(gpio, GPIO_FUNC_PWM)`
 - » `gpio`
 - * Número da porta que será usada
- `GPIO_FUNC_PWM`
 - » Constante que indica que o pino será usado como PWM
- `slice= pwm_gpio_to_slice_num(gpio)`
 - » `Slice`
 - * Endereço do slice para o pino gpio

Configurações

Configurações do sinal de saída

- A frequência do sinal de PWM é dada pela fórmula abaixo:

$$f_{PWM} = \frac{f_{clock}}{\left(divisor\ inteiro + \frac{divisor\ fracionário}{16} \right) \cdot wrap}$$

Onde :

f_{PWM} é a frequência do sinal de PWM.

f_{clock} é a frequência do sinal de clock base (125 Mhz).

divisor inteiro é a parte inteiro do divisor de clock, possui 8 bits.

divisor fracionário é a parte fracionária do divisor de clock, possui 4 bits.

wrap é o valor máximo de contagem, possui 16 bits.

Fig. 6.

A imagem apresenta uma fórmula matemática que calcula a frequência de um sinal PWM (Modulação por Largura de Pulso).

A fórmula é a seguinte:

$f_{PWM} = f_{clock} / (\text{divisor inteiro} + \text{divisor fracionário}/16) * \text{wrap}$

Onde:

f_{PWM} : Representa a frequência do sinal PWM que se deseja obter. É o resultado final do cálculo.

f_{clock} : É a frequência de um sinal de base, chamada de "clock", que serve como referência para o cálculo. No exemplo fornecido, f_{clock} vale 125 MHz.

divisor inteiro: É um número inteiro que contribui para dividir a frequência do sinal de clock. Ele tem uma representação binária de 8 bits.

divisor fracionário: É um número fracionário que também contribui para dividir a frequência do sinal de clock. Ele tem uma representação binária de 4 bits e é dividido por 16 dentro da fórmula para ajustar sua escala.

wrap: Representa o valor máximo de contagem de um contador interno. Ele tem uma representação binária de 16 bits e define o período do sinal PWM.

Configurações

Configurações do sinal de saída

- As funções abaixo são usadas para configurar o divisor do clocke o período (wrap):
 - » `pwm_set_clkdiv(slice, divider)`
 - * slice
 - Número do slicedo PWM
 - `divider`
 - » Número floatque indica o valor do divisor do clock de entrada
 - `pwm_set_wrap(slice, period)`
 - » Slice
 - * Número do slicedo PWM
 - `period`
 - » Valor máximo de contagem

Configurações

Configurações do sinal de saída

- As funções abaixo são usadas para configurar o dutycycle (level):
 - » `pwm_set_gpio_level(gpio, level)`
 - * `gpio`
 - Número da porta usada para PWM
- `level`
 - » Valor de contagem do duty cycle

Configurações

Configurações do sinal de saída

- Ajustando o Duty Cycle
 - » `pwm_set_gpio_level(gpio, level)`
 - * De forma periódica ou sob demanda
 - * Interrupção

Exemplos de Códigos

Controle do brilho de LED

- Vamos controlar a intensidade do LED azul da BitDogLab
 - » Configurar o gpio
 - » Configurar o PWM
 - * Frequência de 3,9 kHz
- Periodicamente (1Hz) ajustar o dutycycle
 - » De forma crescente/decrescente

Exemplos de Códigos

Controle do brilho de LED

- Após criar o projeto no VS Code
- Abra o arquivo CMakeLists.txt
 - » E faça a alteração abaixo, adicionando a biblioteca hardware_pwm

```
# Add the standard library to the build
target_link_libraries(PWM_LED_0
    pico_stdlib hardware_pwm)
```

» Vamos ver o nosso código principal!

Fig. 7. Exemplo de código.
Fonte: imagem do autor

A linha de código, provavelmente em C ou C++, adiciona bibliotecas ao projeto.
- Comentário: # Add the standard library to the build explica a função da linha, mas não é executado.
- Função Principal: target_link_libraries adiciona bibliotecas ao projeto.
- Bibliotecas Incluídas: (PWM_LED_0 pico_stdlib hardware_pwm);
 - PWM_LED_0: Biblioteca para controle de LED com PWM.
 - pico_stdlib: Biblioteca padrão para funcionalidades básicas (ex. Raspberry Pi Pico).
 - hardware_pwm: Biblioteca para controle de PWM em hardware.
A linha configura as bibliotecas necessárias para o projeto.

Exemplos de Códigos

Controle do brilho de LED

```
#include <stdio.h>
#include "pico/stlplib.h"
#include "hardware/pwm.h"

const uint LED = 12; // Pino do LED conectado
const uint16_t PERIOD = 2000; // Período do PWM (valor máximo do contador)
const float DIVIDER_PWM = 16.0; // Divisor fracional do clock para o PWM
const uint16_t LED_STEP = 100; // Passo de incremento/decremento para o duty cycle do LED
uint16_t led_level = 100; // Nível inicial do PWM (duty cycle)
```

Fig. 8. Exemplo de código.
Fonte: imagem do autor

O trecho de código em C, voltado para microcontroladores como o Raspberry Pi Pico, realiza as seguintes funções:
1- Inclusão de Bibliotecas: Adiciona stdio.h, pico_stlplib.h e hardware/pwm.h para entrada/saída padrão, interações básicas com o hardware do Pico e controle de PWM.

2- Definição de Constantes:

- LED: Número do pino do LED.
- PERIOD: Período do ciclo PWM.
- DIVIDER_PWM: Ajuste da frequência PWM.
- LED_STEP: Variação no nível de PWM.
- led_level: Nível inicial de brilho do LED.

O código configura o hardware e os parâmetros PWM para controlar o LED.

Exemplos de Códigos

Controle do brilho de LED

```
void setup_pwm()
{
    uint slice;
    gpio_set_function(LED, GPIO_FUNC_PWM); // Configura o pino do LED para função PWM
    slice = pwm_gpio_to_slice_num(LED); // Obtém o slice do PWM associado ao pino do LED
    pwm_set_clkdiv(slice, DIVIDER_PWM); // Define o divisor de clock do PWM
    pwm_set_wrap(slice, PERIOD); // Configura o valor máximo do contador (período do PWM)
    pwm_set_gpio_level(LED, led_level); // Define o nível inicial do PWM para o pino do LED
    pwm_set_enabled(slice, true); // Habilita o PWM no slice correspondente
}
```

Fig. 9. Exemplo de código.
Fonte: Imagem do autor

A imagem apresenta uma função em linguagem C chamada `setup_pwm()`. Essa função tem como objetivo configurar um pino de um microcontrolador para funcionar em modo PWM (Modulação por Largura de Pulso), permitindo o controle da intensidade de um LED, por exemplo. Vamos analisar o código linha a linha:

1. `void setup_pwm():`
 - void: Indica que a função não retorna nenhum valor.
 - `setup_pwm()`: É o nome da função, sugerindo que ela configura o PWM.
2. `uint slice;:` Declara uma variável inteira sem sinal (`uint`) chamada `slice`, que será utilizada para identificar um determinado canal PWM no microcontrolador.
3. `gpio_set_function(LED, GPIO_FUNC_PWM);:` Configura o pino do LED (identificado pela constante `LED`) para funcionar como saída PWM.
4. `slice = pwm_gpio_to_slice_num(LED);:` Associa o pino do LED a um canal PWM específico e armazena o número desse canal na variável `slice`.
5. `pwm_set_clkdiv(slice, DIVIDER_PWM);:` Define o divisor de clock para o canal PWM especificado. O divisor controla a frequência do sinal PWM.
6. `pwm_set_wrap(slice, PERIOD);:` Configura o valor máximo do contador do PWM, que determina o período do sinal PWM.
7. `pwm_set_gpio_level(LED, led_level);:` Define o nível inicial do sinal PWM no pino do LED, determinando o brilho inicial do LED.
8. `pwm_set_enabled(slice, true);:` Habilita o PWM no canal especificado.

Exemplos de Códigos

Controle do brilho de LED

```
int main()
{
    uint up_down = 1; // Variável para controlar se o nível do LED aumenta ou diminui
    stdio_init_all(); // Inicializa o sistema padrão de I/O
    setup_pwm(); // Configura o PWM
    while (true)
    {
        pwm_set_gpio_level(LED, led_level); // Define o nível atual do PWM (duty cycle)
        sleep_ms(1000); // Atraso de 1 segundo
        if (up_down)
        {
            led_level += LED_STEP; // Incrementa o nível do LED
            if (led_level >= PERIOD)
                up_down = 0; // Muda direção para diminuir quando atingir o período máximo
        }
        else
    }
}
```

Fig. 10. Exemplo de código.
Fonte: imagem do autor

A função main() em um programa C inicia a execução e controla o ciclo de brilho de um LED.
Declaração e Inicialização: Define a variável up_down para indicar se o brilho do LED aumenta ou diminui.
Configurações Iniciais: stdio_init_all() habilita funções de E/S e setup_pwm() configura o PWM para o LED.
Loop Infinito:

- Define o brilho do LED com pwm_set_gpio_level().
- Pausa o código por 1 segundo com sleep_ms(1000).
- Aumenta ou diminui led_level baseado em up_down, invertendo a direção ao atingir o máximo (PERIOD).
- A função ajusta continuamente o brilho do LED em um ciclo.

Exemplos de Códigos

Controle do brilho de LED

```
while (true)
{
    pwm_set_gpio_level(LED, led_level); // Define o nível atual do PWM (duty cycle)
    sleep_ms(1000); // Atraso de 1 segundo
    if (up_down)
    {
        led_level += LED_STEP; // Incrementa o nível do LED
        if (led_level >= PERIOD)
            up_down = 0; // Muda direção para diminuir quando atingir o período máximo
    }
    else
    {
        led_level -= LED_STEP; // Decrementa o nível do LED
        if (led_level <= LED_STEP)
            up_down = 1; // Muda direção para aumentar quando atingir o mínimo
    }
}
```

Fig. 11. Exemplo de código.

Fonte: imagem do autor

O código cria um loop infinito que ajusta o brilho de um LED de forma gradual e cíclica.

1. Loop Infinito (while(true)): Executa repetidamente.

2. Controle de Brilho (pwm_set_gpio_level(LED, led_level)): Define o nível de brilho do LED.

3. Pausa (sleep_ms(1000)): Aguarda 1 segundo antes de repetir.

4. Incremento ou Decremento:

- Se up_down for verdadeiro, aumenta o brilho (led_level += LED_STEP).

- Se led_level atinge o máximo (PERIOD), up_down é alterado para 0, iniciando a diminuição.

- Caso contrário, diminui o brilho (led_level -= LED_STEP).

5. Reversão no Mínimo: Ao atingir o valor mínimo (LED_STEP), up_down volta a 1, reiniciando o aumento.
O código faz o brilho oscilar entre valores máximo e mínimo.

Principais Pontos

- PWM é um dispositivo que nos permite controlar a potência entregue a dispositivos eletrônicos
- Existem várias aplicações de PWM, como: controle de brilho de led, motores, conversores de potência, dentre outras
- O RP2040 pode gerar 16 sinais de PWM
- O SDK da Raspberry nos fornece funções simples para configurar e usar o PWM

PWM

