UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

CURSO DE GRADUAÇÃO EM **ENGENHARIA DE COMPUTAÇÃO**
SÉTIMO SEMESTRE

QXD0143 - MICROCONTROLADORES

Relatório 05: Modulação por Largura de Pulso (PWM) usando o Temporizador TPM

QUIXADÁ - CE
2022

471047 — PEDRO HENRIQUE **MAGALHÃES BOTELHO**

475664 — DAVID MACHADO **COUTO BEZERRA**

Relatório 05: Modulação por Largura de Pulso (PWM) usando o Temporizador TPM

Orientador: Prof. Dr. Thiago Werlley Bandeira da Silva

Quinto relatório escrito para a disciplina de Microcontroladores, no curso de graduação em Engenharia de Computação, pela Universidade Federal do Ceará (UFC), campus em Quixadá.

QUIXADÁ - CE
2022

Conteúdo

Lista de Figuras

1 Introdução

Esse é o quinto relatório da disciplina de **Microcontroladores**, ministrada pelo professor Thiago W. Bandeira. Nesse relatório é discutido sobre a utilização de sinais **PWM** (*Pulse-width Modulation*, ou Modulação por Largura de Pulso) para regular o fornecimento de potência a uma carga, utilizando o periférico **TPM** (*Timer/PWM Module*). Com isso podemos aumentar ou diminuir o brilho de um LED, ou controlar o eixo de um motor, fornecendo uma potência específica.

Nas práticas realizadas em laboratório foi utilizada a placa de desenvolvimento **Freedom FRDM-KL25Z**, da Kinetis/NXP, com o microcontrolador **MKL25Z128VLK4**.

Este documento contém informações e imagens retiradas do manual de referência do microcontrolador usado, como mostra as figura 01:

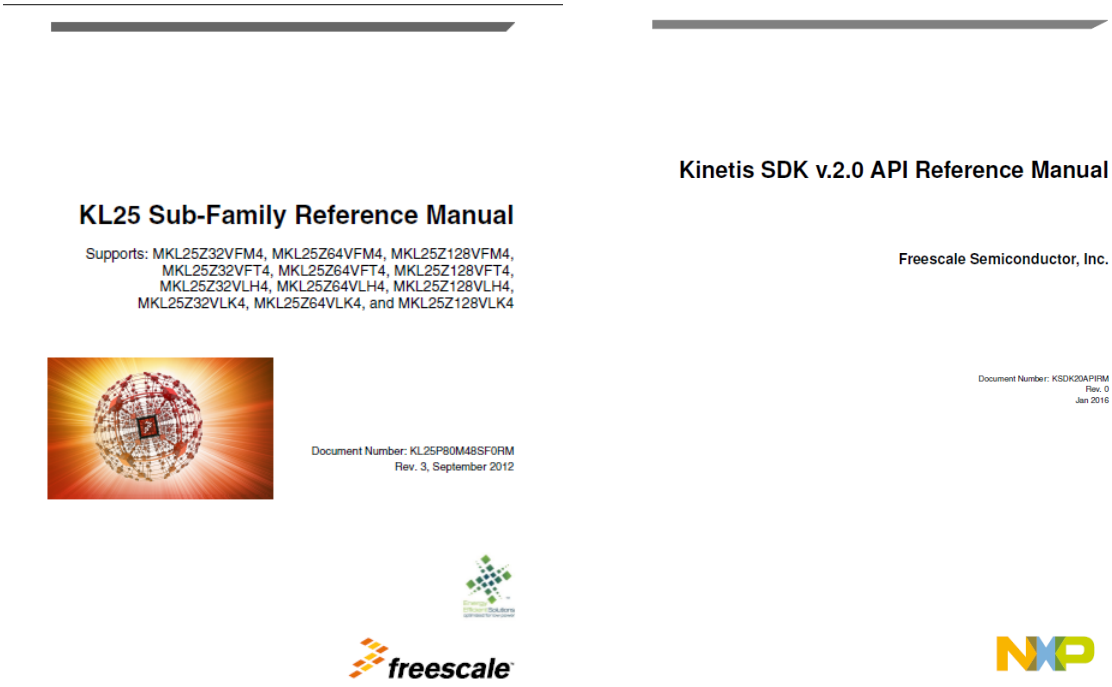

Figura 1: Capas dos Manuais de Referência do MKL25Z128VLK4 e do SDK.

Também foi utilizado o manual de referência do *Software Development Kit* da Kinetis, o **SDK**, como mostra a figura 01. O SDK é uma API que fornece diversas ferramentas para a programação de microcontroladores da Kinetis/NXP.

Os projetos da disciplina estão disponíveis em um repositório no Github, em [1], onde estão os relatórios, códigos-fonte e esquemáticos (os projetos completos encontram-se comprimidos em arquivos .zip). O link também se encontra nas referências.

2 Modulação por Largura de Pulso (PWM)

PWM, do inglês *Pulse-Width Modulation*, é uma técnica utilizada para controlar a potência fornecida à uma carga através de um sinal digital. Variando a largura dos pulsos de uma onda digital "pulsante" (com níveis altos e baixos separados por um intervalo) varia-se a potência fornecida à carga. A figura 02 mostra como um sinal PWM se comporta ao variarmos a largura do pulso do sinal.

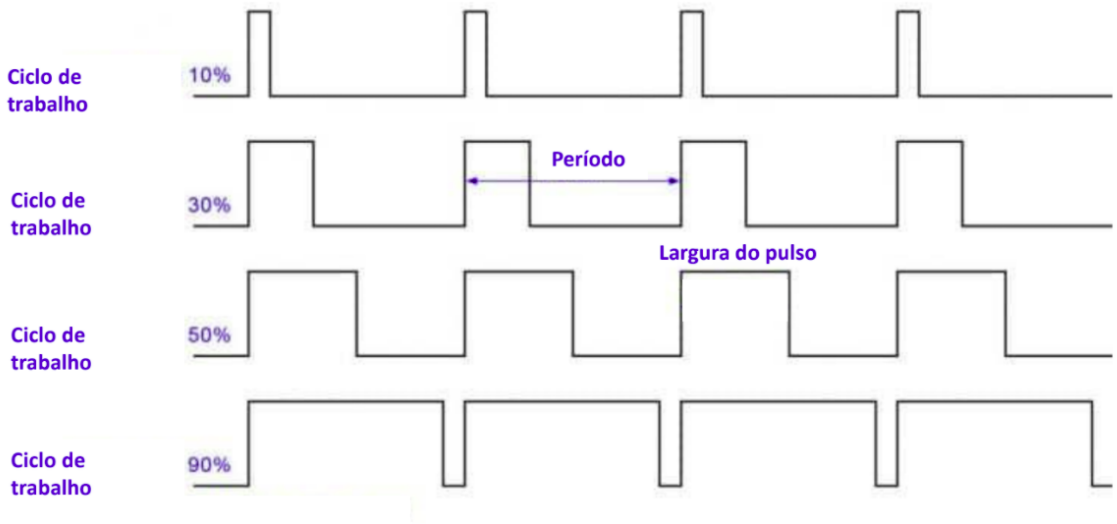

Figura 2: Característica de um Sinal PWM

Podemos, por exemplo, usar um sinal PWM para variar a potência fornecida à um LED, aumentando ou diminuindo a intensidade de sua luz, emitindo o sinal ao LED pelo pino do microcontrolador. Temos algumas variáveis em um sinal PWM:

- **Pulso**: Momento em que um sinal encontra-se em nível lógico alto.
- **Sinal Pulsante**: Sinal que estabelece um pulso em intervalos bem definidos, realizando uma oscilação.
- **Largura do Pulso**: Tempo em que o sinal está em nível lógico alto.
- **Período**: Tempo necessário para a onda realizar um ciclo. Também pode ser visto como o tempo entre duas bordas de subida do sinal.
- **Ciclo de Trabalho**: Quantidade de tempo do período que o pulso permanece em nível lógico alto, medido em porcentagem. Podemos calcular essa porcentagem com a seguinte fórmula:

$$\text{Ciclo de Trabalho} = (\text{Largura do Pulso} \times 100)/\text{Período}$$

3 Geração de PWM usando o Temporizador TPM

Podemos gerar sinais PWM utilizando o periférico temporizador TPM (Timer/PWM Module). Cada módulo TPM possui um conjunto de canais, onde cada canal controla um pino, que pode ser utilizado para emitir um sinal PWM. Devemos, para isso, selecionar a função do pino para **TPMx_CHn**, onde x denota o número do módulo e n o número do canal. Podemos ver essa função na tabela de multiplexação do pino, no capítulo 10 do manual do processador. É possível ver essa relação na figura 03.

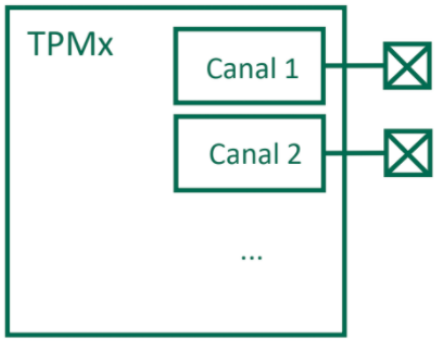

Figura 3: Periférico TPM usado para geração de sinal PWM

Devemos, para gerar um sinal PWM em um pino, habilitar o *clock* do módulo TPM com o qual o pino pode trabalhar. Podemos fazer isso no registrador SCGC6, nos bits 24, 25 e 26, para habilitar os módulos TPM0, TPM1 e TPM2, respectivamente. Devemos também habilitar o *clock* do registrador PORTx referente ao pino, para que possamos configurar o registrador PCR e selecionar a opção de PWM. Por exemplo, o pino PTC1 pode gerar sinal PWM. Para configurá-lo para isso basta que selecionemos a função ALT4 do pino, colocando valor 0b100 nos bits 10, 9 e 8, respectivamente, do PCR do pino.

Devemos então configurar o módulo TPM, de maneira semelhante ao que fizemos anteriormente. Devemos selecionar qual *clock* será emitido ao periférico no registrador SOPT2, do módulo SIM, como mostra a figura 04.

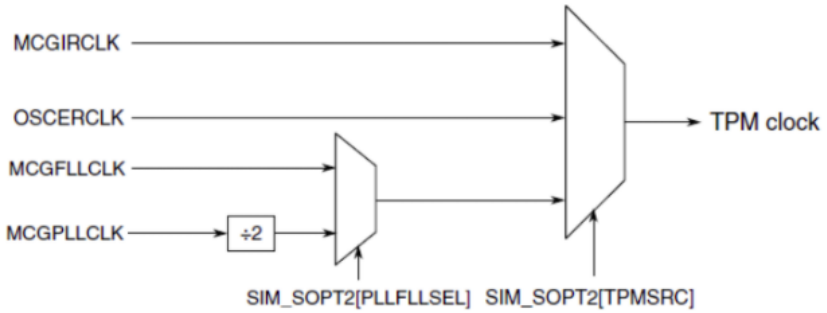

Figura 4: Controle do *clock* repassado ao periférico TPM

Se quisermos selecionar o *clock* de 20,971520 MHz (MCGFLLCLK) para o TPM devemos fazer a seguinte configuração:

```
/* Select MCGFLLCLK clock */
SIM->SOPT2 &= ~(1 << 16);
/* Choose MCGFLLCLK clock or MCGPLLCLK/2 for TPM */
SIM->SOPT2 = (0b01 << 24);
```

Devemos então configurar o registrador de Controle e Status (SC) do módulo TPM para escolhermos o modo de contagem (bit CPWMS, sendo 0 alinhado à borda e 1 alinhado ao centro) e o *prescaler* (campo de bits PS) e, ao final da configuração, habilitar o PWM (no bit CMOD, com valor 0b01). Podemos ver o formato do registrador SC na figura 05.

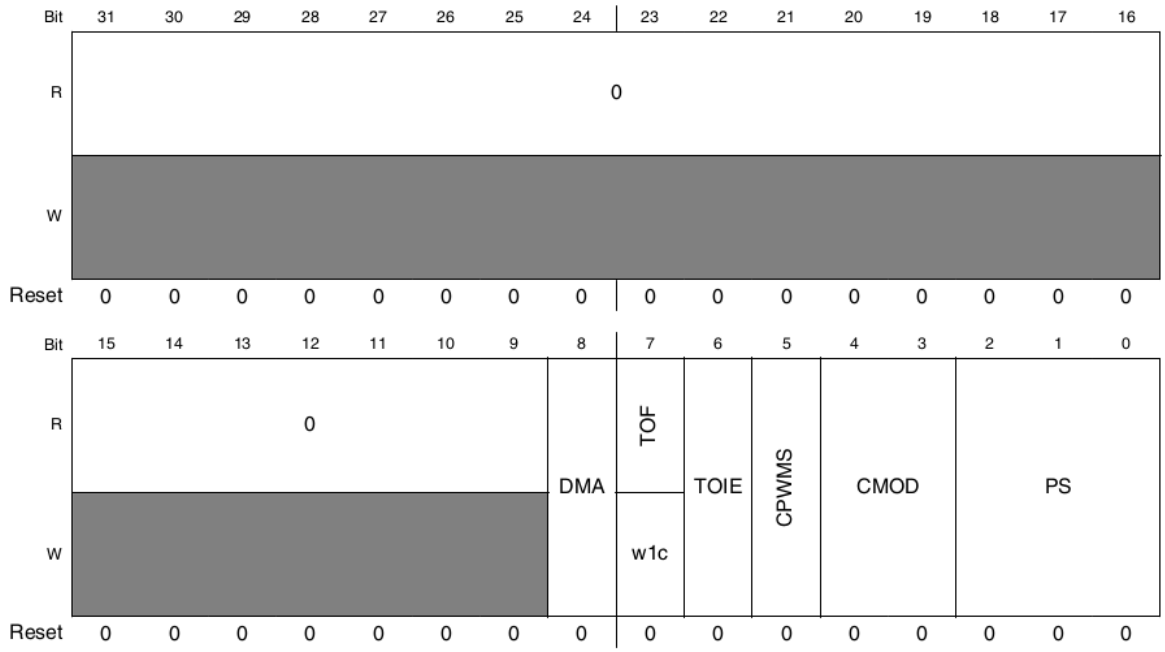

Figura 5: Registrador de Controle do Temporizador TPM

Levando em conta a configuração de *prescaler* e o *clock* definido para o periférico podemos configurar o registrador **MOD**, que irá definir o período da onda, usando a fórmula:

$$\text{MOD} = T_{overflow} \times \frac{f_{source}}{ps}$$

Devemos, agora, configurar o registrador de comparação e o registrador de controle e *status* do canal, bem como o registrador de comparação. Todos os canais de um módulo compartilham o mesmo registrador contador, tendo registradores de comparação (CnV) e configuração (CnSC) individuais.

O registrador de comparação do canal tem de 16-bits, e mantém o valor de contagem que determina o ciclo de trabalho. Ou seja, o valor em CnV é uma porcentagem do valor em MOD, sendo MOD o valor que representa o período total. Se o ciclo de trabalho for de 100%,

por exemplo CnV será igual a MOD. O valor de CnV pode ser modificado em tempo real, não precisando parar e reiniciar o temporizador.

O registrador CnSC é mostrado na figura 06:

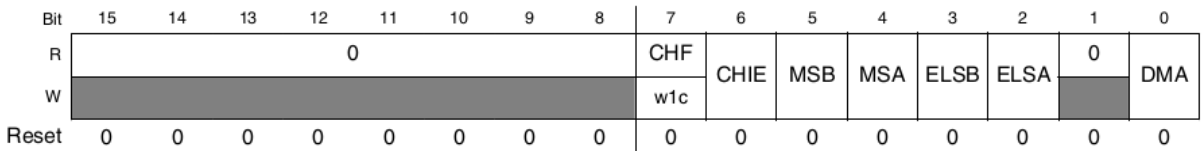

Figura 6: Registrador de Controle e *Status* do Canal do TPM

Devemos configurar quatro bits importantes desse registrador para configurarmos o sinal PWM: os bits **MSnB:MSnA** (Seleção do Modo do Canal) e **ELSnB:ELSnA** (Seleção de Borda ou Nível), que definem como o sinal PWM será manejado. Esses detalhes podem ser vistos na figura 07:

CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mode	Configuration
X	00	00	None	Channel disabled
X	01/10/11	00	Software compare	Pin not used for LPTPM
0	00	01	Input capture	Capture on Rising Edge Only
		10		Capture on Falling Edge Only
		11		Capture on Rising or Falling Edge
	01	01	Output compare	Toggle Output on match
		10		Clear Output on match
		11		Set Output on match
	10	10	Edge-aligned PWM	High-true pulses (clear Output on match, set Output on reload)
		X1		Low-true pulses (set Output on match, clear Output on reload)
	11	10	Output compare	Pulse Output low on match
		X1		Pulse Output high on match
1	10	10	Center-aligned PWM	High-true pulses (clear Output on match-up, set Output on match-down)
		X1		Low-true pulses (set Output on match-up, clear Output on match-down)

Figura 7: Valores para Configuração do Registrador CnSC

Por exemplo, para configurarmos um sinal PWM alinhado à borda com pulsos *high true*, ou seja, que o pulso é referente a um nível lógico baixo, devemos configurar o bit CPWMS do registrador SC como 0, e devemos colocar 0b10 em MSnB:MSnA e ELSnB:ELSnA.

Devemos nos atentar a se o pulso é verdadeiro em alto ou baixo. Por exemplo, para os LEDs internos da placa, que são conectados em ânodo comum, o ciclo de trabalho deve estar configurado para *low true*. Para um LED externo em cátodo comum devemos ter a configuração de *high true*. A relação entre *low true* e *high true* é mostrada na figura 08. De forma intuitiva, se o sinal está em nível lógico alto 75% do período, ele está em nível lógico baixo 25% do período. Utilizaremos a modalidade de *high true*.

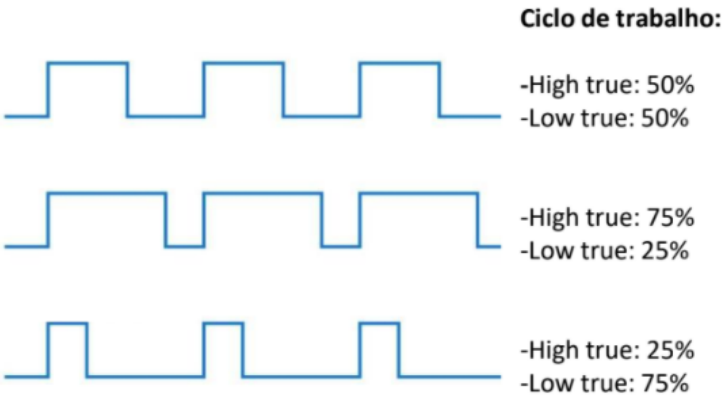

Figura 8: Configuração do Ciclo de Trabalho para Verdadeiro em Alto ou Baixo

Devemos também nos atentar ao alinhamento do sinal PWM. Caso a contagem esteja alinhada à borda (borda da verificação), como mostra a figura 09, o TPM apenas irá contar até o valor de MOD e gerará um *overflow*, reiniciando a contagem de zero. Essa contagem é chamada também de "contagem crescente". Utilizaremos essa modalidade.

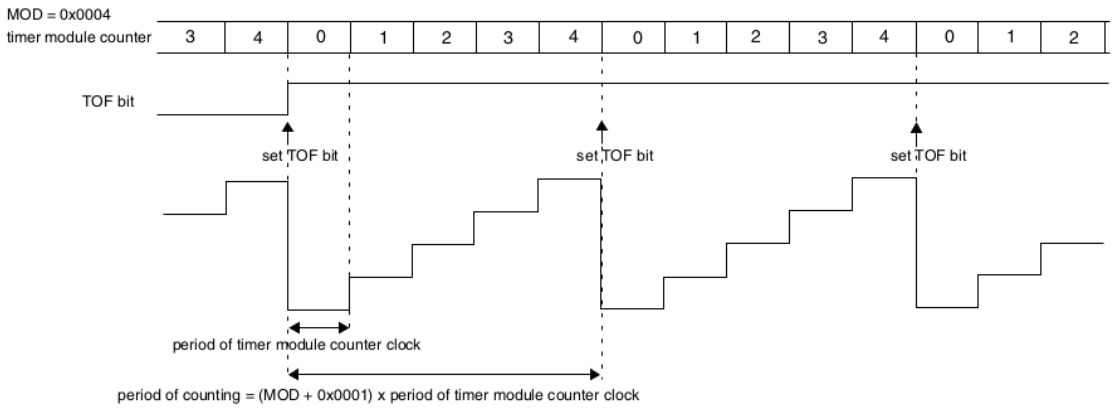

Figura 9: Sinal PWM alinhado à borda

Caso a contagem esteja alinhada ao centro, como mostra a figura 10, a contagem inicia-se com o valor de MOD e decrementa até zero, incrementando novamente para gerar um *overflow*, reiniciando o ciclo. Essa contagem é chamada também de "contagem crescente-decrescente".

Ressaltando que para que ocorra esse tipo de contagem devemos ativar o bit CPWMS no registrador SC.

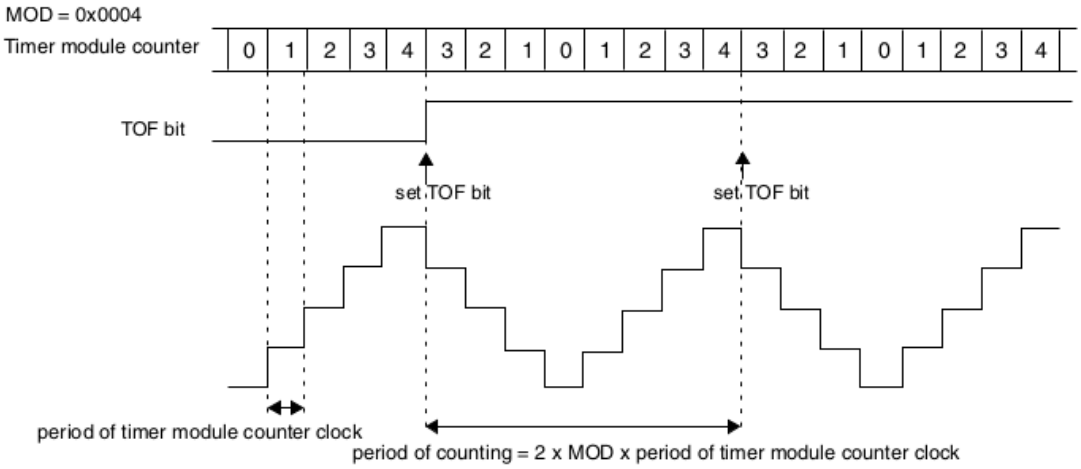

Figura 10: Sinal PWM alinhado ao centro

Com essas informações em mãos podemos configurar o periférico TPM para gerar sinais PWM pelos pinos, utilizando-se de canais dos módulos. No geral, utilizamos o sinal PWM para dois propósitos principais: controlar a potência fornecida à uma carga, como uma lâmpada, e controlar o ciclo de trabalho de um sinal emitido à um motor, controlando então a posição do eixo do motor.

4 Controlando o Ciclo de Trabalho do PWM

Abaixo está o código que configura o ciclo de trabalho, o *duty cycle*, de dois sinais PWM para 10% e 80%, nos pinos PTA12(canal 0 do TPM1) e PTB2(canal 0 do TPM2). Dessa forma esses pinos irão gerar um sinal PWM com um pulso de largura especificada nos respectivos registradores **CnV** dos canais.

```
#include "MKL25Z4.h"

/**
 * Inicializa o Canal 0 do modulo TPM1 em PTA12.
 */
void initTpm1Ch0(void) {

  /* Habilitando o clock de PORTA */
  SIM->SCGC5 |= (1 << 9);

  /* Seleciona ALT3: TPM1_CH0 */
  PORTA->PCR[12] |= (0b011 << 8);

  /* Habilita o clock de TPM1 */
  SIM->SCGC6 |= (1 << 25);

  // MOD = (20971520/128)*0,02 = 3277
  TPM1->MOD = 3276;

  /* Seleciona Modo de Contagem Crescente (Padrao) e Prescale de 128 */
  TPM1->SC |= (0b111 << 0);

  /* Desativar o PWM no modulo TPM1_CH0 (PTA12) */
  TPM1->CONTROLS[0].CnSC = 0;

  /* Ativar o PWM no modulo TPM1_CH0 (PTA12) alinhado a borda com pulsos high
    true */
  TPM1->CONTROLS[0].CnSC |= (0b10 << 4) | (0b10 << 2);

  /* Inicia a Contagem */
  TPM1->SC |= (0b01 << 3);
}

/**
 * Inicializa o Canal 0 do modulo TPM2 em PTB2.
 */
void initTpm2Ch0(void) {

  /* Habilitando o clock de PORTB */
  SIM->SCGC5 |= (1 << 10);

  /* Seleciona ALT3: TPM2_CH0 */
  PORTB->PCR[2] |= (0b011 << 8);

  /* Habilita o clock de TPM2 */
```

```
45    SIM->SCGC6 |= (1 << 26);
46
47    // MOD = (20971520/128)*0,02 = 3277
48    TPM2->MOD = 3276;
49
50    /* Seleciona Modo de Contagem Crescente (Padrao) e Prescale de 128 */
51    TPM2->SC |= (0b111 << 0);
52
53    /* Desativar o PWM no modulo TPM2_CH0 (PTB2) */
54    TPM2->CONTROLS[0].CnSC = 0;
55
56    /* Ativar o PWM no modulo TPM2_CH0 (PTB2) alinhado a borda com pulsos high
        true */
57    TPM2->CONTROLS[0].CnSC |= (0b10 << 4) | (0b10 << 2);
58
59    /* Inicia a Contagem */
60    TPM2->SC |= (0b01 << 3);
61  }
62
63
64  /**
65   * Inicializa o PWM usado no programa.
66   */
67  void initPwm(void){
68
69    /* Seleciona o clock MCGFLLCLK */
70    SIM->SOPT2 &= ~(1 << 16) & ~(0b11 << 24);
71
72    /* Seleciona "MCGFLLCLK clock or MCGPLLCLK/2" */
73    SIM->SOPT2 |= (0b01 << 24);
74
75    initTpm1Ch0();
76
77    initTpm2Ch0();
78
79  }
80
81
82  int main(void) {
83
84    initPwm();
85
86    /* Ciclo de Trabalho de 80%: 2620 (80% de 3276) */
87    TPM2->CONTROLS[0].CnV = 2620;
88
89    /* Ciclo de Trabalho de 10%: 327 (10% de 3276) */
90    TPM1->CONTROLS[0].CnV = 327;
91
92    while(1);
93  }
```

4.1 Característica do Sinal PWM em um Osciloscópio

Vejamos agora como o sinal PWM se comporta, utilizando um osciloscópio. Conectamos cada um dos dois pinos utilizados para emitir um sinal PWM a uma das ponteiras do osciloscópio para que pudéssemos ver como o sinal se comporta, isto é, a relação entre a largura do período e a largura do pulso de ambos os sinais.

Na figura 11 temos o sinal PWM configurado anteriormente, para gerar um *duty cycle* de 10%. Isto é, a largura do pulso tem 10% da largura do período. Sendo o período 20 ms, o pulso tem uma largura de 2 ms.

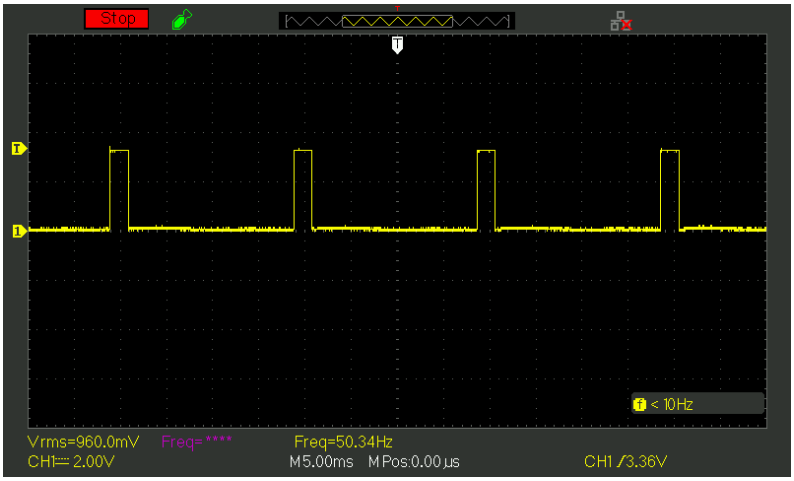

Figura 11: Sinal PWM com *duty cycle* de 10%

Já na figura 12 temos o sinal PWM configurado para ter um *duty cycle* de 80%, com um período de 20 ms. Dessa forma, seu pulso tem uma largura de 16 ms.

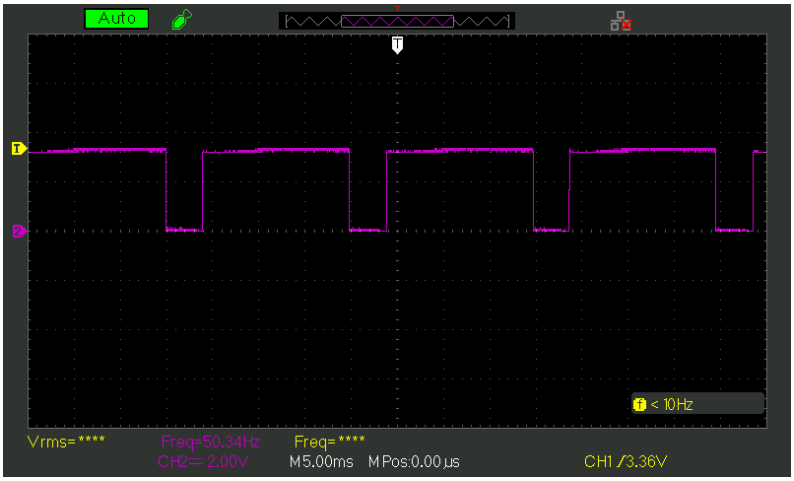

Figura 12: Sinal PWM com *duty cycle* de 80%

Na figura 13 temos, então, os dois sinais acima sobrepostos. Nesse figura podemos ver a diferença da largura dos pulsos dos sinais, mesmo com um período igual. Vemos então a importância de podermos controlar a largura do pulso, podendo controlar a quantidade de potência fornecida a uma carga, já que com um ciclo de trabalho de 100% temos a potência total sendo fornecida. Um sinal PWM com uma potência menor que isso deve ser capaz de fornecer menos potência ainda usando um sinal digital. Chamamos isso de **controle digital**.

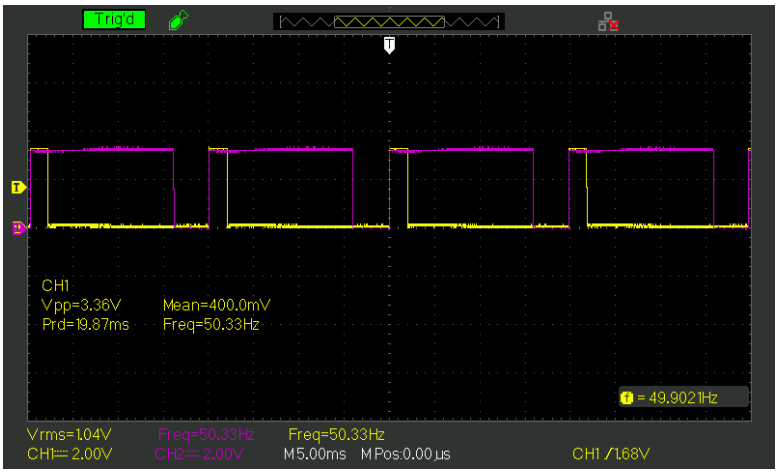

Figura 13: Os dois sinais PWM anteriores sobrepostos

Na figura 14 temos um exemplo de aplicação de sinais PWM para controle de potência, onde é possível controlar a luminosidade de uma lâmpada com um sinal digital (0V ou 220V):

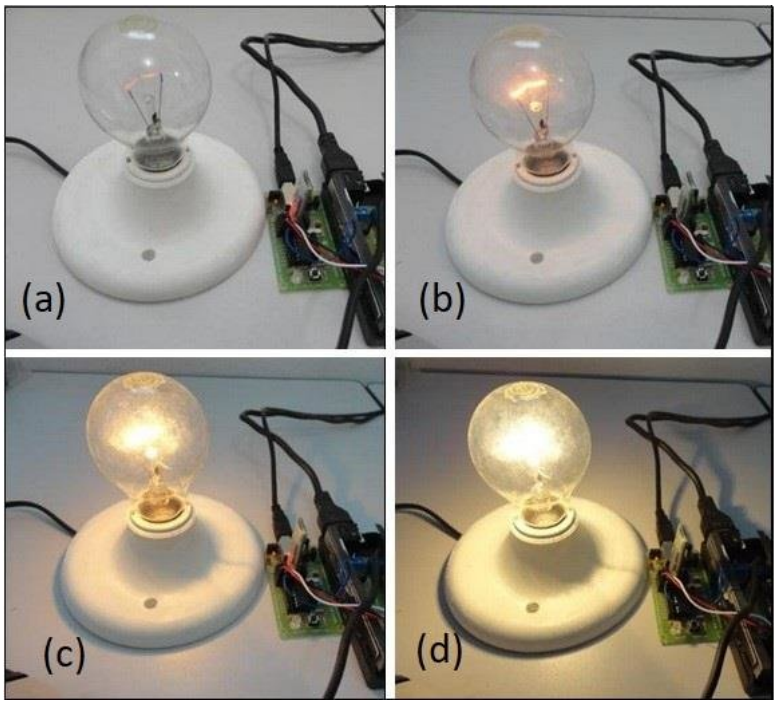

Figura 14: Controle da luminosidade de uma lâmpada usando sinal PWM

5 Controlando a Rotação de um Servo Motor

Iremos agora configurar um sinal PWM para realizar um giro de 180º em um servo motor, como o mostrado na figura 15. O motor servo estará conectado ao pino PTE20, onde emitiremos um sinal PWM para realizar o controle da rotação do eixo do motor: de 0° para 180°, e de 180° par 0°, indefinidamente.

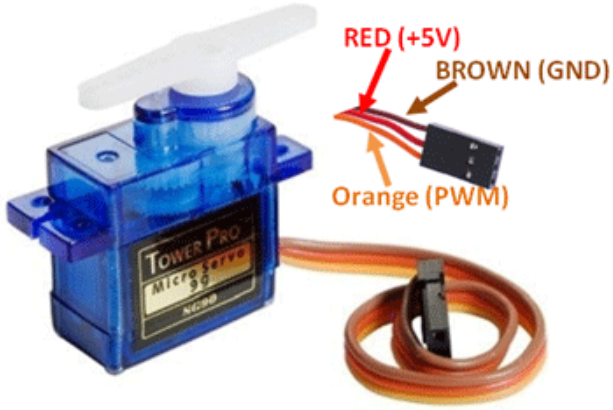

Figura 15: Servo Motor SG90 utilizado

5.1 Funcionamento do Servo Motor

Servo Motor é um dispositivo eletromecânico utilizado para movimentar, com precisão, um objeto, permitindo-o girar em ângulos ou distâncias específicas, com garantia do posicionamento e garantia da velocidade.

O servo motor trabalha com servo-mecanismo que usa o *feedback* de posição para controlar a velocidade e a posição final do motor. Internamente, um servo motor combina um motor com um circuito de realimentação, um controlador e outros circuitos complementares. Ele usa um codificador ou sensor de velocidade (*encoder*) que tem a função de fornecer o *feedback* de velocidade e posição.

Podemos configurar a posição do eixo do motor de acordo com o *duty cycle*, para um período total de 20 ms, necessário para fazer o eixo do motor rotacionar:

- Ângulo de 180°: Duração do pulso de 2,5 ms.
- Ângulo de 90°: Duração do pulso de 1,5 ms.
- Ângulo de 0°: Duração do pulso de 0,5 ms.

Na figura 16 está o mapeamento das larguras de pulso do sinal para os ângulos realizados pelo eixo do motor:

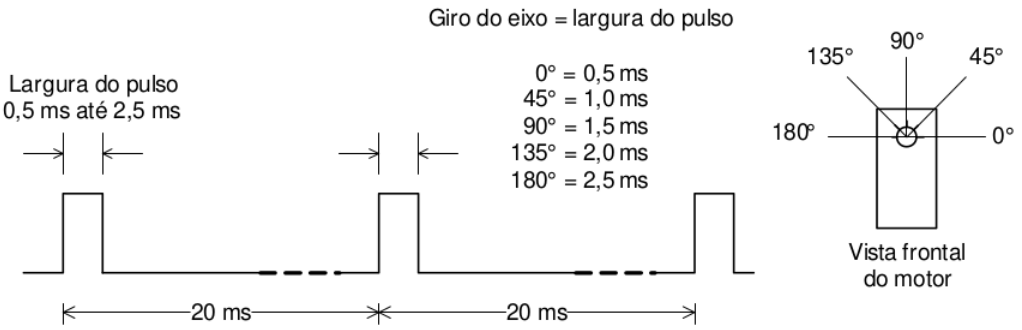

Figura 16: Servo Motor SG90 utilizado

Abaixo está o cálculo para obter o CnV para o giro que se deseja realizar no servo motor:

$$\text{CnV} = (\text{MOD} \times \text{Largura do Pulso})/\text{Período}$$

5.2 Passos para a Configuração do PWM

Podemos quebrar a logica do código em dois passos, configuração do PWM e controle do motor através do sinal de PWM.

O primeiro e feito por meio de duas funções, chamadas de **initPwm** e **initTpm1Ch0**, onde a primeira função é responsável por configurar o *clock* para o TPM de forma geral, selecionando o *clock* que vai ser usado, que nesse caso é o **MCGFLLCLK**, e por fim chama **initTpmCh0** para realizar configuração do canal 1 de TPM. Primeiro é inicializado o *clock* para o PORTE e configurando o pino 20 como TPM1_CH0, depois inicializado o *clock* para o TPM1 e selecionando o MOD com o valor de 3277, referente a um período de 20 ms.

No registrador SC temos como selecionado o modo padrão que é o de contagem crescente e um *prescale* de 128, desativa o PWM para realizar as configurações e ativa o PWM no pino 20 do PORTE e configura os pulsos do PWM como *high true*. Por fim, inicia a contagem.

Na *main* é aonde ocorre o controle do motor servo, por meio de um loop infinito que altera o *duty cycle* para controle, definindo uma variável booleana que informa quando se pode aumentar o *duty cycle* do PWM. Se passar do valor máximo ele configura para diminuir na próxima instância do *loop*. No final pegamos a variável PWM e atribuímos ao CnV para realizar a movimentação do eixo.

5.3 Código para Controle do Giro de 180°

Abaixo está o código para realizar o giro do motor em 180°, seguindo os passos para configuração mencionados:

```
#include "MKL25Z4.h"

#include <stdbool.h>

#define MIN_SERVO_VALUE 82
#define MAX_SERVO_VALUE 410

/**
 * Inicializa o Canal 0 do modulo TPM1 em PTE20.
 */
void initTpm1Ch0(void) {

  /* Habilitando o clock de PORTA */
  SIM->SCGC5 |= (1 << 13);

  /* Seleciona ALT3: TPM1_CH0 */
  PORTE->PCR[20] |= (0b011 << 8);

  /* Habilita o clock de TPM1 */
  SIM->SCGC6 |= (1 << 25);

  // MOD = (20971520/128)*0,02 = 3277
  TPM1->MOD = 3276;

  /* Seleciona Modo de Contagem Crescente (Padrao) e Prescale de 128 */
  TPM1->SC |= (0b111 << 0);

  /* Desativar o PWM no modulo TPM1_CH0 (PTA12) */
  TPM1->CONTROLS[0].CnSC = 0;

  /* Ativar o PWM no modulo TPM1_CH0 (PTA12) alinhado a borda com pulsos high
    true */
  TPM1->CONTROLS[0].CnSC |= (0b10 << 4) | (0b10 << 2);

  /* Inicia a Contagem */
  TPM1->SC |= (0b01 << 3);
}


/**
 * Inicializa o PWM usado no programa.
 */
void initPwm(void){

  /* Seleciona o clock MCGFLLCLK */
  SIM->SOPT2 &= ~(1 << 16) & ~(0b11 << 24);

```

```
47    /* Seleciona "MCGFLLCLK clock or MCGPLLCLK/2" */
48    SIM->SOPT2 |= (0b01 << 24);
49
50    initTpm1Ch0();
51
52  }
53
54
55  int main(void) {
56
57    initPwm();
58
59    TPM1->CONTROLS[0].CnV = MIN_SERVO_VALUE;
60
61    bool aumentar_duty_cycle = true;
62
63    /* Uso de uma variavel auxiliar para impedir overflow */
64    int pwm = MIN_SERVO_VALUE;
65
66    while(1) {
67      /* AUMENTAR DUTY CYLCE */
68      if(aumentar_duty_cycle) {
69        pwm += 10;
70        if(pwm > MAX_SERVO_VALUE) {
71          pwm = MAX_SERVO_VALUE;
72          aumentar_duty_cycle = false;
73        }
74      }
75      /* DIMINUIR DUTY CYLCE */
76      else {
77        pwm -= 10;
78        if(pwm < MIN_SERVO_VALUE) {
79          pwm = MIN_SERVO_VALUE;
80          aumentar_duty_cycle = true;
81        }
82      }
83
84      /* Atribui o valor calculado ao CnV, movendo o eixo */
85      TPM1->CONTROLS[0].CnV = pwm;
86    }
87  }
```

6 Controlando a Rotação de um Motor de Passos

Iremos agora configurar um sinal PWM para realizar um giro de 360^{o} em um motor de passos, como o mostrado na figura 16. O motor servo estará conectado ao pino PTE20, onde emitiremos um sinal PWM para realizar o controle da rotação do eixo do motor: de 0° para 360°, e de 360° para 0°, indefinidamente.

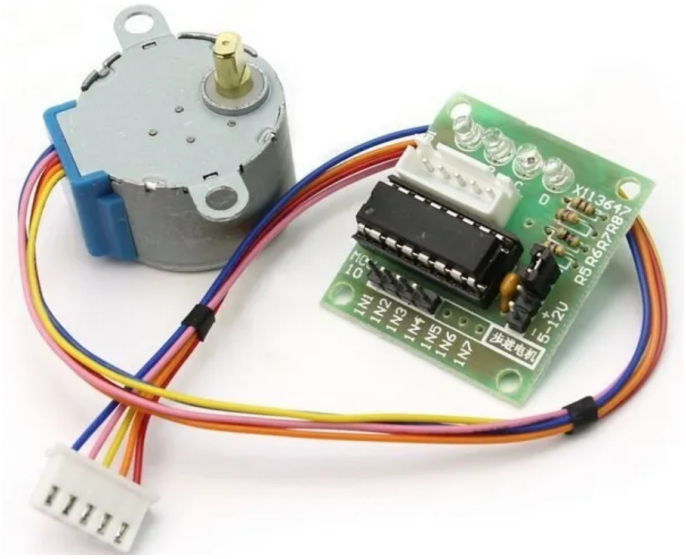

Figura 17: Motor de Passos 28BYJ-48 utilizado

É possível ver que o motor utilizado (à esquerda) vem acompanhado de um circuito (à direita). Este circuito é chamado de ***driver***. O *driver* converte os sinais de pulso vindos do pino, ou seja, o sinal PWM, em um movimento do motor, atuando nas bobinas, para conseguir o posicionamento requerido.

6.1 Funcionamento do Motor de Passos

Um motor de passo é um dispositivo amplamente, utilizado que traduz pulsos elétricos em movimento mecânico. Em aplicações como impressoras matriciais e robótica, o motor de passo é usado para controle de posição. Os motores de passo geralmente têm um rotor magnético (também chamado de eixo) cercado por um estator, como está evidenciado na figura 17, onde temos 4 bobinas e um rotor magnético.

O motor, então, funciona através do controle individual de bobinas que, quando ativadas, atraem um dos polos do ímã conectado ao eixo do motor e provocam a rotação do motor.

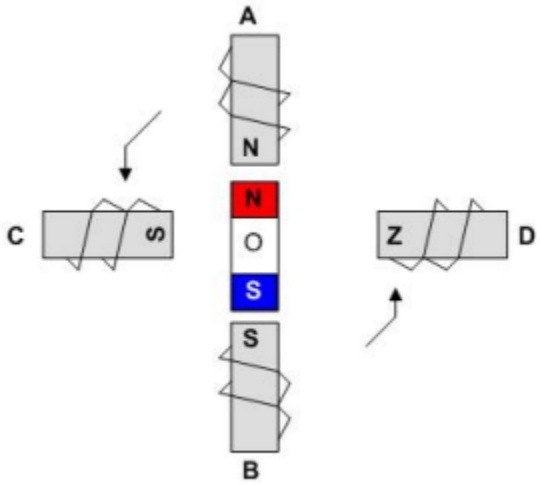

Figura 18: Funcionamento do Motor de Passos

6.2 Passos

Os passos feitos em software para realizar o controle do motor de passos a partir de um PWM, pode ser quebrado em duas etapas, em que a primeira é a configuração dos registradores do TPM para preparar o ambiente do PWM e segundo a logica que vai ser feita pra controle do motor.

O primeiro passo é definido na função **initPwm**, na qual vai ser feitas a preparação do canal para o sinal PWM. Vamos definir primeiro o **MCGFLLCLK**, configurando SIM_SOPT2 com 0 no bit 16, e escolhendo-o com 01 no bit 24 e 25.

Após a configuração do *clock* do TPM é chamado a função **initTpm1Ch0**, que irá configurar o canal 0 do TPM1. Primeiro configurando o pino 0 do PORTB como canal 1 do módulo TPM1, utilizando o registrador PCR e o campo MUX, que vem especificado no chip.

Após isso ativamos o TPM1, setando em 1 o bit 25 do registrador SCGC6(já tendo sido definimdo anteriormente qual clock vai ser passado para o TPM).

Com isso, podemos definir o MOD por meio de:

$$\text{MOD} = T_{overflow} \times \frac{f_{source}}{ps}$$

$$\text{MOD} = 20ms \times \frac{20{,}971520MHz}{128} \approx 3277$$

Com isso, MOD = 3276. Agora configuramos o registrador SC, definindo um modo de contagem crescente, onde a contagem incrementa a cada pulso de *clock*, com um *prescale* de 128. Note que o PWM foi desativado antes da configuração e depois ativado, na linha 39, com o início da contagem.

Após feita a configuração a função *main* chega ao *loop* infinito, onde temos dois laços **for** internos para controlar o giro na direção y e de y para x. No primeiro **for** vamos do valor MIN_STEPPER_VALUE até o MAX_STEPPER_VALUE, onde toda interação altera o *duty cycle*, realizando um giro, atribuindo o valor i que está sempre incrementando. No segundo **for** parte do MAX indo para MIN e realizando a mesma operação, desfazendo o giro.

6.3 Código para Controle do Giro de 360°

Abaixo está o código que realiza o giro de 360° usando motor de passos. Perceba o tempo entre as mudanças do CnV. É um tempo para que o motor acomode a mudança e que fique mais fácil visualizar o giro.

```
#include "MKL25Z4.h"

#define MIN_STEPPER_VALUE 58
#define MAX_STEPPER_VALUE 327

void delay_ms(int ms){
  for(int i = 0; i < ms; i++){
    for(int j = 0; j < 7000; j++);
  }
}

/**
 * Inicializa o Canal 0 do modulo TPM1 em PTB0.
 */
void initTpm1Ch0(void) {

  /* Habilitando o clock de PORTB */
  SIM->SCGC5 |= (1 << 10);

  /* Seleciona ALT3: TPM1_CH0 */
  PORTE->PCR[20] |= (0b011 << 8);

  /* Habilita o clock de TPM1 */
  SIM->SCGC6 |= (1 << 25);

  // MOD = (20971520/128)*0,02 = 3277
  TPM1->MOD = 3276;

  /* Seleciona Modo de Contagem Crescente (Padrao) e Prescale de 128 */
  TPM1->SC |= (0b111 << 0);

  /* Desativar o PWM no modulo TPM1_CH0 (PTA12) */
  TPM1->CONTROLS[0].CnSC = 0;

  /* Ativar o PWM no modulo TPM1_CH0 (PTA12) alinhado a borda com pulsos high
    true */
  TPM1->CONTROLS[0].CnSC |= (0b10 << 4) | (0b10 << 2);
```

```
37
38    /* Inicia a Contagem */
39    TPM1->SC |= (0b01 << 3);
40  }
41
42  /**
43   * Inicializa o PWM usado no programa.
44   */
45  void initPwm(void){
46
47    /* Seleciona o clock MCGFLLCLK */
48    SIM->SOPT2 &= ~(1 << 16) & ~(0b11 << 24);
49
50    /* Seleciona "MCGFLLCLK clock or MCGPLLCLK/2" */
51    SIM->SOPT2 |= (0b01 << 24);
52
53    initTpm1Ch0();
54
55  }
56
57  int main(void) {
58
59    initPwm();
60
61    while(1) {
62
63      // Controla o giro do motor da direcao para y
64      for(int i = MIN_STEPPER_VALUE; i < MAX_STEPPER_VALUE; ++i){
65        TPM1->CONTROLS[0].CnV = i;
66        delay_ms(5);
67      }
68
69      // Controla o giro do motor da direcao y para x
70      for(int i = MAX_STEPPER_VALUE; i >= MIN_STEPPER_VALUE; --i){
71        TPM1->CONTROLS[0].CnV = i;
72        delay_ms(5);
73      }
74    }
75  }
```

7 Conclusão

Com isso concluímos o relatório, na qual foi abordado o tema de sinais PWM e como utilizá-los para realizar diferentes tarefas, como controlar motores(servo e de passos) e controlar o fornecimento de potência à uma carga.

Tivemos uma ideia de como utilizar o módulo TPM para gerar sinais usando a modulação por largura de pulso e tivemos noções de aplicações e da importância na utilização dos mesmos.

Ao final da prática obtivemos conhecimentos em usar o sinal PWM para o controle de motores e pudemos reforçar o conhecimento adquirido nas aulas teóricas.

Referências

[1] Repositório da disciplina "microcontroller_practices". https://github.com/pedrobotelho15/microcontroller_practices. Accessed: 2022-04-06.