UNIVERSIDADE
FEDERAL DO CEARÁ
CAMPUS QUIXADÁ

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

QXD0143 - MICROCONTROLADORES

Relatório 06: Convertendo Leituras de Sensores Analógicos usando Conversor ADC

QUIXADÁ - CE
2022

471047 — PEDRO HENRIQUE **MAGALHÃES BOTELHO**

427602 — ERICK **CORREIA SILVA**

Relatório 06: Convertendo Leituras de Sensores Analógicos usando Conversor ADC

Orientador: Prof. Dr. Thiago Werlley Bandeira da Silva

Sexto 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 sexto relatório da disciplina de **Microcontroladores**, ministrada pelo professor Thiago W. Bandeira. Nesse relatório é discutido sobre a utilização de conversores analógico-digital para converter leituras de sensores analógicos em valores processáveis por um sistema digital, como é o caso dos microcontroladores. Utilizaremos então o periférico de ADC da placa da disciplina para obter sinais analógicos do pino para poderem ser utilizados no programa.

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:

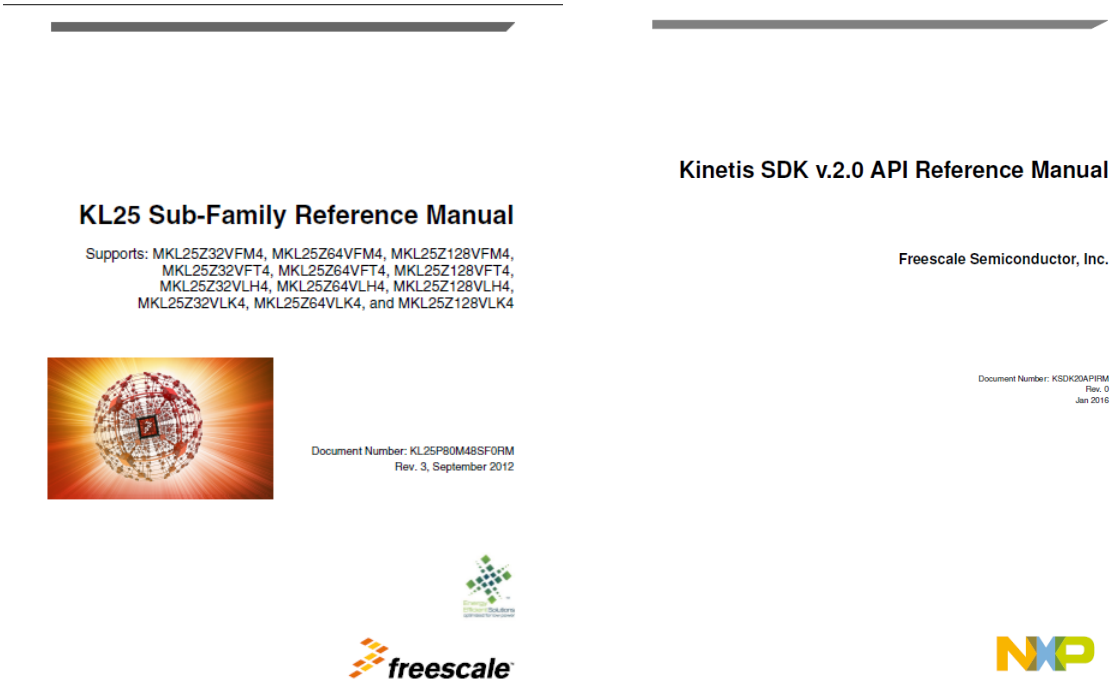

Figura 1: Capas dos Manuais de Referência do MKL25Z128VLK4 e do SDK.

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 Conversor Analógico-Digital (ADC)

Os conversores analógico-digitais estão entre os dispositivos mais utilizados para aquisição de dados. Os computadores digitais usam valores binários (discretos), mas no mundo físico tudo é analógico (contínuo).

Temperatura, pressão (vento ou líquido), umidade e velocidade são alguns exemplos de grandezas físicas com as quais lidamos todos os dias.

2.1 Sensores Analógicos

Uma quantidade física é convertida em sinais elétricos (tensão, corrente) usando um dispositivo chamado transdutor. Os transdutores usados para gerar saídas elétricas também são chamados de sensores. Sensores de temperatura, velocidade, pressão, luz e muitas outras quantidades físicas naturais produzem uma saída que é voltagem (ou corrente).

Portanto, precisamos de um conversor analógico-digital para traduzir os sinais analógicos em números digitais para que o microcontrolador possa ler e processar os números. Veja na figura 02 como é esquematizado essa ideia.

Figura 2: Conexão do Microcontrolador ao Sensor via ADC

2.2 Resolução do Conversor

Podemos definir a resolução como a faixa de valores de entrada possíveis que o sensor pode discernir. Portanto, a resolução também é uma medida de precisão.

O ADC tem resolução de n bits, onde n pode ser 8, 10, 12, 16 ou até 24 bits. ADCs de resolução mais alta fornecem um tamanho de etapa menor, onde o tamanho da etapa é a menor alteração que pode ser discernida por um ADC.

Algumas resoluções amplamente utilizadas para ADCs são mostradas na tabela abaixo. Embora a resolução de um ADC seja decidida no momento de seu projeto e não possa ser alterada, podemos controlar o tamanho do passo, ou etapa, com a ajuda do que é chamado de V_{ref} . Isso é discutido abaixo.

Resolução	Número de Etapas	Tamanho da Etapa
8	256	5V/256 = 19.53 mV
10	1024	5V/1024 = 4.88 mV
12	4096	5V/4096 = 1.2 mV
16	65536	5V/65,536 = 0.076 mV

Na tabela acima foi considerada uma tensão V_{ref} de 5V para os cálculos.

2.3 Tensão de Referência V_{ref}

Na figura 03 temos a representação do diagrama de blocos do ADC, onde o ADC recebe um sinal de entrada analógica, em V_{IN}, que será convertida em 8 sinais digitais, D0 a D7 (para um ADC com resolução de 8-bits). O ADC ainda recebe uma tensão de referência V_{ref} que servirá como ponto de partida para os cálculos, e um sinal que irá iniciar a conversão.

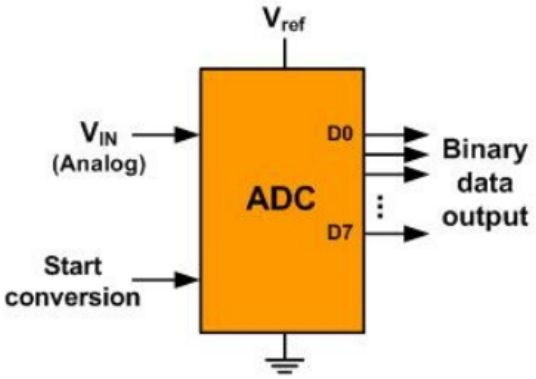

Figura 3: Representação de um ADC de 8-bit

V_{ref} é uma tensão de entrada usada como tensão de referência. A tensão conectada a este pino, juntamente com a resolução do ADC, determinam o tamanho do passo. Para um ADC de 8 bits, o tamanho do passo é $V_{ref} \div 256$, porque é um ADC de 8 bits, e 2 elevado a 8 nos dá 256 passos. Consulte a tabela acima. Temos, então, que a fórmula geral é:

$$\text{Tamanho do Passo} = V_{ref} \div 2^n, \text{ onde } n \text{ é a resolução do ADC.}$$

Por exemplo, se a faixa de entrada analógica precisar ser de 0 V a 3,3 V, V_{ref} será conectado a 3,3 V. Isso dá $3,3V \div 256 = 12,89$ mV para o tamanho do passo de um ADC de 8 bits. Em outro caso, se precisarmos de um tamanho de passo de 10 mV para um ADC de 8 bits, então V_{ref} = 2,56 V, porque 2,56 V / 256 = 10 mV. Para o ADC de 10 bits, se V_{ref} = 5V, o tamanho do passo é 4,88 mV, e por aí vai.

2.4 Resultado da Conversão

Em um ADC de 8 bits temos uma saída digital de 8 bits, com uma faixa de D0 a D7, enquanto que no ADC de 10 bits a saída de dados é D0–D9. Para calcular a tensão de saída, usamos a seguinte fórmula:

$$D_{OUT} = V_{IN} \div \text{Tamanho do Passo}$$

D_{OUT} é então n sinais digitais carregando a conversão do sinal analógico V_{IN} em sinal digital. Por exemplo. para um ADC de 8-bit, se V_{IN} for 2,1 V, V_{ref} for 2,56 V, temos que:

$$\text{Tamanho do Passo} = 2,56V \div 2^8 = 10mV$$

$$D_{OUT} = 2,1V \div 10 \text{ mV} = 210, \text{ em decimal.}$$

210 em binário é 0b10101011, que é os valores de saída para os pinos D7-D0. Portando, ao chegar 2,1 V no pino V_{IN} do ADC, o *chip* irá emitir 0b10101011 em D7-D0.

2.5 Canais de Entrada Analógica

Muitas aplicações de aquisição de dados precisam de mais de uma entrada analógica para ADC. Por esse motivo, os periféricos de ADC vem com 2, 4, 8 ou até 16 canais em um único chip.

A multiplexação de entradas analógicas é amplamente utilizada como mostrado no ADC848 e MAX1112. Nesses chips, temos 8 canais de entradas analógicas, permitindo monitorar várias grandezas como temperatura, pressão, vazão e assim por diante. Atualmente, alguns chips de microcontroladores ARM vêm com ADC de até 16 canais no chip.

2.6 Sinais de Controle

Para que a conversão seja controlada pela CPU, são necessários sinais de início de conversão (SC) e fim de conversão (EOC). Quando o SC é enviado, o ADC começa a converter o valor da entrada analógica de V_{IN} em um número digital. A quantidade de tempo que leva para converter varia dependendo do método de conversão.

Quando a conversão de dados estiver completa, o sinal de fim de conversão notifica a CPU que os dados convertidos estão prontos para serem coletados. É possível, então, realizar a verificação do *status* de concluído usando interrupção ou *polling*.

3 Programação do Periférico ADC

O microcontrolador ARM da placa KL25Z possui um único módulo ADC que pode suportar até 31 canais ADC, como mostra a figura 04. Esses canais ADC têm resolução (máxima) de 16 bits. Para programá-los precisamos entender alguns dos principais registradores de controle do periférico.

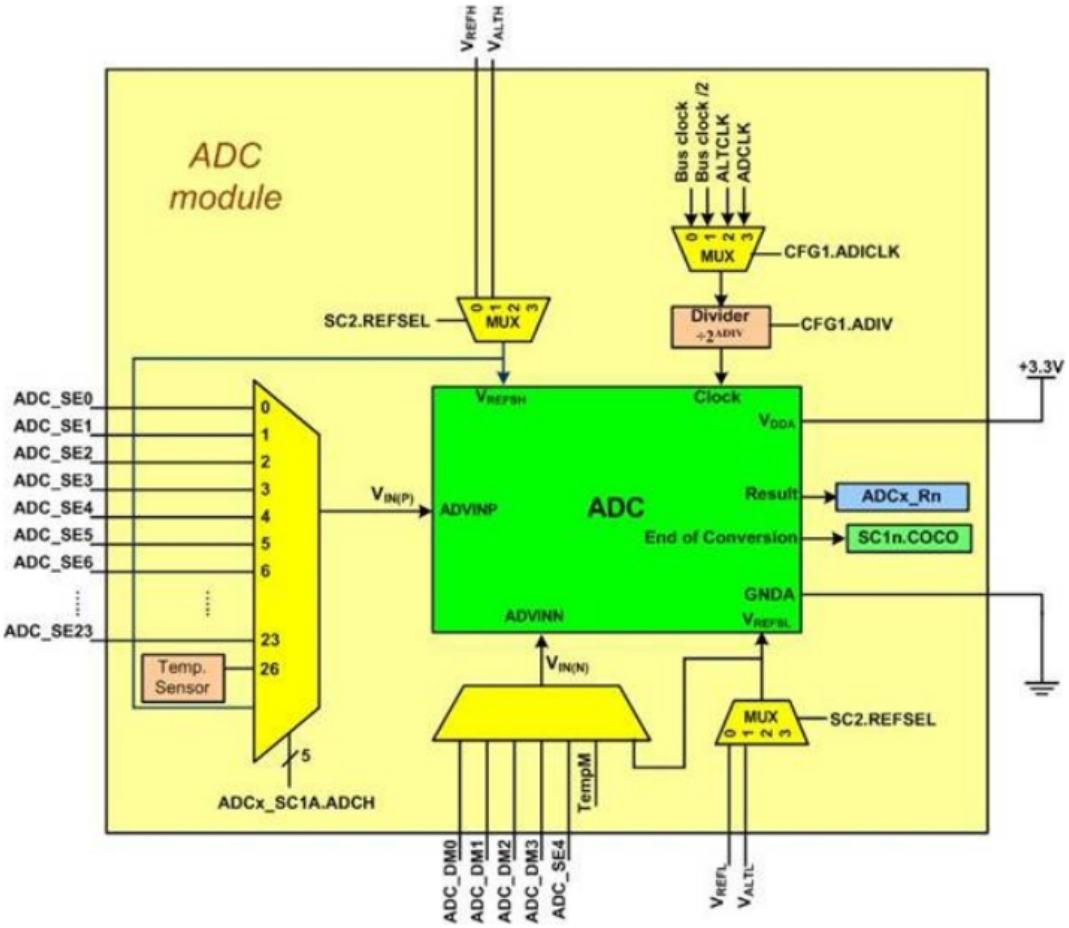

Figura 4: Diagrama de Blocos do Periférico ADC da Placa KL25Z

A primeira coisa que precisamos fazer é habilitar o *clock* para o módulo ADC0, ativando o bit 27 no registrador SCGC6, do módulo SIM.

Existem duas opções de gatilhos, ou *trigger*, para iniciar a conversão: gatilho de *hardware* e gatilho de *software*. A seleção do *trigger* de *hardware* ou software *para* conversão é feita através do bit 6 (ADTRG, *ADC Trigger*) do registrador de controle SC2 do ADC0.

O gatilho de hardware pode ser um pino externo, comparador ou temporizadores (TPMx, LPTMR0, PIT ou RTC). A seleção do trigger de hardware é feita no registrador SOPT7. O gatilho padrão é o gatilho por *software*, que é o que usamos nos códigos.

A seleção do canal é feita no registrador SC1[0], como mostra a figura 05. Os 5 primeiros

bits são usados para selecionar um dos 31 canais a serem convertidos. Nem todos os canais estão conectados a pinos de entrada.

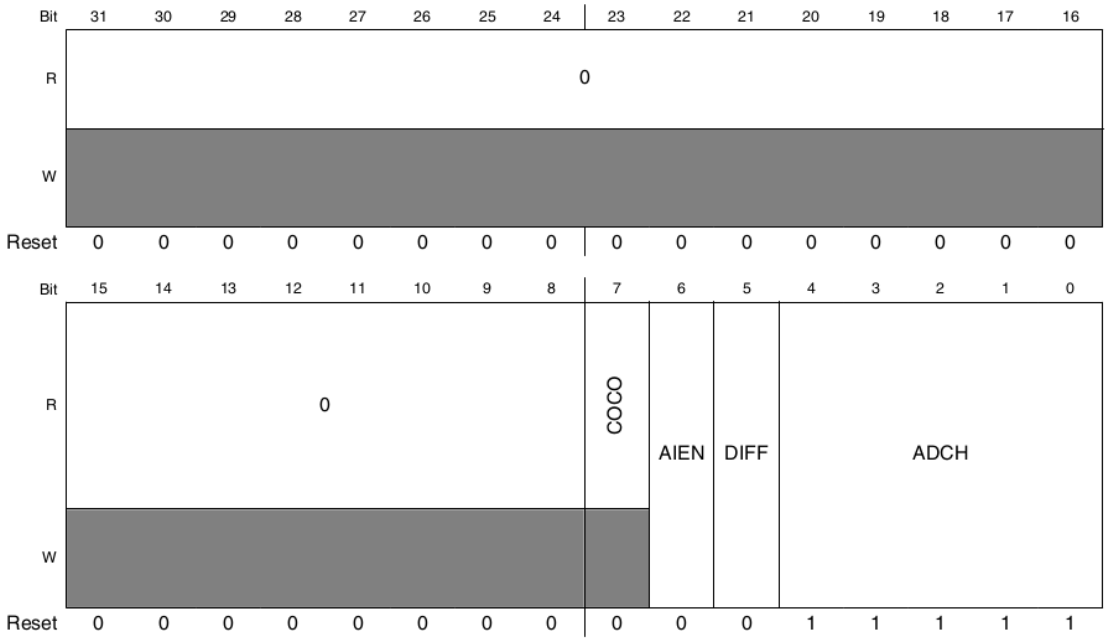

Figura 5: Registrador SC1[0] do ADC0

O fim da conversão é indicado por um bit no registrador SC1[0]. Após a conclusão da conversão, o sinalizador de conversão concluída no bit 7 (COCO, **Conversion Complete**) fica 1. Ao verificar este sinalizador, sabemos se a conversão foi concluída e, se sim, podemos ler o valor no registrador de resultado de dados, o R[0].

Também podemos usar uma interrupção para nos informar que a conversão está completa, mas isso exigirá que definamos o bit AIEN (Interrupt Enable, no bit 6) como 1, no registrador SC1[0]. Por padrão, a interrupção não está habilitada e nós faremos a verificação via *polling*, ou seja, constantemente verificando a tensão lida. A *flag* de conversão completa (COCO), no registrador SC1[0], é limpa automaticamente quando os dados do registrador R[0] são lidos.

Após a conclusão da conversão, o resultado é colocado no registrador R[0]. Este é um registro de 32 bits, mas apenas os 16 bits inferiores são usados, já que a resolução máxima para o ADC é 16-bit. Outras resoluções são de 8-bit, 10-bit e 12-bit.

Usamos o registrador CFG1 para selecionarmos a resolução do ADC, como mostra a figura 06, nos bits 2 e 3 (campo de bits MODE): 8 (0b00), 10 (0b10), 12 (0b01) ou 16 bits (0b11). Este registrador também é usado para selecionar a frequência da fonte de *clock* para o ADC. Para isso configuramos os dois primeiros bits do registrador, o campo ADICLK. O padrão é 0b00 (*clock* do barramento, que tem metade da frequência do *clock* do núcleo), porém podemos configurar um divisor dessa frequência colocando 0b10 em ADICLK para obter metade da frequência do barramento, e podemos configurar os bits 6-5, o campo ADIV, que define mais uma divisão de frequência. O padrão é 0b00, que divide por 1, porém é possível dividir por 2 (0b01), 4 (0b10) e 8 (0b11).

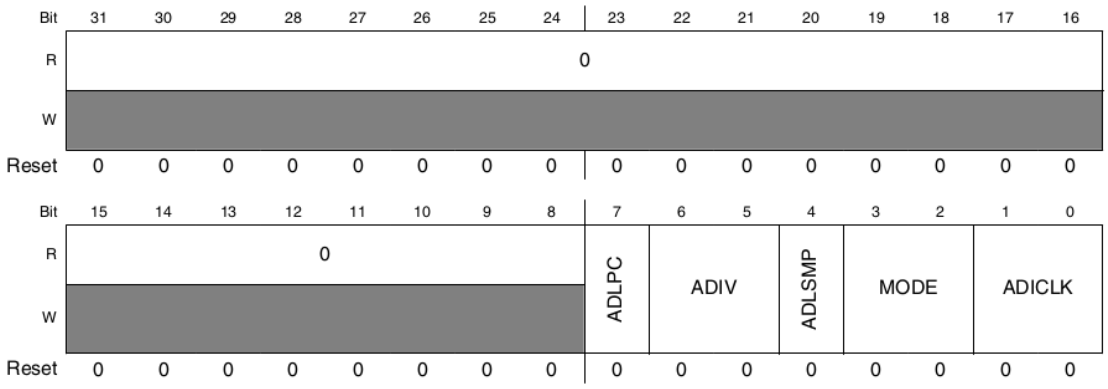

Figura 6: Registrador CFG1 do ADC0

Na placa *Freedom* o pino V_{ref} é chamado **VREFH**, que é conectado a 3,3V, a mesma tensão de alimentação da parte digital do *chip*. Com VREFH = 3,3V, temos o tamanho do passo de $3,3V \div 65.536 = 0,05$ mV, pois a resolução máxima do ADC para KL25Z é de 16 bits.

Para realizar a leitura deve-se informar o canal a ser lido nos primeiros 5 bits de SC1[0], o campo de bits ADCH, todas as vezes, pois ao limpar a *flag* COCO no ato da leitura de R[0] o ADC volta a ser desabilitado, ficando com ADCH = 0b11111. Após isso deve-se esperar que a *flag* COCO seja ativada. Quando a *flag* for ativada pode realizar a leitura do registrador de dados. Ao ler de R[0] a *flag* será limpa. O processo então se repete.

4 Leitura da Tensão em um Potenciômetro

Devemos, para a primeira prática, realizar a leitura coletada de um potenciômetro, mostrado na figura 07, ligado no pino PTB1, configurado para ADC, e mostrar o valor coletado e a tensão calculada no terminal.

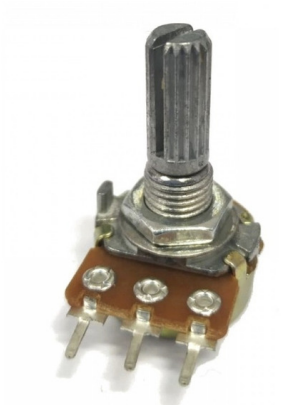

Figura 7: Potenciômetro utilizado na prática

Abaixo está o código utilizado para a prática:

```
#include "MKL25Z4.h"

#include <stdio.h>
#include <stdint.h>

void adcInitModule(void);

int main(void) {

  adcInitModule();

  uint32_t data;
  uint32_t voltage;

  while(1){

    ADC0->SC1[0] = 9;

    while(!(ADC0->SC1[0] & (1 << 7)));

    data = ADC0->R[0];

    voltage = (data * 3300) / 65535;

    printf("Data: %d\nVoltage: %d\n", data, voltage);
  }
}

```

```
/* Inicia o Canal 9 do ADC em PTB1*/
void adcInitModule(void){
  // Habilita o clock para PORTB
  SIM->SCGC5 |= (1 << 10);

  // Funcao analogica para PTB1
  PORTB->PCR[1] &= ~(0b111 << 8);

  // Habilita clock para o ADC0
  SIM->SCGC6 |= (1 << 27);

  // Acionamento por Software
  ADC0->SC2 &= ~(1 << 6);

  // Seleciona clock do barramento
  ADC0->CFG1 &= ~(0b11 << 0);

  // Resolucao de 16-bit
  ADC0->CFG1 |= (0b11 << 2);

  // Clock de Entrada dividido por 8
  ADC0->CFG1 |= (0b11 << 5);

  // Tempo de amostragem longo
  ADC0->CFG1 |= (1 << 10);

  // Media de 4 amostras
  ADC0->SC3 |= (1U << 2);
  ADC0->SC3 &= ~(0b11 << 0);
}
```

O código acima realiza a configuração de uma conversor digital analogico (ADC) com o objetivo de captar o sinal analógico de um potenciômetro de 10k, e mostrar o valor total da tensão obtida.

Primeiramente na função principal estamos chamando as funções de configurações do componente ADC, adiante, dentro do laço infinito, estamos acessando a leitura no canal de entrada dos dados referente ao pino de conversão ADC, o canal 9, usando o registrador SC1A, após isso monitoramos a *flag* COCO no registrador. Logo após para encontrarmos o valor digital lindo da conversão fazemos o seguinte cálculo para obter a tensão:

$$\text{Tensão (em mV)} = (\text{Leitura Digital} * 3300) \ / \ 65535$$

Utilizamos o valor 3300 para que possamos ver as casas decimais da tensão, já que o valor da leitura irá variar de 0 a 65535, devendo multiplicarmos a razão pelo máximo valor que devemos obter.

Já na função de configuração do módulo de leitura ADC, em adcInitModule, estamos realizando toda configuração do pino de leitura do conversor ADC, como habilitar o *clock* da porta, e *clock* do conversor, e a faixa de amostragem do conversor, a resolução.

5 Controle de um Motor por meio de um Potenciômetro

Utilizando como base o código anterior, dessa vez utilizamos a tensão coletada do potenciômetro e controlamos um servo motor, como mostra a figura 08, conectado no pino PTE20, configurado para PWM. Portanto, a rotação eixo do potenciômetro configura a rotação do eixo do motor servo.

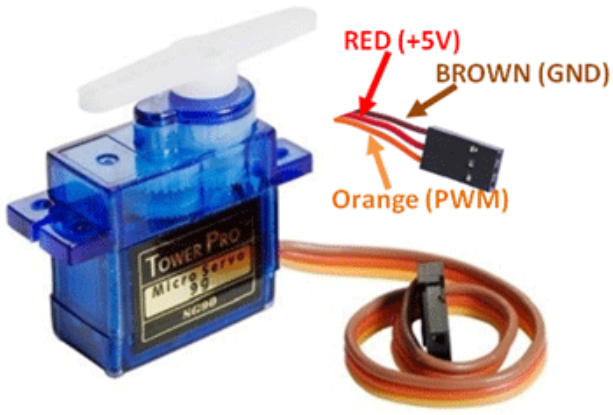

Figura 8: Motor Servo SG90 utilizado na prática

Como foi dito no relatório 05, o *duty cycle* do motor referente a cada ângulo é definido pela fabricante, sendo 20ms o período, 2,5ms para 180° e 0,5ms para 0°. O menor e maior valor para o CnV para os *duty cycle* informados são, respectivamente, 82 e 410, para um módulo de 3276.

Abaixo encontra-se o código da prática.

```
#include "MKL25Z4.h"

#include <stdio.h>
#include <stdint.h>

#define MIN_VALUE_SERVO 82
#define MAX_VALUE_SERVO 410

void tpmInitModule(void);

void adcInitModule(void);

bool adcCalibration(void);

int main(void) {

  tpmInitModule();

```

```
19    adcInitModule();
20
21    uint32_t result = 0;
22    uint32_t step_motor = 0;
23
24    while(1){
25
26      while(!adcCalibration());
27
28      ADC0->SC1[0] = 9;
29
30      while(!(ADC0->SC1[0] & (1 << 7)));
31
32      result = ADC0->R[0];
33
34      // Mapeia o valor do ADC para o PWM
35      step_motor = (result * MAX_VALUE_SERVO) / 65535;
36
37      if(step_motor < MIN_VALUE_SERVO) {
38          step_motor = MIN_VALUE_SERVO;
39      }
40
41      TPM1->CONTROLS[0].CnV = step_motor;
42
43          // O "printf" interferira no tempo de obten  o do ADC
44          // printf("Motor Servo: %d\n", step_motor);
45    }
46  }
47
48  /**
49   * Inicializa o Canal 0 do modulo TPM1 em PTE20 para PWM.
50   */
51  void tpmInitModule(void) {
52
53    /* Habilitando o clock de PORTE */
54    SIM->SCGC5 |= (1 << 13);
55
56    /* Seleciona ALT3: TPM1_CH0 */
57    PORTE->PCR[20] |= (0b011 << 8);
58
59    /* Habilita o clock de TPM1 */
60    SIM->SCGC6 |= (1 << 25);
61
62    // MOD = (20971520/128)*0,02 = 3277
63    TPM1->MOD = 3276;
64
65    /* Seleciona Modo de Contagem Crescente (Padrao) e Prescale de 128 */
66    TPM1->SC |= (0b111 << 0);
67
68    /* Desativar o PWM no modulo TPM1_CH0 (PTE20) */
69    TPM1->CONTROLS[0].CnSC = 0;
70
71    /* Ativar o PWM no modulo TPM1_CH0 (PTE20) alinhado a borda com pulsos high
        true */
```

```
72    TPM1->CONTROLS[0].CnSC |= (0b10 << 4) | (0b10 << 2);
73
74    /* Inicia a Contagem */
75    TPM1->SC |= (0b01 << 3);
76  }
77
78  /* Inicia o Canal 9 do ADC em PTB1*/
79  void adcInitModule(void){
80    // Habilita o clock para PORTB
81    SIM->SCGC5 |= (1 << 10);
82
83    // Funcao analogica para PTB1
84    PORTB->PCR[1] &= ~(0b111 << 8);
85
86    // Habilita clock para o ADC0
87    SIM->SCGC6 |= (1 << 27);
88
89    // Acionamento por Software
90    ADC0->SC2 &= ~(1 << 6);
91
92    // Seleciona clock do barramento
93    ADC0->CFG1 &= ~(0b11 << 0);
94
95    // Resolucao de 16-bit
96    ADC0->CFG1 |= (0b11 << 2);
97
98    // Clock de Entrada dividido por 8
99    ADC0->CFG1 |= (0b11 << 5);
100
101   // Tempo de amostragem longo
102   ADC0->CFG1 |= (1 << 10);
103
104   // Media de 4 amostras
105   ADC0->SC3 |= (1U << 2);
106   ADC0->SC3 &= ~(0b11 << 0);
107 }
108
109 bool adcCalibration(void) {
110   uint16_t calib;
111
112   // Inicia a calibracao
113   ADC0->SC3 |= (1 << 7);
114
115   // Espera a calibracao terminar
116   while (ADC0->SC3 & (1 << 7));
117
118   // Verifica se foi bem sucedida, se nao, retorna falso
119   if (ADC0->SC3 & (1 << 6))
120     return false;
121
122   calib = 0;
123   calib += ADC0->CLPS + ADC0->CLP4 + ADC0->CLP3 +
124            ADC0->CLP2 + ADC0->CLP1 + ADC0->CLP0;
125   calib /= 2;
```

```
126
127   // Ativa o MSB
128   calib |= 0x8000;
129   ADC0->PG = calib;
130
131   calib = 0;
132   calib += ADC0->CLMS + ADC0->CLM4 + ADC0->CLM3 +
133            ADC0->CLM2 + ADC0->CLM1 + ADC0->CLM0;
134   calib /= 2;
135
136   // Ativa o MSB novamente
137   calib |= 0x8000;
138   ADC0->MG = calib;
139
140   return true;
141 }
```

O código acima realiza a configuração de uma conversor digital analogico (ADC) com o objetivo de captar o sinal analógico de um potenciômetro de 10 kΩ, e de acordo com o valor obtido realizar o movimento de um servo motor.

Primeiramente, na função principal, estamos chamando as funções de configurações do ADC e TPM(para geração do PWM), e dentro do laço infinito estamos realizando a leitura no canal de entrada dos dados referente ao pino de conversão ADC, PTB1, o canal 9.

Após isso, para encontrarmos o valor para o CnV, que irá definir o *duty cycle*, a partir do valor digital lido da conversão, fazemos o seguinte cálculo:

$$\text{CnV} = (\text{Valor Lido} \times 410)/65535$$

O valor máximo para CnV será 410, já que o valor máximo lido sera 65535, como consta a resolução de 16-bit do ADC. Caso este valor seja menor que 82, o CnV receberá 82. Com esse cálculo podemos emitir o sinal de PWM, responsável por controlar o movimento, de forma correta.

Já na função de configuração do módulo de leitura ADC, **adcInitModule**, estamos realizando toda configuração do pino de leitura do conversor ADC. Diferentemente da prática anterior configuramos também o módulo TPM, na função **tpmInitModule**, para emitir o sinal de controle do servo motor referente ao valor obtido do potenciômetro.

Foi adicionada uma função de calibração, que irá calibrar o módulo ADC sempre antes de uma conversão. Falamos mais da função de calibragem na próxima seção. Usamos ela aqui para suavizar o movimento do motor, já que os dados obtidos pelo conversor serão mais aproximados uns dos outros. Isso diminuirá a trepidação do motor.

6 Leitura da Temperatura com o Sensor LM35

Utilizando a mesma configuração de ADC devemos utilizar um sensor de temperatura, o LM35, como mostra a figura 09, para coletar a temperatura do ambiente e mostrá-la no terminal. O terminal central do sensor, então, encontra-se conectado ao pino PTB1.

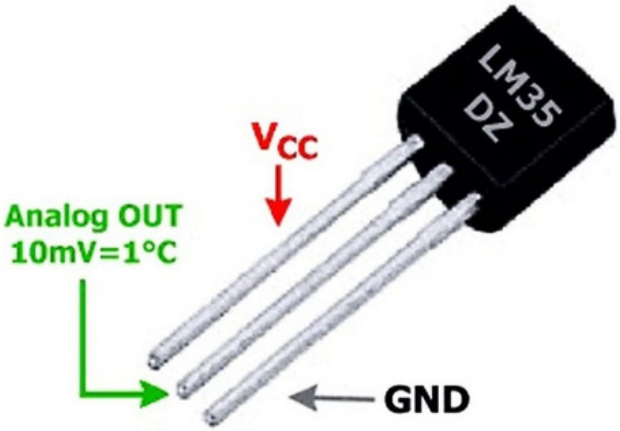

Figura 9: Sensor de Temperatura LM35

O sensor irá emitir uma tensão de 10 mV para cada 1 grau *celsius*. Devemos, então, sentir essa tensão e calcularmos o valor da temperatura a partir da tensão emitida neste pino.

Abaixo encontra-se o código da prática:

```
#include "MKL25Z4.h"

#include <stdio.h>
#include <stdint.h>
#include <stdbool.h>

void adcInitModule(void);

bool adcCalibration(void);

int main(void) {

  adcInitModule();

  uint32_t data;
  float temperature;

  while(1){

    while(!adcCalibration());

    ADC0->SC1[0] = 9;
```

```
23
24      while(!(ADC0->SC1[0] & (1 << 7)));
25
26      data = ADC0->R[0];
27
28      temperature = data * 330.0 / 65535;
29
30      printf("Temperatura: %d C\n", (int) temperature);
31    }
32  }
33
34  /* Inicia o Canal 9 do ADC em PTB1*/
35  void adcInitModule(void){
36    // Habilita o clock para PORTB
37    SIM->SCGC5 |= (1 << 10);
38
39    // Funcao analogica para PTB1
40    PORTB->PCR[1] &= ~(0b111 << 8);
41
42    // Habilita clock para o ADC0
43    SIM->SCGC6 |= (1 << 27);
44
45    // Acionamento por Software
46    ADC0->SC2 &= ~(1 << 6);
47
48    // Seleciona clock do barramento
49    ADC0->CFG1 &= ~(0b11 << 0);
50
51    // Resolucao de 16-bit
52    ADC0->CFG1 |= (0b11 << 2);
53
54    // Clock de Entrada dividido por 8
55    ADC0->CFG1 |= (0b11 << 5);
56
57    // Tempo de amostragem longo
58    ADC0->CFG1 |= (1 << 10);
59
60    // Media de 4 amostras
61    ADC0->SC3 |= (1U << 2);
62    ADC0->SC3 &= ~(0b11 << 0);
63  }
64
65  bool adcCalibration(void) {
66    uint16_t calib;
67
68    // Inicia a calibracao
69    ADC0->SC3 |= (1 << 7);
70
71    // Espera a calibracao terminar
72    while (ADC0->SC3 & (1 << 7));
73
74    // Verifica se foi bem sucedida, se nao, retorna falso
75    if (ADC0->SC3 & (1 << 6))
76      return false;
```

```
77
78    calib = 0;
79    calib += ADC0->CLPS + ADC0->CLP4 + ADC0->CLP3 +
80             ADC0->CLP2 + ADC0->CLP1 + ADC0->CLP0;
81    calib /= 2;
82
83    // Ativa o MSB
84    calib |= 0x8000;
85    ADC0->PG = calib;
86
87    calib = 0;
88    calib += ADC0->CLMS + ADC0->CLM4 + ADC0->CLM3 +
89             ADC0->CLM2 + ADC0->CLM1 + ADC0->CLM0;
90    calib /= 2;
91
92    // Ativa o MSB novamente
93    calib |= 0x8000;
94    ADC0->MG = calib;
95
96    return true;
97  }
```

O código acima realiza a configuração de uma conversor digital analogico (ADC) com o objetivo de captar o sinal analógico de um sensor de temperatura LM35, e de acordo com o valor obtido mostrar o valor obtido no terminal.

Primeiramente, na função principal, estamos chamando as funções de configurações dos componentes, e, no laço, obtendo o valor da leitura e realizando o seguinte cálculo para obter o valor da temperatura:

$$\text{Temperatura} = ((\text{Resultado da Leitura} * 3.3\text{V}) \,/\, 65535) \,/\, 10\text{mV}$$

Logo, obtemos o seguinte cálculo:

$$\text{Temperatura} = (\text{Resultado da Leitura} * 330.0) \,/\, 65535$$

Diferente das outras práticas, por estarmos trabalhando com sensor de temperatura, devemos realizar a calibragem da leitura analógica. Para isso criamos a função **adcCalibration** que através do registrador **SC3** inicia esse processo de calibragem da leitura. Além disso vale salientar que para essa calibragem utilizamos os registradores CLPS, CLP4, CLP3, CLP2, CLP1, CLP0. O processo completo de calibragem pode ser encontrado no manual de referência do processamento. A leitura, então, só é realizada quando o processo de calibração é realizado com sucesso.

7 Conclusão

Ao realizarmos a prática a equipe obteve uma noção melhor de como funciona o ADC em sistemas embarcados. Além disso, os conteúdos teóricos ajudaram a ter uma introdução a temas mais avançados, estudados em outras disciplinas como Eletrônica II, Sinais e Sistemas e Processamento Digital de Sinais.

Outro benefício que a prática proporcionou à equipe foi o melhor entendimento de como se comportam os sinais de escada referentes às harmônicas derivada da conversão analógica-digital que foram vistos nas aulas teóricas. Além de mostrar e fixar para os integrantes da equipe como é realizado a amostragem de um sinal analógico.

Ao final podemos concluir que a realização da prática fixou e melhorou os ensinamentos passados nas aulas teóricas da disciplina de microcontroladores, e introduziu um entendimento teórico de conteúdos mais complexos vistos em disciplinas futuras da engenharia.

Referências

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