EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 9 – ADC e LPTMR

Profa. Wu Shin-Ting

OBJETIVO: Apresentação das funcionalidades do módulo ADC (Conversor Analógico-Digital).

ASSUNTOS: Programação do módulo ADC do MKL25Z128 para conversão de sinais analógico-digitais com iniciação controlada por *software* ou por *hardware*.

O que você deve ser capaz ao final deste experimento?

Saber a diferença entre amostragem e quantização.

Entender o princípio de funcionamento de um conversor ADC por registrador de aproximações sucessivas (SAR).

Entender os termos utilizados para caracterizar um conversor ADC.

Entender os erros inerentes a um algoritmos de quantização de um sinal analógico.

Entender os erros envolvidos no *hardware* de conversão e o uso de auto-calibração para corrigir automaticamente os erros de *offsset* e de ganho de um ADC.

Entender a importância da filtragem dos sinais analógicos e o impacto de uma resistência filtro no tempo de amostragem.

Entender a importância de filtragem dos sinais de alimentação e referência.

Saber configurar o conversor ADC para operar em diferentes modos de operação.

Saber estimar o tempo de conversão de um circuito ADC.

Entender o princípio de funcionamento de um temporizador/contador LPTMR.

Saber configurar LPTMR para operar como um temporizador.

Saber programar em KL25Z conversões periódicas disparadas por *hardware*.

Entender a técnica de filtragem exponencial na aquisição de dados com "memória".

Saber recuperar uma grandeza física a partir do valor binário amostrado por um conversor ADC.

Saber implementar os exemplos de aplicação dos módulos do microcontrolador apresentados em [5].

INTRODUÇÃO

A maioria dos sensores e sistemas audiovisuais gera sinais analógicos. Para serem processados pelos processadores digitais, como nosso MCU, estes sinais precisam ser digitalizados, ou seja, convertidos em sinais digitais através de um **conversor Analógico-Digital** (ADC). Podemos conceituar um conversor ADC como um circuito que busca mapear, de forma mais linear possível, as amostras de tensões no intervalo `[VREFL,VREFH]` em sinais digitais específicos, correspondentes aos códigos binários 0 a 2^N-1, onde `N` é o número de *bits* predefinidos para representação digital.

O bloco comum a todos os conversores analogico-digital (**ADC**) é um circuito de captura/**amostragem e retenção** (*sample and hold*) em que o circuito lê uma **amostra** Vi do sinal a

ser convertido e a **mantém retida** num componente, como um capacitor, de modo que, mesmo que o sinal varie, a amostra Vi é preservada até o próximo instante de captura. Em seguida, o valor analógico Vi é, convertido em um valor digital por um processo denominado **quantização**. Há diferentes técnicas de quantização, envolvendo distintas tecnologias [4]. Essencialmente, distinguem-se dois esquemas: com e sem uso de um conversor digital-analógico (**DAC**). Os mais utilizados são os que envolvem conversores DAC. Dada uma saída digital de N *bits*, o procedimento consiste em varrer as possíveis combinações binárias de N *bits*, convertendo-as em valores analógicos e comparando-os com o valor analógico de entrada, até alcançar um valor que se aproxime da entrada. Esse processo ocorre em 5 passos, conforme ilustrado na Figura 1:

1. Inicialização por disparo (`Start`);
2. Geração de um valor binário por meio de um contador (`Counter`), por exemplo;
3. Conversão do valor binário em tensão analógica Va via DAC;
4. Comparação entre Va e a tensão de entrada Vi;
5. Ativação do fim da conversão (EOC, *end-of-conversion*), quando Va-Vi é suficientemente pequena; caso contrário, volta-se ao passo (2) para tentar com o próximo valor binário do `Counter`.

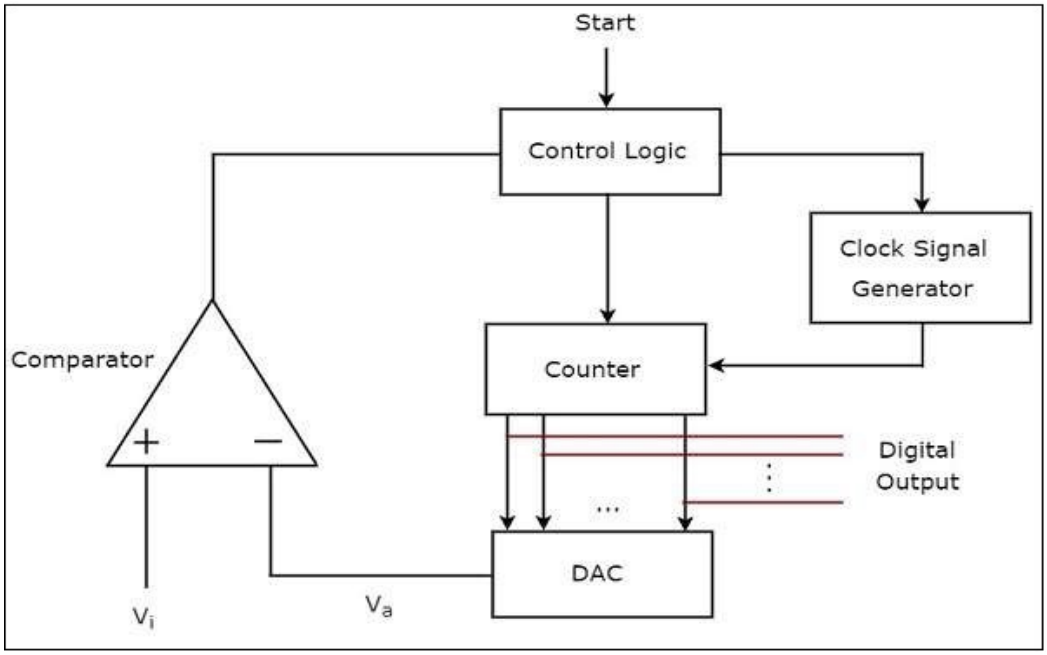


Figura 1: Diagrama de blocos de um conversor ADC (Fonte: [16]).

Neste experimento, vamos explorar as funções configuráveis do módulo ADC por meio de dois exemplos de projeto. Em seguida, aplicaremos o conhecimento adquirido no desenvolvimento de um projeto `controle_cooler`, que visa controlar a velocidade de um *cooler* através de um potenciômetro rotativo. Esse potenciômetro permite alterar as tensões girando um botão conectado ao seu eixo de giro. As tensões na saída do potenciômetro são amostradas periodicamente pelo pino PTB1 e convertidas em códigos binários pelo módulo ADC.

Os valores binários resultantes são utilizados para controlar o ciclo de trabalho de um sinal PWM gerado pelo canal `TPM1_CH0` (PTB0), que alimenta a base de um transistor Darlingtong NPN TIP31. Esse transistor chaveia entre os estados de corte e de saturação em um circuito em série com um *cooler* (Figura 10). Temporizadores são configurados para gerar disparos periódicos, inicializando conversões no ADC e assegurando a periodicidade na amostragem dos sinais do potenciômetro.

Amostragem e Quantização

De acordo com o t**eorema de amostragem de *Nyquist***, a frequência de amostragem (f) deve ser pelo menos duas vezes a maior frequência presente no espectro do sinal. Além disso, o tempo de amostragem e de conversão deve ser menor que 1/f para permitir a recuperação perfeita do sinal a partir de uma sequência infinita de amostras.

Um sistema digital de N *bits* pode representar até 2^N valores distintos. Se as tensões de entrada forem divididas igualmente entre 2^N **níveis de quantização**, a menor unidade representável é o ***bit* menos significativo** (***least significant bit***, **LSB**). Uma variação discreta nesse *bit* corresponde a uma faixa de variações contínuas no sinal analógico de entrada. O valor do LSB é definido como a razão entre o valor máximo VREF e 2^N níveis (LSB = VREF/2^N). Essa razão é conhecida como a **resolução** do conversor, frequentemente expressa em termos percentuais do valor máximoVREF, isto é,

$$Resolução = \frac{LSB}{VREF} \times 100\,. \qquad (1)$$

Na prática, essa resolução é especificada pelo número N de *bits* do conversor.

Chamamos de **erro de quantização** a diferença entre o valor analógico VA e o valor analógico correspondente ao nível de quantização de VA. Quando o valor analógico representativo de um nível de quantização corresponde ao ponto médio do intervalo, o erro máximo de quantização é |0.5 LSB|. Se o valor analógico representativo for o extremo inferior do intervalo, o erro máximo de quantização é |1 LSB|.

A Seção 28.6.2.4/página 505 em [1] mostra que no microcontrolador KL25Z o valor analógico representativo é o ponto médio para as conversões de 8, 10 e 12 *bits* conforme ilustrado na Figura 2. Note a assimetria na conversão: a largura do degrau do código binário 0 é 0.5LSB e a do último código binário 2^N-1 é 1.5LSB.

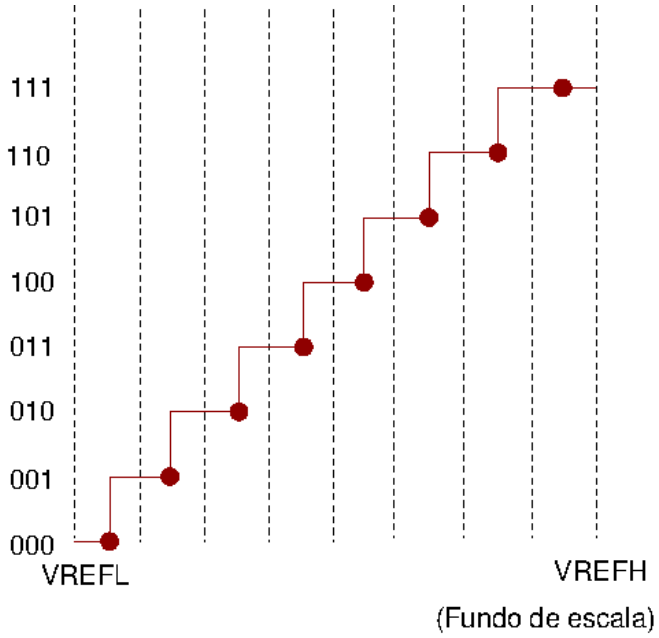


Figura 2: Níveis de quantização: pontos médios de intervalos analógicos.

Por outro lado, para a conversão de 16 *bits*, o valor analógico representativo é o extremo inferior do intervalo como mostra na Figura 3. Neste caso, o erro de quantização varia entre -1LSB (maior "desvio" em relação à entrada amostrada) e 0LSB, e a largura do degrau é 1LSB para todos os níveis de quantização.

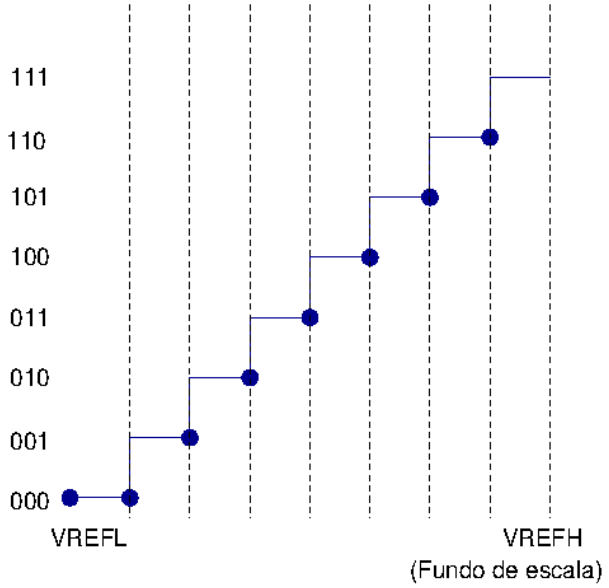


Figura 3: Níveis de quantização: extremo inferior de intervalos analógicos.

Erros inerentes do *Hardware* de ADC

Além do erro de quantização, há outros erros inerentes aos componentes de um conversor, como as tensões de referência, os resistores e os comparadores. Esses erros podem causar desvios entre o valor digitalizado e o valor analógico real. A **acurácia** de um conversor é uma medida dessa diferença, geralmente epressa em percentagem de VREF. Embora, na prática, a acurácia e a resolução estejam na mesma ordem de grandeza, são dois parâmetros independentes. Aumentar a quantidade de *bits* não necessariamente aumenta a acurácia dos componentes de um conversor, .

Os erros inerentes aos componentes de um conversor podem ser classificados em erros estáticos e erros dinâmicos. Os **erros dinâmicos** são desvios que ocorrem durante operações transitórias ou dinâmicas de um sistema. Os **erros estáticos**, por sua vez, são associados a condições estáveis, como converter um sinal de corrente contínua (CC). Eles podem ser completamente descritos por quatro tipos de erros, que são aplicados na análise do desempenho de um conversor ADC [15]:

- **erros de *offset***: é o valor de entrada no meio do degrau quando o nível de quantização é 0.
- **erro de ganho**: é a diferença entre o valor ideal do meio do degrau e o valor real do meio do degrau quando o nível de quantização atinge o máximo.
- **erro de linearidade diferencial**: é a diferença entre a largura ideal de 1LSB de um degrau e a largura real do degrau.
- **erro de linearidade integral:** é o desvio da inclinação da função real de conversão em relação à função ideal.

Registrador por Aproximações Sucessivas

A Figura 1 apresenta uma das versões mais simples do ADC, na qual um contador (`Counter`) é empregado para varrer incrementalmente os possíveis códigos binários, até encontrar um valor que esteja suficientemente próximo do valor amostrado. Devido ao sinal de saída do `Counter` ter a forma de onda de uma rampa, este tipo de conversor é conhecido como ADC de **rampa digital**.

Para otimizar o tempo de busca por um código binário mais próximo, reduzir o consumo de energia e simplificar o circuito, os ADCs integrados em microcontroladotes adotam o **registrador de aproximações sucessivas** (***Successive Approximation Register,*** SAR) [2]. O SAR é um circuito digital que usa uma técnica de busca binária para determinar o valor digital correspondente ao valor analógico de entrada.

A **técnica de busca binária** consiste em dividir o intervalo de valores possíveis em duas metades a cada iteração, até encontrar o valor que mais se aproxima do valor analógico de entrada. Isso resulta em uma redução na quantidade de comparações necessárias. Enquanto um ADC de N *bits* de rampa digital pode exigir até 2^N comparações para encontrar o valor digital correspondente ao valor analógico de entrada, um ADC de registrador de aproximações sucessivas pode realizar a busca em apenas $\log_2(2^N) = N$ comparações. Essa redução é possível devido à natureza eficiente da técnica de busca binária utilizada pelo registrador de aproximações sucessivas, que é análoga à estrutura de uma árvore binária de busca balanceada.

Módulo ADCx

O microcontrolador KL25Z incorpora um conversor analógico-digital de 16 *bits* (Capítulo 28/página 457, em [1]). A técnica implementada é a de **aproximações sucessivas**, utilizando um registrador de aproximações sucessivas (*successive approximation register SAR*) de 16 *bits*[2]. Nesse método, o `Counter` mencionado na Figura 1 é substituído por um circuito SAR, como ilustra a Figura 4. O comparador retroalimenta o circuito do registrador SAR com a diferença entre os sinais, atualizando continuamente o conteúdo do SAR com base nesta diferença. Esse processo se repete sucessivamente até que a tensão correspondente ao código binário no SAR se aproxime suficientemente de V_{IN} dentro de uma faixa de tolerância pré-estabelecida. Nesse momento, o *bit* de estado EOC (*end of conversion*)/COCO (*conversion complete*) é definido como ‘1’.

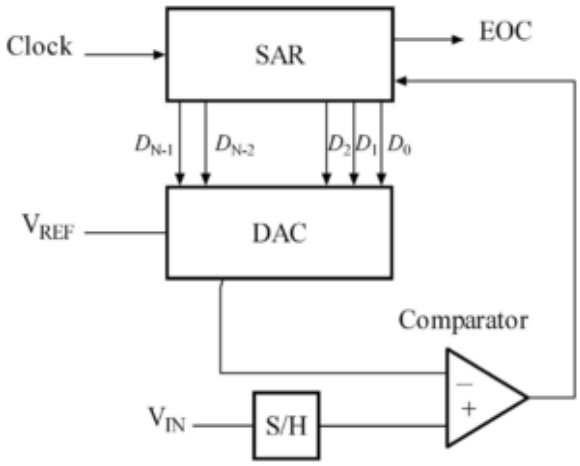


Figura 4: Diagrama de blocos de um ADC por SAR (Fonte: [18]).

De acordo com a tabela na Seção 3.7.1.3.1/página 79 em [1], o conversor ADC do KL25Z possui 16 pinos físicos destinados à entrada dos sinais analógicos. Esses pinos podem ser configurados para operar como 24 entradas únicas para conversões unipolares (*singular*) ou como 4 entradas

diferenciais para conversões bipolares (*differential*). As tensões de referência VREFH e VREFL utilizadas durante a conversão (Seção 28.4.2/página 484 em [1]) são configuráveis por meio dos *bits* ADCx_SC2_REFSEL (Seção 28.3.6/página 470 em [1]). O *kit* disponivel no laboratório foi configurado para operar no modo 0b00 (VREFH~3V3 e VREFL=0V) (Seção 28.6.1.2/página 502 em [1]). Todas as amostras de tensão dentro desse intervalo são diretamente proporcionais aos códigos binários de 0 até 2^N-1, sendo N a quantidade de *bits* alocados para representação digital.

No KL25Z, a resolução de uma conversão pode ser configurada para 8, 10, 12 ou 16 *bits*, através dos *bits* ADCx_CFG1_MODE (Seção 28.3.2/página 465 em [1]). Amostras menores que VREFL e maiores que VREFH têm, respectivamente, os códigos binários truncados em 0 e 2^N-1, conforme explicado na Seção 28.6.1.3/página 503 em [1].

O modo de operação, uni- ou bipolar, em cada entrada é controlado pelos *bits* de controle ADCx_SC1n[DIFF. Como há apenas um circuito conversor, o sinal analógico processado em cada instante é determinado pelos 5 *bits* ADCx_SC1n_ADCH da entrada SC1n selecionada pelo *bit* de configuração ADCx_CFG2_MUXSEL.

O microcontrolador incorpora um sensor de temperatura AN3031 [3] (Seção 28.4.8/página 497 em [1]). Esse sensor pode ser usado para monitorar a temperatura do processador. Ele é designado ao canal 0b11010 do conversor. Portanto, para amostrar as tensões provenientes deste sensor, basta configurarmos este canal no campo ADCx_SC1A_ADCH. Para desabilitar a entrada SC1n do módulo ADC0, atribuímos o código 0b1111 a ADCx_SC1n_ADCH.

Para garantir que o módulo ADC esteja operacional, é necessário habilitar o sinal de relógio do módulo ADC através do *bit* de configuração SIM_SCGC6_ADC0 (Seção 12.2.10/página 207 em [1]). O ADCx é alimentado por duas fontes de sinais de relógio: o sinal de relógio para comunicação com o barramento (*bus clock*) e o sinal de relógio ADCK para o circuito de conversão (Tabela 5-2/página 121 em [1]). A fonte de ADCK é configurável pelos *bits* de controle ADC0_CFG1_ADICLK (Seção 28.3.3/página 466 em [1]). Além disso, dentro do módulo ADC, há um divisor de frequência ADC0_CFG1_ADIV para reduzir a frequência do sinal de relógio (Seção 28.4.1/página 483 em [1]).

A forma como uma conversão é iniciada ou disparada é configurável através do campo ADCx_SC2_ADTRG (Seção 28.4.3/página 484 em [1]). O disparo pode ser realizado por *software*, através de um acesso de escrita ao registrador ADCx_SC1A, ou por *hardware*, selecionando a fonte de disparo pelos *bits* SIM_SOPT7_ADCxTRGSEL (Seção 12.2.6/página 200 em [1]). No caso de disparo por *hardware*, a fonte selecionada deve estar devidamente configurada para produzir os eventos geradores de disparos. Os disparos acontecem automaticamente, sem intervenção de *software*.

O microcontrolador KL25Z tem 2 entradas abstraídas em dois registradores SC1A e SC1B. Elas são configuráveis separadamente e operáveis no modo "*ping-pong*". Somente a entrada SC1A suporta os dois tipos de disparos, por *software* e por *hardware*. A entrada SC1B só suporta disparos por *hardware*. A tabela na Seção 3.7.1.3.1/página 79 em [1] mostra os sinais de entrada permitidos para ambas as entradas A e B e os nomes dos pinos usados na tabela da Seção 10.3.1/página 161 em [1].

Enquanto uma conversão estiver em progresso, o *bit* ADCx_SC2_ADACT permanece definido em '1'. O modo de conversão pode ser único (uma só vez) ou contínuo (sucessivamente após um único disparo inicial), configurável pelo *bit* ADCx_SC3_ADCO (Seção 28.3.7/página 472 em [1]). Quando

uma conversão é concluída (Seção 28.4.4.2/página 486 em [1]), o valor resultante é agregada aos valores anteriores de conversão para gerar um resultado que seja uma média dos valores amostrados. Essa abordagem ajuda a reduzir o efeito de ruído e variações temporárias nos sinais analógicos, proporcionando uma medida mais estável e confiável. O número de amostras consideradas para cada resultado é configurável através dos campos ADCx_SC3_AVGE e ADCx_SC3_AVGS (Seção 28.4.4.7/página 492 em [1]).

Quando uma função de comparação está ativada pelo *bit* de configuração ADCx_SC2_ACFE, a média gerada é automaticamente comparada com os limites do intervalo predefinido nos registradores de dados ADCx_Cvn. Esses limites devem ser configurados de acordo com as condições especificados pelo fabricante e apresentadas na Tabela 28-78/página 493 em [1]. A forma como essa comparação é realizada pode ser configurada pelos *bits* de controle ADCx_SC2_ACFGT e ADCx_SC2_ACREN.

Se o resultado satisfaz as funções comparadoras configuradas, o *bit* de estado ADCx_SC1n_COCO é automaticamente setado em '1', e o resultado da conversão é armazenado no registrador de dados ADCx_Rn da entrada n selecionado pelo *bit* de configuração ADCx_CFG2_MUXSEL.

O resultado transferido para ADCx_Rn representa, portanto, a média de um conjunto de amostras.

Funções de Comparação em ADCx

Em KL25Z, os termos ACFE, ACREN e ACFGT descrevem as seis funções de comparação disponíveis nos conversores ADC, que determinam a transferência de um resultado de conversão para o registrador de saída ADCx_Rn. Essas 6 funções são configuradas através dos seguintes *bits* (Seção 28.4.5/página 493 em [1]):

1. ADCx_SC2_ACFE (habilitação da função de comparação): quando setado, a função de comparação configurada é aplicada sobre os resultados de conversão.
2. ADCx_SC2_ACFGT (comparação do resultado em relação a ADCx_CV1): quando setado, a função de comparação aplicada no resultado de uma conversão é determinada pela relação "maior que" em relação ao valor ADCx_CV1. Se for resetado, a função é "menor ou igual".
3. ADCx_SC2_ACREN (comparação do resultado em relação aos intervalos definidos por ADCx_CV1 e ADCx_CV2): quando é resetado em '0', a função de comparação configurada em ADCx_SC2_ACFGT é aplicada no resultado da conversão em relação a ADCx_CV1. Se o *bit* é definido como '1', a comparação do resultado de conversão é feita em relação a um dos quatro intervalos de valores possíveis, dependendo das configurações de ADCx_SC2_ACFGT, ADCx_CV1 e ADCx_CV2, Esses intervalos podem ser:
   - (-∞,ADCx_CV1) ou (ADCx_CV2,∞), se ADCx_SC2_ACFGT==0 e ADCx_CV1 ⩽ ADCx_CV2,
   - (ADCx_CV2, ADCx_CV1), se ADCx_SC2_ACFGT==0 e ADCx_CV1 > ADCx_CV2,
   - [ADCx_CV1, ADCx_CV2], se ADCx_SC2_ACFGT==1 e ADCx_CV1 ⩽ ADCx_CV2, e
   - (-∞,ADCx_CV2] ou [ADCx_CV1,∞), se ADCx_SC2_ACFGT==1 e ADCx_CV1 > ADCx_CV2.

Configuração de Disparos de Conversão por *Hardware*

Vimos que uma conversão pode ser desencadeada tanto por *software* quanto por *hardware*. Em KL25Z, todas as entradas suportam disparos por *hardware*. No entanto, apenas a entrada ADCx_SC1A também suporta adicionalmente uma conversão desencadeada por *software*. Ao optar pelo disparo por *hardware*, é necessário configurar e habilitar o módulo responsável pela geração dos sinais de disparo.

Na Tabela 3-1/página 45 em [1], é apresentada uma lista de módulos interconectados em KL25Z, incluindo aqueles cujos eventos gerados podem ser utilizados como disparadores para o módulo ADC (*ADC Trigger*). A sexta coluna dessa tabela mostra os campos do registrador `SIM_SOPT7` que configuram as fontes de disparo para as entradas `A` e `B` do ADC0. Os códigos válidos para serem setados estão descritos na Seção 12.2.6/página 201 em [1].

O projeto `rot9_example1` [17], uma implementação do exemplo de configuração apresentado no Capítulo 11/página 117 em [5], ilustra o uso do temporizador LPTMR0 para gerar disparos periódicos de amostragem e conversão de um sinal analógico no pino PTB1 (canal 9 do ADC0) (Seção 3.7.1.3.1/página 79 e Seção 10.3.1/página 162 em [1]). O projeto `rot9_aula` [11], por sua vez, aplica o temporizador PIT0 para gerar disparos periódicos para conversões no mesmo canal.

Filtragem de Dados Analógicos

Os sinais analógicos estão sujeitos a uma variedade de ruídos e distorções, como interferências eletromagnéticas e flutuações de tensão, que podem introduzir distorções no processo de conversão, comprometendo a precisão das medidas. Para mitigar esses efeitos indesejados, é comum incluir um circuito RC simples e econômico nos pinos de entrada.

Através de sua carga e descarga, o capacitor (C) atua como um filtro, permitindo a passagem de sinais de frequência mais baixa (como o sinal analógico desejado), enquanto bloqueia sinais de frequência mais alta (como ruídos e interferências). O resistor (R), por sua vez, limitaa a corrente que flui para o capacitor, controlando assim a taxa de carga e descarga. O valor típico do resistor (R) é de 100 Ohms, enquanto o valor do capacitor (C) é escolhido para garantir uma atenuação adequada das frequências acima da frequência de Nyquist, que é a metade da frequência de amostragem (Seção 11.2.2/página 120 em [5]):

$$f_{amostragem} = 2 \times f_{Nyquist} = \frac{2}{2 \times \pi \times R \times C} = \frac{1}{\pi \times R \times C} \quad (2)$$

Por exemplo, para uma frequência de amostragem de 30Hz, o valor C deve ser aproximadamente 100uF. No entanto, é importante notar que **a resistência R pode afetar a constante de tempo do circuito**, resultando em uma resposta mais lenta às mudanças no sinal de entrada.Isso pode exigir um tempo de amostragem maior para garantir uma amostragem adequada do sinal filtrado. O módulo ADC0 oferece a possibilidade de ajustes no tempo de amostragem através dos *bits* `ADCx_CFG1_ADLSMP` (Seção 28.3.2/página 466 em [1]) e `ADCx_CFG2_ADLSTS` (Seção 28.3.3/página 467 em [1]).

Além da filtragm dos sinais de entrada, é importante garantir uma fonte de alimentação limpa e sinais de referência estáveis e livres de ruído no projeto de *hardware*. Neste caso, é comum incluir capacitores de baixa resistência equivalente (*equivalent series resistence*, ESR) em pontos-chave do circuito, como entre as referências de tensão (VREFH e VREFL) e entre as alimentações analógicas e de terra (VDDA, VSSA) (Seção 28.6.1.2/página 502 em [1]). Com base em dados empíricos, observa-se que capacitores de 0,1 µF, com baixa resistência equivalente em série, são muitas vezes suficientes. Vale ressaltar que esses capacitores não são obrigatórios em todos os casos, mas quando usados, devem ser posicionados o mais próximo possível dos pinos de interesse e compartilhar o mesmo terra analógico do microcontrolador.

Alocação de Pinos para ADCx

Para a entrada dos sinais analógicos (tensões), deve-se alocar pinos físicos aos canais do módulo ADCx e configurá-los para a função de "entradas analógicas". A tabela na Seção 10.3.1/página 162 em [1] contém todas as funções multiplexáveis disponíveis para os pinos físicos. Cada pino é designado para servir apenas um canal de entrada, A ou B. Por exemplo, de acordo com a tabela, o pino PTB3 pode ser atribuído à entrada 13 do ADCx se `PORTB_PCR3_MUX`==0x00. Destaca-se aqui que as funções configuráveis para os pinos digitais, controladas pelos *bits* de configuração `PORTx_PCRn_DSE`, `PORTx_PCRn_PFE`, `PORTx_PCRn_PRE` e `PORTx_PCRn_PE` (Seção 11.5.1/página 183 em [1]), não são aplicáveis aos sinais analógicos.

Processamento de Interrupções em ADCx

O conversor ADC opera em conjunto com o controlador NVIC. Quando o *bit* de controle `ADCx_SC1n_AIEN` e o *bit* de estado `ADCx_SC1n_COCO` estão setados em '1', indicando que o resultado da conversão está disponível no registrador de dados `ADCx_Rn` (Seção 28.4.4.2/página 486 em [1]), uma solicitação de interrupção IRQ15 é gerada (Tabela 3-7/página 53 em [1]). Se essa llinha de requisição estiver devidamente habilitada no NVIC, o fluxo de controle é automaticamente desviado para a rotina de serviço associada. O nome da rotina de serviço para IRQ15 é declarada como `ADC0_IRQHandler` no arquivo `Project_Settings/Startup_Code/kinetis_sysinit.c`, gerado pelo IDE *CodeWarrior*. O *bit* de estado `ADCx_SC1n_COCO` é automaticamente resetado em '0' quando ocorre uma leitura do registrador `ADCx_Rn` ou uma escrita em `ADCx_SC1n`.

Calibração de ADCx

Para aumentar a precisão dos valores convertidos, o ADC possui uma função de auto-calibração que pode ser acionada por *software*. Os *bits* de estado `ADCx_SC3_CAL` e `ADCx_SC3_CALF` indicam, respectivamente, o progresso e o resultado do processo da calibração. O término da calibração é também sinalizado pelo *bit* `ADCx_SC1n_COCO` (Seção 28.4.6/página 494, em [1]). Durante a calibração, são gerados valores de compensação para os erros de *offset* e de ganho. A compensação para erros de *offset* é configurada automaticamente em `ADCx_OFS`. Quanto aos valores de compensação para os erros de ganhos, a calibração gera uma série de valores e os armazena nos registradores `ADCx_CLM*` e `ADCx_CLP*`. Esses valores permitem calcular as compensações para ganhos positivos e negativos, os quais podem ser utilizados para configurar os registradores de compensação de erros de ganhos `ADCx_PG` (positivo) e `ADCx_MG` (negativo). Na seção 28.4.6/página 494 em [1] encontra-se um procedimento de calibração recomendado pelo fabricante. Nos projetos `rot9_example1` [17] e `rot9_aula` [11] foi implementado este procedimento de calibração.

Estimativa do Tempo de Conversão em ADCx

Os tempos necessários para realizar uma conversão, que incluem a amostragem e a quantização, são determinados pelo número de ciclos de ADCK e pelo relógio do barramento (*bus clock*). Esses tempos dependem do modo de amostragem configurado nos campos ADC0_CFG1_ADLSMP e ADC0_CFG2_ADLSTS, da velocidade de conversão configurada no campo ADC0_CFG2_ADHSC e da resolução do resultado digital (Seção 28.4.4.5/página 487 em [1]). A fórmula para calcular esses tempos é fornecida na Figura 28-62/página 489 em [1], e os valores de cada termo são especificados pelo fabricante (Figuras 5 a 7). **Estimar o tempo necessário para uma conversão é crucial para determinar a frequência máxima que pode ser aplicada na amostragem dos sinais.**

Para ilustrar a estimativa do tempo de conversão total, com base nos dados apresentados nas Figuras 5 a 7, vamos considerar uma configuração do módulo ADC:

- frequência ADCK: 10485760Hz, sendo a frequência do sinal de barramento 20971520Hz. (ADC0_CFG1_ADIV = 0b01 ou ADC0_CFG1_ADICLK = 0b01),
- resolução de 16 *bits* (ADC0_CFG1_MODE = 0b11),
- tempo de amostragem longo, com 12 ADCK ciclos extra, ativado (ADC0_CFG1_ADLSMP = 1; ADC0_CFG2_ADLSTS = 0b01),
- alta velocidade de conversão ativada (ADC0_ADHSC = 1),
- média ativada para 8 amostras por conversão (ADC0_SC3_AVGE = 1; ADV0_SC3_AVGS = 0b01).

Segundo a tabela na Figura 5, SFCAdder = 3 ciclos de ADCK + 5 ciclos de *bus clock*. Pelas tabelas da Figura 6, AverageNum = 8 e BCT = 25 ciclos de ADCK. Das tabelas da Figura 7, temos LSTAdder = 12 ciclos de ADCK e HSCAdder = 2 ciclos de ADCK. Substituindo esses valores na equação acima, temos

Tempo de conversão total = SFCAdder + AverageNum*(BCT+LSTAdder+HSCAdder)
= (3 ciclos de ADCK + 5 ciclos de bus clock) + 8 *(25+12+2) ciclos de ADCK
= (3 ciclos de ADCK + 5/2 ciclos de ADCK) + 8 *(25+12+2) ciclos de ADCK
= 317,5 ciclos de ADCK
= 317.5*(1/10485760) = 0,000030279s = 30,279us

Além disso, o fabricante fornece os tempos necessários para a amostragem dos sinais, expressos em ciclos do sinal de relógio ADCK (Figura 8). Por exemplo, o tempo de amostragem mostrado na Figura 8 é de aproximadamente 18 ciclos de ADCK, equivalente a cerca de 1,7us. Com base nesses dados, o intervalo total entre a amostragem e a conversão de um valor no registrador ADCx_RA não deve ser inferior a aproximadamente (30,279us+1,7us) para o módulo ADC com a configuração acima.

$$ConversionTime = SFCAdder + AverageNum \times (BCT + LSTAdder + HSCAdder)$$

Figure 28-62. Conversion time equation

Table 28-70. Single or first continuous time adder (SFCAdder)

CFG1[ADLSMP]	CFG2[ADACKEN]	CFG1[ADICLK]	Single or first continuous time adder (SFCAdder)
1	x	0x, 10	3 ADCK cycles + 5 bus clock cycles
1	1	11	3 ADCK cycles + 5 bus clock cycles[1]
1	0	11	5 µs + 3 ADCK cycles + 5 bus clock cycles
0	x	0x, 10	5 ADCK cycles + 5 bus clock cycles
0	1	11	5 ADCK cycles + 5 bus clock cycles[1]
0	0	11	5 µs + 5 ADCK cycles + 5 bus clock cycles

Figura 5: Diferentes tempos para o termo SFCAdder.

Table 28-71. Average number factor (AverageNum)

SC3[AVGE]	SC3[AVGS]	Average number factor (AverageNum)
0	xx	1
1	00	4
1	01	8
1	10	16
1	11	32

Table 28-72. Base conversion time (BCT)

Mode	Base conversion time (BCT)
8b single-ended	17 ADCK cycles
9b differential	27 ADCK cycles
10b single-ended	20 ADCK cycles
11b differential	30 ADCK cycles
12b single-ended	20 ADCK cycles
13b differential	30 ADCK cycles
16b single-ended	25 ADCK cycles
16b differential	34 ADCK cycles

Figura 6: Diferentes quantidade de amostras por resultado digital e diferente tempos de BCT.

Table 28-73. Long sample time adder (LSTAdder)

CFG1[ADLSMP]	CFG2[ADLSTS]	Long sample time adder (LSTAdder)
0	xx	0 ADCK cycles
1	00	20 ADCK cycles
1	01	12 ADCK cycles
1	10	6 ADCK cycles
1	11	2 ADCK cycles

Table 28-74. High-speed conversion time adder (HSCAdder)

CFG2[ADHSC]	High-speed conversion time adder (HSCAdder)
0	0 ADCK cycles
1	2 ADCK cycles

Figura 7: Diferentes tempos de LSTAdder e HDCAdder.

ADC configuration			Sample time (ADCK cycles)	
CFG1[ADLSMP]	CFG2[ADLSTS]	CFG2[ADHSC]	First or Single	Subsequent
0	X	0	6	4
1	00	0	24	
1	01	0	16	
1	10	0	10	
1	11	0	6	
0	X	1	8	6
1	00	1	26	
1	01	1	18	
1	10	1	12	
1	11	1	8	

Figura 8: Diferentes tempos de amostragem.

Módulo LPTMRx

O módulo LPTMR (*Low Power Timer*) é um circuito versátil que pode ser configurado para operar como um contador de tempo interno ou como um contador de pulsos externos de 16 *bits*, mesmo em modos de baixo consumo de energia. A sua operação se baseia em dois sinais de relógio distintos: o sinal de relógio de barramento (*bus clock*), utilizado para os circuitos de comunicação com o barramento, e o sinal `LPTMRx clock`, empregado para o contador (Tabela 5-2/página 121 em [1]). Para habilitar o *bus clock*, utiliza-se o *bit* `SIM_SCGC5_LPTMR` (Seção 12.2.8/página 206 em [1]), enquanto o sinal `LPTMRx clock` pode ser selecionado entre 4 fontes por meio dos *bits* de configuração `LPTMRx_PSR_PCS`: sinal de referência interno `MCGIRCLK`, `LPO` (*`Low Power Oscillator`*) 1kHz, `ERCLK32K`, e sinal de referência externo `OSCERCLK` (Seção 3.8.3.3/página 90 em [1]).

É possível ainda dividir a frequência do sinal `LPTMRx clock` resetando o *bit* de configuração `LPTMRx_PSR_PBYP` em '0' e ajustando o divisor no campo `LPTMRx_PSR_PRESCALE` (Seção 33.3.2/página 591 em [1]), resultando na frequência de operação do contador f_{LPTMR}. Após configurar e habilitar os sinais de relógio de ambos os domínios, é necessário ativar o módulo LPTMR, setando

o *bit* de controle LPTMRx_CSR_TEN (Seção 33.3.1/página 589 em [1]) em '1'.

No modo de operação como **contador de tempo** interno (o *bit* de configuração LPTMRx_CSR_TMS resetado em '0'), o contador LPTMRx_CNR (Seção 33.3.4/página 592 em [1]) realiza uma contagem até o valor configurado no registrador de dados LPTMRx_CMR (Seção 33.3.3/página 592 em [1]). Quando LPTMRx_CNR alcança o valor de LPTMRx_CMR, o *bit* de estado LPTMRx_CSR_TCF é automaticamente setado em '1' (Seção 33.3.1/página 589 em [1]). Posteriormente, o valor do contador LPTMRx_CNR é resetado para '0', caso o *bit* de configuração LPTMRx_CSR_TFC (Seção 33.3.1/página 589 em [1]) esteja em '0'; caso contrário, o LPTMRx_CNR continuará a contar até atingir o estouro (0xFFFF) antes de ser resetado para '0'.

No modo de operação como **contador de pulsos** externos (o *bit* de configuração LPTMRx_CSR_TMS setado em '1'), os pulsos que incrementam o contador LPTMRx_CNR são provenientes de sinais externos configuráveis pelos *bits* de configuração LPTMRx_CSR_TPS. Na Seção 3.8.3.2/página 89 em [1] são listados os códigos binários correspondnetes a esses *bits*, relacionando-os com as funções multiplexáveis dos pinos: 0b00 para os pinos multiplexáveis para a função CMP0_OUT, 0b01 para LPTMR_ALT1, 0b10 para LPTMR_ALT2, e 0b11 para LPTMR_ALT3. Além disso, é É possível configurar a polaridade dos pulsos que incrementam LPTMRx_CNR: ativo-alto ou na borda de subida (se o *bit* de configuração LPTMRx_CSR_TPP estiver resetado em '0') e ativo-baixo ou na borda de descida (se o *bit* de configuração LPTMRx_CSR_TPP estiver setado em '1'). Ao operar como contador de pulsos, os *bits* de configuração LPTMRx_PSR_PBYP e LPTMRx_PSR_PRESCALE funcionam como registradores de configuração de filtros de transientes (*glitches*) antes de iniciar contagem. É importante ressaltar que a frequência desses sinais externos não pode ultrapassar a frequência máxima de operação do contador, f_{LPTMR}=24MHz, conforme especificada na folha de dados técnicos [7].

Configuração de um Período em LPTMRx

Quando LPTMRx é configurado no modo de contador de tempo, o período (contagem máxima) de LPTMR_CNR depende não apenas da frequência da fonte f_{LPTMR} (LPTMR clock) selecionada, mas também dos valores configurados em LPTMRx_CMR (valor de referência para a contagem máxima, Seção 33.3.3/página 592 em [1]) e em LPTMRx_PSR_PRESCALE (divisor *prescaler*, Seção 33.3.2/página 590 em [1]). Para o LPTMRx , devemos diferenciar temos os seguintes casos:

1) LPTMRx_PSR_PBYP==0 e LPTMRx_CSR_TFC == 0

$$Periodo = LPTMRx\_CMR \, x \frac{2^{LPTMRx\_PSR\_PRESCALE+1}}{f_{LPTMR}} \quad (3.a)$$

2) LPTMRx_PSR_PBYP==1 e LPTMRx_CSR_TFC == 0

$$Periodo = LPTMRx\_CMR \, x \frac{1}{f_{LPTMR}} \quad (3.b)$$

3) LPTMRx_PSR_PBYP==0 e LPTMRx_CSR_TFC == 1

$$Periodo = 65535 \, x \frac{2^{LPTMRx\_PSR\_PRESCALE+1}}{f_{LPTMR}} \quad (3.c)$$

4) `LPTMRx_PSR_PBYP==1 e LPTMRx_CSR_TFC == 1`

$$Periodo = 65535 \frac{1}{f_{LPTMR}} \qquad (3.d)$$

Alocação de Pinos para LPTMRx

Para capturar os sinais externos no modo de operação de contador, é necessário alocar um pino físico que esteja multiplexado para a função configurada em `LPTMRx_CSR_TPS`. Para isso, podemos fazer uso da tabela na Seção 10.3.1/página 162 em [1] que nos apresenta as seguintes alternativas: `CMP0_OUT (PTC0, PTC5, PTE0)`, `LPTMR0_ALT1 (PTA19)` e `LPTMR0_ALT2 (PTC5)`.

Processamento de Interrupções em LPTMRx

Quando tanto o *bit* de estado `LPTMRx_CSR_TCF` quanto o *bit* de controle `LPTMRx_CSR_TIE` estiverem setados em '1', é gerado um evento de interrupção IRQ 28 para o controlador NVIC (Tabela 3-7/página 54 em [1]). Se essa linha de requisição estiver devidamente habilitada no NVIC, o fluxo de controle é automaticamente desviado para a rotina de serviço. Ao consultar o arquivo `Project_Settings/Startup_Code/kinetis_sysinit.c` gerado pelo IDE *CodeWarrior*, podemos encontrar o nome da rotina de serviço declarado para IRQ28, que é `LPTimer_IRQHandler`. O *bit* de estado `LPTMRx_CSR_TCF` pode ser resetado para '0' se fizermos um acesso de escrita no *bit* (*write-1-to-clear*) ou se desabilitarmos o módulo (Seção 33.4.7/página 596 em [1]).

Disparos de LPTMRx para outros Módulos

Os eventos detectados pelo *bit* de estado `LPTMRx_CSR_TCF` atuam como fontes de disparos para outros módulos, sem necessidade de intervenção de *software*. Quando `LPTMRx_CMR` é configurado com um valor diferente de 0 e `LPTMRx_CSR_TCF` estiver definido como '1', ocorre um disparo no próximo incremento no contador `LPTMRx_CNR` quando a contagem desse contador estoura (Seção 33.4.6/página 596 em [1]).

Reuso de Estruturas Predefinidas

Dada a quantidade considerável de parâmetros (campos) distribuídos em 5 registradores para configurar a operação de um módulo ADCx, uma abordagem em C é definir um novo tipo de dado que ofereça uma configuração mais centralizada para ADCx. No projeto `rot7_aula` [12] é criado o tipo de dado `struct _UART0Configuration_tag`, cujos membros correspondem aos campos dos registraddores de configuração/controle conforme especificado pelo fabricante. No entanto, uma alternativa sugerida na seção 11.2.1/página 117 em [5] permite reusar as estruturas e macros já disponíveis no IDE *CodeWarrior*.

No arquivo `Project_Headers/MKL25Z.h`, é definida para cada módulo uma nova estrutura que acessa diretamente os endereços físicos de memória por meio de nomes dos registradores conforme documentados nos manuais. Para o módulo ADCx, é definidio o tipo de dado `struct ADC_MemMap` e o respectivo ponteiro `ADC_MemMapPtr`, cujos membros são registradores em vez

dos campos dos registradores. Os nomes dos membros correspondem aos mesmos nomes dos registradores adotados pelo fabricante (Seção 28.3/página 461 em [1]), com exceção dos dois registradores de controle de entrada `ADC0_SC1A` e `ADC0_SC1B`, e dos dois registradores de dados de saída, `ADC0_RA` e `ADC0_RB`. Em vez de declarar quatro membros, são definidos dois vetores de 2 elementos, `SC1[2]` e `R[2]`, de modo que `SC1[0]`, `SC1[1]`, `R[0]` e `R[1]` correspondem, respectivamente, a `ADC0_SC1A`, `ADC0_SC1B`, `ADC0_RA` e `ADC0_RB`:

```
typedef struct ADC_MemMap {
  uint32_t SC1[2];
  uint32_t CFG1;
  uint32_t CFG2;
  uint32_t R[2];
  uint32_t CV1;
  uint32_t CV2;
  uint32_t SC2;
  uint32_t SC3;
  uint32_t OFS, PG, MG;
  uint32_t CLPD, CLPS;
  uint32_t CLP4, CLP3, CLP2, CLP1, CLP0;
  uint8_t RESERVED_0[4];
  uint32_t CLMD, CLMS;
  uint32_t CLM4, CLM3, CLM2, CLM1, CLM0;
} volatile *ADC_MemMapPtr;
```

Para configurar o conteúdo de cada campo do registrador, o desenvolvedor é responsável por aplicar operações *bit*-a-*bit*. Por exemplo, no projeto `rot9_example1` [17], foi declarada em `main.c` a variável `Master_Adc_Config` do tipo `struct ADC_MemMap`. Para inicializar os valores dos registradores `ADC0_SC1A`, `ADC0_SC1B`, `ADC0_CFG1`, `ADC0_CFG2`, `ADC0_CV1`, `ADC0_CV2`, `ADC0_SC2` e `ADC0_SC3`, usa-se o operador lógico *bit*-a-*bit* OU (|) para combinar os valores especificados separadamente em cada campo de um registrador antes de atribuí-los ao registrador. Esses valores são definidos utilizando-se macros `ADC_*` encontradas em `Project_Headers/MKL25Z.h`. Por exemplo, uma expressão de macros para definir os valores dos campos do registrador CFG1

`(0<<7)|ADC_CFG1_ADIV(0b10)|ADC_CFG1_ADLSMP_MASK| ADC_CFG1_MODE(0b11)| ADC_CFG1_ADICLK(0b00)`

equivale a

(0<<7)|(0b10<<5)|(1<<4)|(0b11<<2)|(0b00)

que corrresponde à seguinte palavra:

01011100

Fazendo atribuições análogas a outros registradores, podemos inicilizar todos os registradores de configuração do módulo ADC com o seguinte comando:

```
struct ADC_MemMap Master_Adc_Config = {
          .SC1[0]=(0<<6)        //AIEN
```

```
            | (0<<5)              //DIFF
            | ADC_SC1_ADCH(31),
           .SC1[1]=(0<<6)         //AIEN
            | (0<<5)              //DIFF
            | ADC_SC1_ADCH(31),
           .CFG1=(0<<7)           //ADLPC
            | ADC_CFG1_ADIV(0b10)
            | ADC_CFG1_ADLSMP_MASK
            | ADC_CFG1_MODE(0b11)
            | ADC_CFG1_ADICLK(0b00),
           .CFG2=(0<<4)          //MUXSEL
            | (0<<3)             //ADACKEN
            | ADC_CFG2_ADHSC_MASK
            | ADC_CFG2_ADLSTS(0b00),
           .CV1=0x1234u,
           .CV2=0x5678u,
           .SC2=ADC_SC2_ADTRG_MASK
            | (0<<5)               //ACFE
            | ADC_SC2_ACFGT_MASK
            | ADC_SC2_ACREN_MASK
            | (0<<2)               //DMAEN
            | ADC_SC2_REFSEL(0b00),
           .SC3= (0<<7)            // CAL
            | (0<<3)               // ADCO
            | ADC_SC3_AVGE_MASK
            | ADC_SC3_AVGS(0b11),
};
```

Essas operações *bit*-a-*bit* na inicialização dos valores dos registradores permitem que os seus resultados sejam atribuídos diretamente aos endereços da memória onde estão mapeados os registradores físicos do módulo ADC. Em vez de operações a nível de *bits*, implementadas em `UART0_configure` no projeto `rot7_aula` [12], comandos de atribuição dos valores dos registradores setados em `Master_Adc_Config` ao bloco de memória onde estão mapeados os registradores do módulo ADCx, são suficientes para transferir todos os dados inicializados numa estrutura temporária (`Master_Adc_Config`) aos registradores do módulo ADC, como demonstra a seguinte função definida em `rot9_example1` [17]:

```
void ADC_Config_Alt (ADC_MemMapPtr end, ADC_MemMapPtr dados) {
     end->SC1[0] = dados->SC1[0];
     end->SC1[1] = dados->SC1[1];
     end->CFG1 = dados->CFG1;
     end->CFG2 = dados->CFG2;
     end->CV1 = dados->CV1;
     end->CV2 = dados->CV2;
     end->SC2 = dados->SC2;
     end->SC3 = dados->SC3;
     return;
}
```

A transferência dos dados inicializados na variavel `Master_Adc_Config` para o bloco de endereço `(ADC_MemMapPtr)0x4003B000u)` onde estão mapeados os registradores de ADC0 (Seção 28.3/página 461 em [1]) pode ser realizada com a chamada

```
ADC_Config_Alt ((ADC_MemMapPtr)0x4003B000u), &Master_Adc_Config);
```

Podemos aumentar ainda mais a legibilidade do código utilizando uma macro definida para esse endereço inicial do bloco de memória em `Project_Headers/MKL25Z.h`

```
#define   ADC0_BASE_PTR                ((ADC_MemMapPtr)0x4003B000u)
```

Então, podemos usar essa macro na chamada da função para transferência do conteúdo de `Master_Adc_Config` aos registradores de ADC, como a implementação em `rot9_example1` [17]:

```
ADC_Config_Alt (ADC0_BASE_PTR, &Master_Adc_Config).
```

Definição de Macros das Macros

Para melhorar a legibilidade dos nossos códigos, podemos criar novas macros que refletem melhor o significado de cada código binário, como as definidas em `ADC.h` no projeto `rot9_example1`[17]:

```
#define ADLSTS_20 0b00
#define DMAEN_ENABLED ADC_SC2_DMAEN_MASK
```

Muitas novas macros em `ADC.h` são essencialmente uma espécie de renomeação das macros existentes. Durante a fase de pré-processamento C, elas são substituídas recursivamente até os comandos compiláveis de C. Com uso das novas macros, o código de inicialização da variável `Master_Adc_Config` tende a ser mais autoexplicativo:

```
struct ADC_MemMap Master_Adc_Config = {
          .SC1[0]=AIEN_OFF
           | DIFF_SINGLE
           | ADC_SC1_ADCH(31),
          .SC1[1]=AIEN_OFF
           | DIFF_SINGLE
           | ADC_SC1_ADCH(31),
          .CFG1=0x00
           | ADC_CFG1_ADIV(ADIV_4)
           | ADLSMP_LONG
           | ADC_CFG1_MODE(MODE_16)
           | ADC_CFG1_ADICLK(ADICLK_BUS),
          .CFG2=MUXSEL_ADCA
           | ADACKEN_DISABLED
           | ADHSC_HISPEED
           | ADC_CFG2_ADLSTS(ADLSTS_20),
          .CV1=0x1234u,
          .CV2=0x5678u,
          .SC2=ADTRG_HW
           | ACFE_DISABLED
```

```
              | ACFGT_GREATER
              | ACREN_ENABLED
              | DMAEN_DISABLED
              | ADC_SC2_REFSEL(REFSEL_EXT),
            .SC3=CAL_OFF
              | ADCO_SINGLE
              | AVGE_ENABLED
              | ADC_SC3_AVGS(AVGS_32),
};
```

Filtragem Exponencial para Suavização de Dados Digitalizados

A filtragem exponencial é um método de suavização de séries temporais digitais que combina uma média ponderada do valor medido x_i no instante i e dos resultados anteriores x_{i-1} … x_0 para produzir um resultado y_i suavizado no instante i

$$y_i = \alpha x_i + (1-\alpha)\, y_{i-1} = \alpha x_i + (1-\alpha)\,\alpha x_{i-1} + \ldots + (1-\alpha)^{i-1}\,\alpha x_1 + (1-\alpha)^i \alpha x_0 \quad (4)$$

O peso α dado a cada valor anterior decai ao longo do tempo, permitindo que a resposta do filtro a mudanças recentes seja mais rápida do que a resposta a mudanças antigas. Isso resulta em suavização dos dados, minimizando o ruído e destacando tendências subjacentes.

No projeto `rot9_example1`[17] é aplicada uma filtragem exponencial com α=0.5, pois os comandos em `ADC0_IRQHandler` (`ISR.c`):

```
exponentially_filtered_result += result0A;
exponentially_filtered_result /= 2;
```

somam os resultados anteriores acumulados na variável estática `exponentially_filtered_result` com o valor amostrado `result0A` e divide a soma por 2:

$$exponentially\_filtered\_result = \frac{exponentially\_filtered\_result + result0A}{2}$$

$$exponentially\_filtered\_result = 0.5 \times result0A + (1-0.5) \times exponentially\_filtered\_result$$

A técnica de filtragem exponencial é amplamente utilizada em sistemas embarcados devido à sua simplicidade de implementação e eficácia na redução de ruídos e flutuações nos dados. Além disso, essa técnica requer poucos recursos de *hardware*, tornando-a adequada para implementação em sistemas embarcados com restrições de processamento e energia. Em muitos sistemas embarcados, especialmente aqueles que lidam com sensores ou outras formas de entrada de dados, é comum encontrar variações nos valores amostrados devido a ruídos elétricos, interferências externas ou imprecisões nos sensores. A filtragem exponencial ajuda a suavizar essas variações, tornando os dados mais estáveis e confiáveis para análise ou tomada de decisões.

Interpretação dos Valores Amostrados

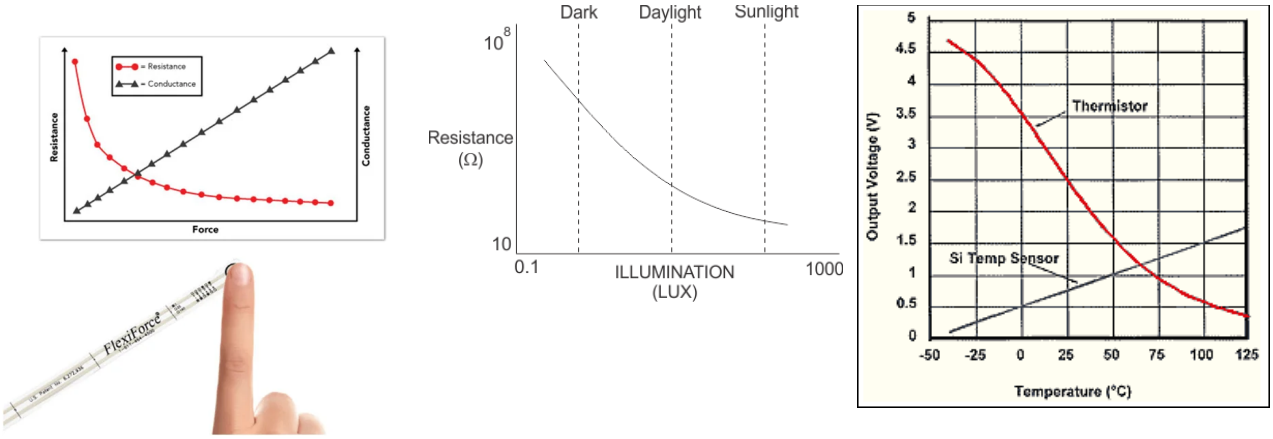
É importante ressaltar que o resultado de uma conversão, acessível pelo registrador `ADCx_Rn`, é uma representação binária em N *bits* do valor de tensão amostrado. Quando devidamente calibrado, podemos considerar que o valor de tensão no intervalo `[VREFL,VREFH]` é diretamente proporcional ao código binário entre 0 e 2^N-1. Dessa forma, o valor da amostra de tensão,

`Tensão_amostrada`, pode ser obtido a partir do código binário armazenado em `ADCx_Rn` por uma regra de três simples usando **operações em ponto flutuante**:

(`Tensão amostrada-VREFL`) → [`ADCx_Rn`]

(`VREFH – VREFL`) → 2^N-1

Se for necessário obter os valores originais das grandezas físicas dos sinais amostrados, é essencial realizar um pós-processamento das amostras de tensão recuperadas, `Tensão_amostrada`, convertendo-as para os valores nas grandezas físicas originais. Isso geralmente requer consulta aos *datasheets* dos fabricantes dos sensores, como exemplifica na Figura 9.



Sensor de força [8] Sensor de luz [9] Sensor de temperatura [10]

Figura 9: Relações entre as grandezas físicas e elétricas geradas por diferentes sensores.

Para o sensor de temperatura AN3031 integrado no KL25Z, e disponível no canal `0b11010` do módulo ADC, o fabricante especifica a relação entre a temperatura medida e a tensão gerada por meio da expressão (Seção 2.1/página 3 em [3]):

$$Temperatura = 25 - \left(\frac{V_{Temperatura} - V_{25}}{m}\right) \qquad (5)\text{, onde}$$

$$m = 1.646\ \text{V/ºC},\ \text{se}\ V_{Temperatura} \geqslant V_{25}\ (\ Temperatura \leqslant 25ºC)$$

$$m = 1.769\ \text{V/ºC},\ \text{se}\ V_{Temperatura} < V_{25}\ (\ Temperatura > 25ºC)$$

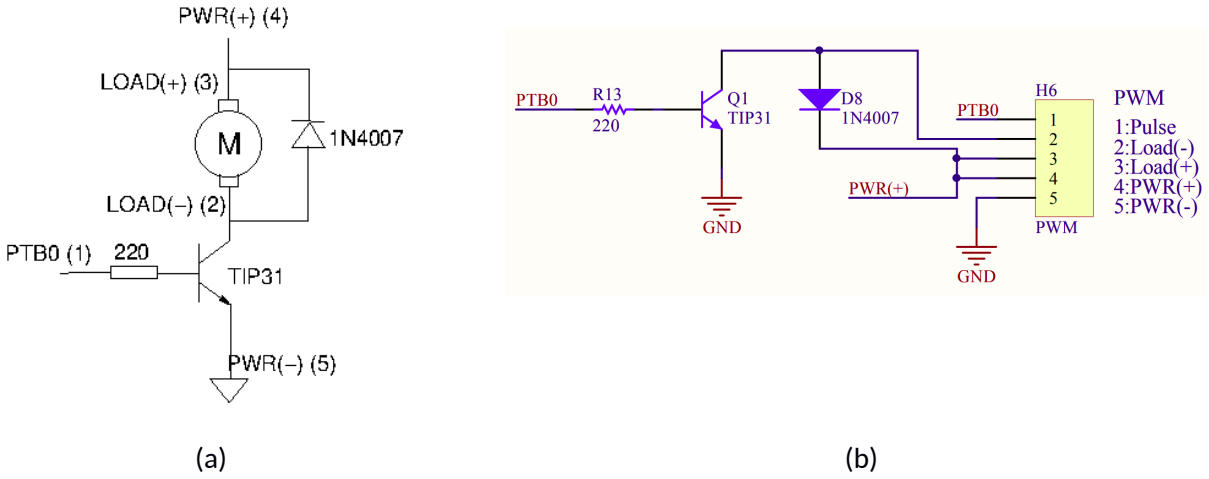
$$V_{25} \sim 0.703125V$$

Essa relação nos permite estimar a temperatura com base no valor de tensão amostrado pelo ADC.

PWM em Controle de Transferência de Potência

O projeto `controle_cooler` [20] utiliza sinais para ajustar a velocidade de rotação de um *cooler*, variando conforme o ângulo de giro do eixo do potenciômetro. A potência aplicada ao *cooler* está diretamente ligada à sua velocidade de rotação. O controle dessa potência é alcançado através de um sinal PWM, uma abordagem simples e eficaz para controlar a potência de uma carga [4]. No entanto, os microcontroladores geralmente fornecem uma corrente muitot baixa para alimentar diretamente a maioria das cargas.

Uma solução amplamente empregada é injetar o sinal PWM na base de um transistor de potência (Figura 10(a)), que funciona como uma chave eletrônica em um circuito capaz de fornecr correntes maiores. Esse transistor fecha o circuito quando está no estado de saturação (pulso no nível 1) e abre quando está no estado de corte (pulso no nível 0), permitindo que a carga receba uma fração de potência máxima que receberia com a alimentação contínua. Na Figura 10(a), o diodo 1N4007 em paralelo com o motor protege-o de tensões reversas (que podem ocorrer quando a corrente através de uma indutância é interrompida subitamente), enquanto a resistência R13 limita a corrente na base do transistor. Uma parte desse circuito está integrada no *shield* FEEC871, como mostrado em um recorte do esquemático do *shield* [6] na Figura 10(b). Ao conectar uma carga, como um *cooler*, entre os pinos 2 e 3 do *header* H6, e uma fonte de alimentação de 12V DC entre os pinos 4 e 5 do mesmo *header*, é possível controlar a velocidade do *cooler* por meio de um sinal PWM gerado no pino PTB0 do KL25Z. Note na Figura 10(b) que o sinal do PTB0 é acessível no pino 1 do *header* H.



(a) (b)

Figura 10: Controle da velocidade de um motor por PWM.

Os pinos marcados com as letras A, B, C, D e E na Figura 11, correspondem aos pinos 5, 4, 3, 2 e 1 na figura 10(b).

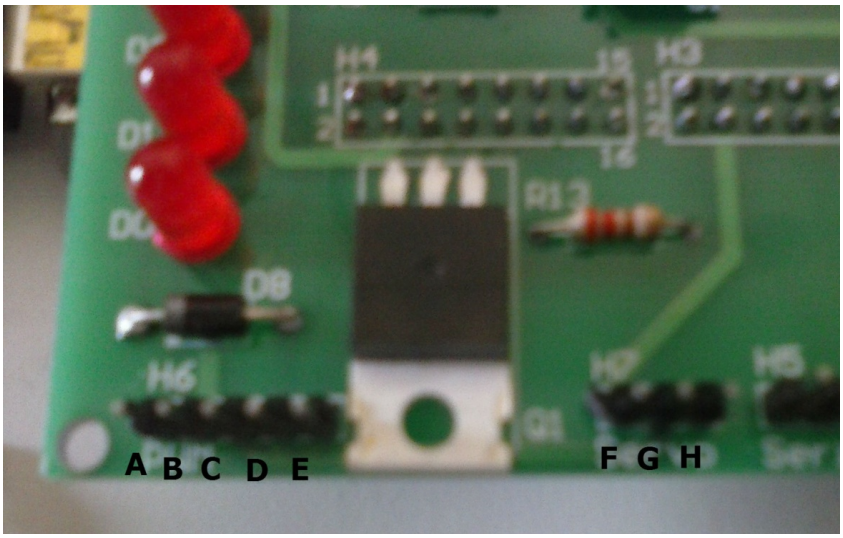


Figura 11: Pinagens em *Shield* FEEC871.

EXPERIMENTO

Este experimento aborda o desenvolvimento do projeto `controle_cooler`, onde o controle da velocidade do *cooler* é realizado por meio de um potenciômetro conectado ao pino PTB1 (pino 1 do *header* H7) do *shield* FEEC871 (Figura 12). Esse potenciômetro é multiplexado ao canal `0b01001` do módulo ADC. O valor do potenciômetro, amostrado periodicamente com uma resolução de 12 *bits* e **sujeito a um processo de filtragem exponencial** com α=0.5. Esse valor filtrado é então usado para atualizar a largura de pulso do sinal gerado no canal `TPM1_CH0`, que é então transferido para o pino PTB0 (Figura 10). O ciclo de trabalho desse canal é exibido no formato "DUTY: YY.YY" no meio da primeira linha do visor do LCD.

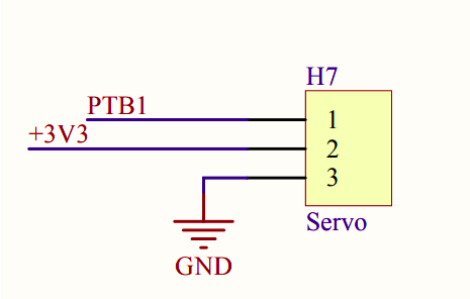


Figura 12: Pinagem do *header* H7 do shield FEEC871.

Além disso, a temperatura do microcontrolador é monitorada periodicamente pelo sensor de temperatura AN3031 integrado no KL25Z. O valor amostrado é convertido para graus Celsius e exibido no formato "TEMP: XX.XX C" no meio da segunda linha do LCD. As conversões periódicas do sinal do sensor de temperatura são iniciadas por *hardware*, pelos eventos de estouro do módulo TPM1. Em seguida, por *software*, é disparada a conversão do sinal amostrado pelo potenciômetro.

O LED R externo, conectado ao *header* H5, oferece uma representação visual da temperatura do núcleo do KL25Z. Apenas a intensidade luminosa vermelha (`TPM1_CH1`) é ativada e essa intensidade varia linearmente conforme a temperatura amostrada, indo de 15°C (LED apagado) até 50°C (brilho máximo).

Na Figura 13, são mostrados os 3 estados do sistema: `AMOSTRA_TEMP`, `AMOSTRA_VOLT` e `ATUALIZAÇÃO`. Nos bastidores desses estados, o temporizador TPM1 gera disparos periódicos, com intervalos aproximados de 500*tempo de conversão do módulo de ADC (detalhado no item 2.7). Esses disparos acionam a amostragem e conversão do sinal proveniente do sensor de temperatura AN3031 no estado `AMOSTRA_TEMP`. Após a coleta do valor amostrado pelo sensor de temperatura, o sistema avança para o estado `AMOSTRA_VOLT`, onde o sinal analógico proveniente do potenciômetro é amostrado. Após a conclusão da amostragem dos dois sinais analógicos, o sistema entra no estado de `ATUALIZAÇÃO`. Nesse estado, o LED azul, a temperatura e o valor do potenciâometro amostrados são atualizados no LCD e a velocidade do *cooler* ajustada. Ao finalizar as atualizações, o sistema retorna ao estado `AMOSTRA_TEMP`, onde aguarda o próximo disparo do TPM1 para iniciar um novo ciclo de amostragens.

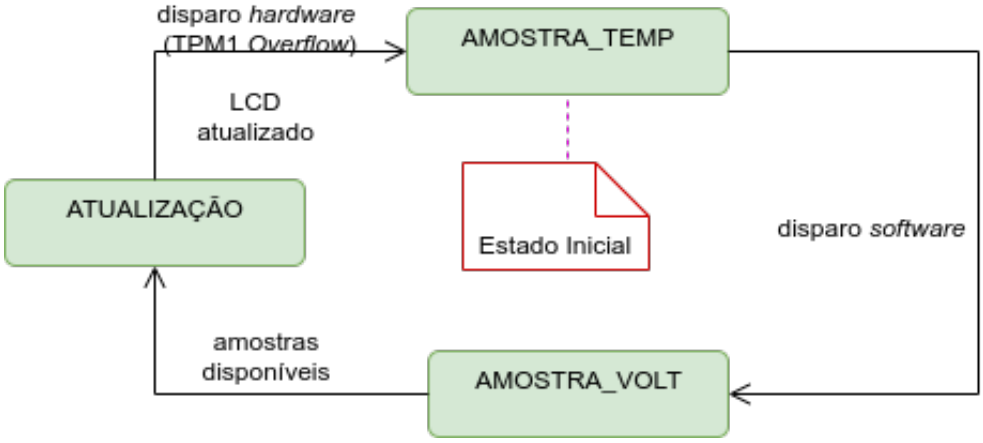


Figura 13: Diagrama de máquina de estados do `controle_cooler` (editado em [14]).

Na Figura 14 é apresentado um diagrama de componentes proposto para este projeto.

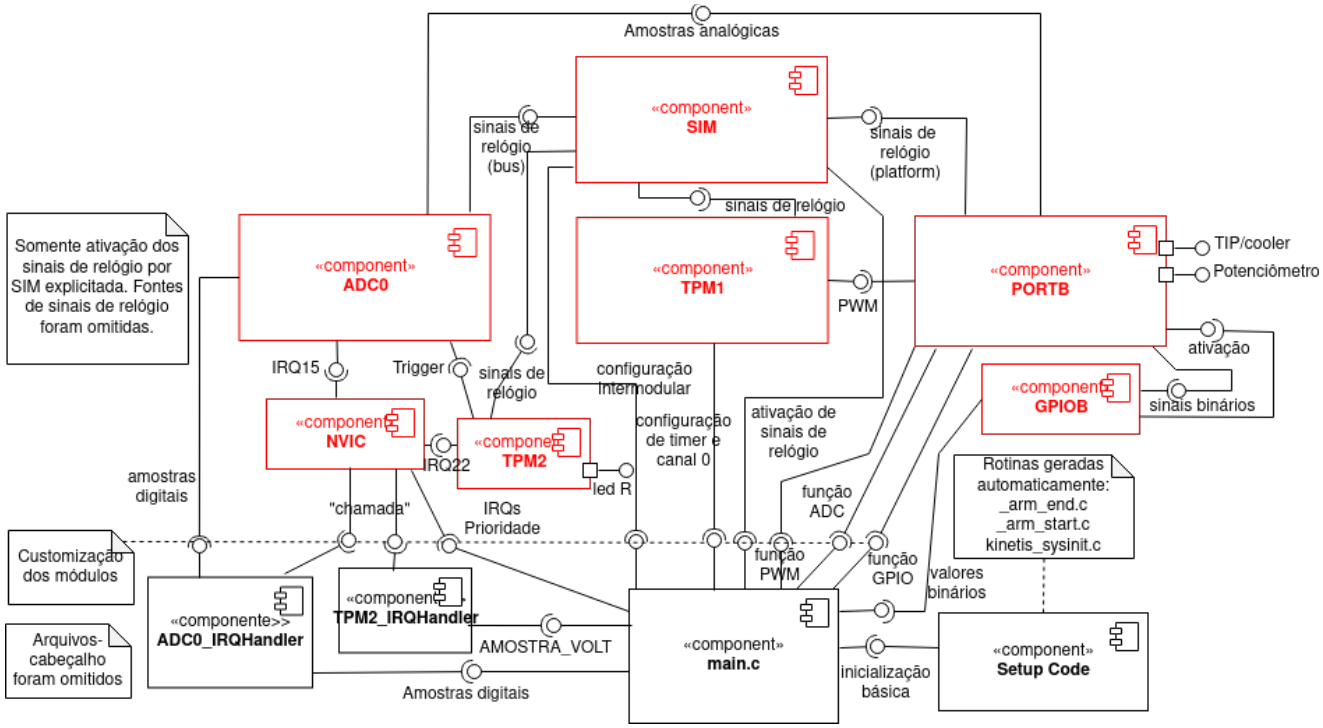


Figura 14: Diagrama de componentes do `controle_cooler` (por legibilidade, foram omitidos os módulos PORTC e GPIOC que controlam LCD, editado em [14]).

O **modo de operação do módulo ADC** especificado é

> **disparo por *hardware* usando o evento TPM1 *Overflow*** (amostragem da tensão do potenciômetro) e por *software* (amostragem do sensor de temperatura)
>
> **frequência ADCK**: 5242880Hz, sendo a frequência do sinal de barramento e a frequêncoa da fonte de ADICK 20971520Hz.
>
> **resolução de 12 *bits***
>
> tempo de amostragem **curto (sem circuito RC na entrada)**
>
> velocidade **normal** de conversão habilitada
>
> média habilitada para 8 **amostras por conversão**

Para facilitar o mapeamento dos valores amostrados do potenciômetro em "ciclos de trabalho", `TPM1_MOD` é 4095 (0x0FFF) em 12 *bits*.

Segue-se um roteiro para o desenvolvimento do projeto. Usa-se na implementação do projeto as macros do arquivo-cabeçalho `derivative.h`.

1 **De conceitos para práticas:** O projeto `rot9_cooler` [20] tem como propósito verificar o funcionamento do *cooler*, além de praticar as conexões físicas do *cooler* com o *shield* FEEC871 e controlar sua rotação ao modificar o ciclo de trabalho. Por outro lado, o projeto `rot9_aula` [11] demonstra a configuração de um módulo ADCx e tratamento de interrupções geradas por ele.

1.a **PWM em Controle de Transferência de Potência:** Conecte o *cooler* nos pinos 2 e 3 do *header* H6 do *shield* FEEC871 e a fonte de alimentação nos pinos 4 (fio vermelho) e 5 (fio preto), mostrados na Figura 10. Ligue a fonte a uma tomada da bancada. Conecte o pino 1 do *header* H6 num canal do analisador. Este pino espelha o sinal PWM que alimenta a base do transistor TIP31. Execute o projeto `rot9_cooler` [20].

No programa foi setado um valor igual a (3.*`TPM1_MOD`)/4 em `TPM1_C0V`. Quais serão os efeitos sobre a forma de onda no analisador lógico e na rotação do *cooler* se reduzirmos o valor de `TPM1_C0V` para 0, `TPM1_MOD`/4 e `TPM1_MOD`/2 e (`TPM1_MOD`+1)?

1.b **Projeto baseado em ADC:** O projeto `rot9_aula` [11] ilustra configurações do módulo ADC para amostrar periodicamente dois sinais analógicos: um proveniente do potenciômetro (canal 9) conectado ao *header* H7 (Figura 12) e outro do sensor AN3031 (canal 26). O módulo PIT gera eventos de estouro, os quais são programados para iniciar a conversão de um sinal amostrado do potenciômetro e em seguida, do sensor AN3031. Os códigos binários dos resultados, em 8 *bits*, são mostrados nos 8 LEDs vermelhos. Além disso, o tempo de amostragem e conversão do módulo ADC, bem como o período configurado para o PIT, são refletidos como pulsos nos pinos 3 e 2 do *header* H5 do *shield* FEEC871, respectivamente.

1.b.1 **Módulo ADC**: São configuráveis a frequência do sinal de relógio ADCK, a resolução, a velocidade da sequência de conversão, fluxo de conversão, e a quantidade de amostras por conversão.

Qual é a frequência do sinal ADCK configurada? Justifique com base na fonte de sinais de relógio configurada para ADC0 e nos valores configurados em `ADC0_CFG1_ADIV` e `ADC0_CFG1_ADICLK`.

Qual é a resolução configurada, em quantidade de *bits*? Justifique com base no valor setado em `ADC0_CFG1_MODE`.

Qual é a velocidade configurada para uma sequência de conversão? Justifique com base no valor setado em `ADC0_CFG2_ADHSC`.

Qual é o fluxo de conversão configurado? Justifique com base no valor setado em `ADC0_SC3_ADCO` e `ADC0_SC3_AVGE`.

Qual é a quantidade de amostras configurada para cada conversão? Justifique com base no valor setado em `ADC0_SC3_AVGE` e `ADC0_SC3_AVGS`.

Verifique se os canais 9 e 26 são usados para amostrar os sinais de potenciômetro e do sensor de temperatura, respectivamente, através dos valores configurados em `ADCx_CFG2_MUXSEL` e `ADCx_SC1A_ADCH`.

1.b.2 **Funções de Comparação em ADC**: É possível remover os resultados de conversão que não sejam de interesse ao configurarmos os intervalos de valores de interesse.
A função de comparação não está habilitada em `rot9_aula`. Para verificar esta afirmação, examine os valores configurados nos registradores `ADC0_SC2`, `ADC0_CV1` e `ADC0_CV2`. Habilite a função, recompile o executável e execute o programa. Defina um ponto de parada na linha "`valor[0] = ADC0_RA;`" e na linha "`valor[1] = ADC0_RA;`" da rotina de serviço `ADC0_IRQHandler`. Analise os diferentes resultados de conversão em `ADC0_RA` ao dar um giro completo no eixo do potenciômetro.

1.b.3 **Configuração de Disparos de Conversão por *Hardware***: Eventos de interrupções podem servir como disparadores para iniciar tarefas em outros módulos através das comunicações intermodulares no KL25Z. É importante ativar o disparador no momento apropriado para evitar interrupções indesejadas. Identifique em `main.c` e na função `ADC_PTB1_config_basica` (ADC.c) os blocos de instruções responsáveis pela configuração de conversões únicas em ADC0 disparadas periodicamente pelos eventos de estouro do PIT.
Para qual dos dois sinais, do sensor de temperatura e do potenciômetro, a conversão é iniciada por *software*? E para qual das duas é por *hardware*?
Quando são iniciados os disparos periódicos?

1.b.4 **Filtragem de Dados Analógicos:** Qual é o tempo de amostragem configurado? Justifique com base nos valores setados em `ADC0_CFG1_ADLSMP` e `ADC0_CFG1_ADLSTS`. Supondo que a impedância de entrada no pino PTB1 seja baixa, dentro da faixa de impedâncias de entrada recomendada, qual configuração de tempo de amostragem você selecionaria? Dica: Leia a recomendação do fabricante sobre a configuração de tempo de amostragem na Seção 28.3.2/página 466 em [1].

1.b.5 **Alocação de pinos para ADCx:** O pino PTB1 é usado para entrada do sinal analógico ao canal 9 de ADC0. Quais configurações foram feitas para habilitar este pino para esta função? Para filtrar ruídos nos sinais, você setaria em '1' o *bit* `PORTB_PCR1_PFE` em vez de usar um circuito externo RC? Justifique. Dica: Leia a função do *bit* PORTx_PCRn_PFE na Seção 11.5.1/página 184 em [1].

1.b.6 **Processamento de Interrupções em ADCx:** As conversões disparadas por *hardware* (PIT), e as conversões disparadas por *sofware* são alternadas na rotina de serviço.
Quais são as configurações necessárias para que o módulo ADCx gere eventos de interrupção e que esses eventos sejam tratados pelo NVIC? Responda com base nas interuções programadas em `ADC_habilitaNVICIRQ` e em `ADC_habilitaInterrupCOCO`.
Qual é o evento de interrupção habilitado em ADC0 para solicitação de interrupção?
Dentro da rotina de serviço, a leitura do resultado de uma conversão é realizada a partir do registrador `ADC0_RA`, seguida pela alternância do modo de disparo da próxima conversão. Identifique os trechos de código responsáveis por essa alternância do modo de disparo.
Por que não há uma instrução de escrita `w1c` na *flag* de interrupção dentro da rotina de serviço `ADC0_IRQHandler`?

1.b.7 **Calibração de ADCx**: Na função `ADC_Cal`, o processo de calibração requer um tempo de processamento para conclusão. Essa conclusão pode ser detectada de duas maneiras: por *polling*, onde a função aguarda ativamente até que a calibração seja concluída, ou por interrupção, onde o

programa é notificado quando a calibração é concluída. Qual estratégia de detecção da conclusão foi implementada na função `ADC_Cal`?

1.b.8 **Reuso de Estruturas Predefinidas**: Como os valores iniciais são definidos nos membros da variável `Master_Adc_Config`? E como esses valores são copiados para os registradores do módulo ADC durante a execução da função `ADC_Config_Alt`?

1.b.9 **Estimativa do Tempo de Conversão em ADCx:** Execute o projeto e capture os sinais usando o analisador lógico. Formas de onda que se assemelham às mostradas na Figura 15 devem ser renderizadas na tela.

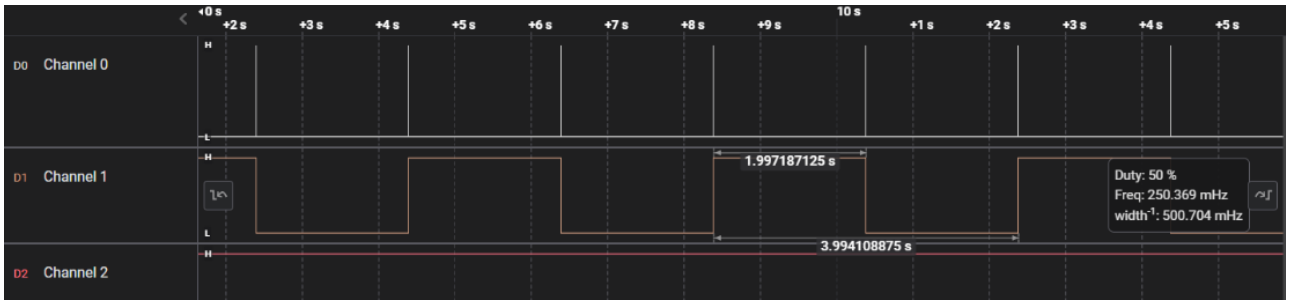


Figura 15: Formas de onda no pino 3 (*Channel* 0) e 2 (*Channel* 1).

Se passarmos da escala de segundos para milisegundos, veremos que os "palitos" brancos no *Channel* 0 são de fato dois pulsos de largura de centenas de micro-segundos que correpondem aos tempos de amostragem e de conversão de uma amostra, primeiro do canal 9 e depois do canal 26.

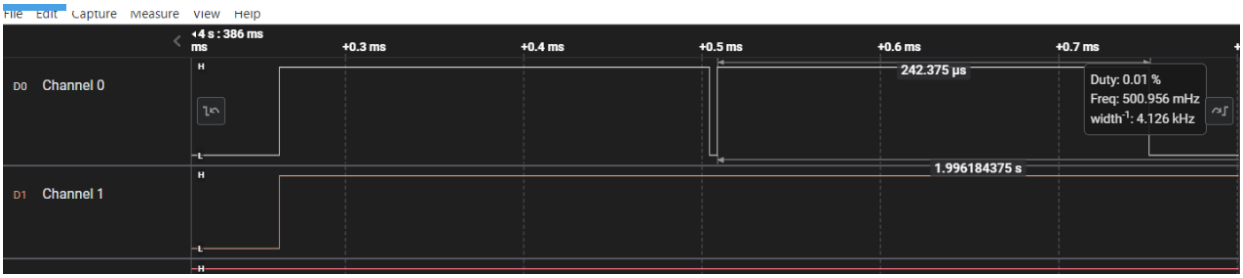


Figura 16: Tempo de amostragem e de conversão de duas amostras no ADC.

Qual período é configurado para o PIT? É condizente com os valores medidos pelo analisador?
Estime o tempo de conversão para a configuração do ADC0 programada em `rot9_aula`. É condizente com os valores medidos pelo analisador?
Qual é o menor período configurável para o PIT para o qual não ocorram sobreposições das conversões?
Estime o tempo de conversão especificado para o ADC0 no projeto `controle_cooler`. Reconfigure o módulo ADC0 de `rot9_aula` com esses valores de configuração, inclusive o valor de `SIM_CLOCKDIV_OUTDIV4`, para verificar o tempo estimado com um analisador lógico. São condizentes com a configuração especificada?

2 **Aprender com os Exemplos dos Manuais**: Na seção 11.2/página 117 em [5], é apresentado um exemplo de configuração do módulo ADC para amostrar o sinal analógico de um potenciômetro integrado ao *kit* dos autores. O potenciômetro é conectado ao canal 4 da entrada B de ADC0. Os eventos de estouro do temporizador LPTMR0 são usados como disparos periódicos de conversões únicas, sem intervenções de *software*. A fonte de sinais de relógio do contador de LPTMR0 é

escolhida como o sinal externo RTC_CLKIN (Seção 5.7.4/página 124 em [1]), cuja entrada é o pino PTC1 (Seção 10.3.1/página 162 em [1]). Devido às **diferenças de ambiente**, foram feitos três ajustes na implementação do exemplo dado em `rot9_example1` [17] para adaptá-lo aos *kits* de desenvolvimento do LE30,

- Como em FRDMKL25Z não há um potenciômetro integrado, foi usado um potenciômetro externo conectado no pino PTB1, multiplexável ao canal 9 de ADC0 (Seção 10.3.1/página 162 em [1]),
- Sendo PTC1 ocupado por um pino de LCD/*Latch*, usou-se LPO como a fonte de sinais de relógio do contador do LPTMR0 (Seção 5.7.4/página 124 em [1]), e
- Não dispondo de um *led* laranja, usou-se um *led* verde conectado no pino PTB19.

Diferentes dos outros projetos que analisamos nos roteiros anteriores, os autores apresentaram apenas um esboço de configuração dos módulos neste projeto. Foi necessário fazer ajustes na **compatibilidade de códigos**, implementando as funções `ADC_Config_Alt`, `ADC_Cal` e `GPIO_initLedG`, em `rot9_example1`. Vale destacar que as instruções de configuração do módulo LPTMR foram alteradas ligeiramente para que elas sejam operações por *bits*.

Destacam-se neste exemplo o uso do LPTMR0, que é um temporizador interno/contador externo de baixo consumo de energia, e o emprego da técnica de filtragem exponencial para suavização de dados digitalizados.

Compile o projeto `rot9_example1`. Em seguida, execute o projeto no modo *Debug* do IDE *CodeWarrior*.

2.a **Filtragem Exponencial para Suavização de Dados Digitalizados**: Destaque as instruções que implementam essa filtragem exponencial dos resultados de conversões.

2.b **Módulo LPTMR**: Qual é o modo de operação configurado para LPTMR0? Justifique com base no valor setado em `LPTMR0_CSR_TMS`.

2.c **Configuração de um Período em LPTMR:** Qual é o período configurado em LPTMR0? Justifique com base na fonte do sinais de relógio selecionada, nos valores setados em no valor setado em `LPTMR0_PSR_PRESCALE`, `LPTMR0_PSR_PBYP`, `LPTMR0_PSR_PCS` e `LPTMR0_CMR`.

2.d **Alocação de pinos para LPTMRx**: Analise a função `LPTMR_config_especifica` e as instruções equivalentes no exemplo da Seção11.2/página 117 em [5]. É alocado um pino físico a LPTMR0? Caso sim, qual?

2.e **Processamento de interrupções em LPTMRx/Disparos para outros Módulos**: Para gerar disparos periódicos ao módulo ADC0, é necessário habilitar o mecanismo de interrupção de LPTMRx? Justifique com base na comparação das instruções de configuração das interrupções entre o exemplo da Seção11.2/página 117 em [5] e o projeto `rot9_example1`.

3 **Praticar as práticas**: Desenvolva o projeto `controle_cooler` em que o sinal analógico do potenciômetro e o sinal analógico do sensor de temperatura são amostrados periodicamente em conformidade com o diagrama de máquina de estados apresentado na Figura 13.

No estado `AMOST_TEMP` o canal 26 de ADC0 deve estar selecionado para que, ao ocorrer o disparo do TPM1, a amostragem e a conversão do sinal analógico neste canal sejam iniciadas automaticamente. Após a conclusão da conversão, o evento de interrupção COCO é acionado, desviando o fluxo de controle para a rotina de serviço `ADC0_IRQHandler`. Dentro dessa rotina,

além de salvar o resultado da conversão, o estado é alterado para `AMOST_VOLT` e o modo de disparo é configurado para *software*. Dessa forma, ao escrever o canal 9 no campo `ADC0_SC1A_ADCH`, a amostragem e conversão do sinal analógico no canal 9 são disparadas. Novamente, ao concluir a conversão, o evento de interrupção `COCO` é acionado, desviando o fluxo de controle para `ADC0_IRQHandler`. Dentro da rotina de serviço, além de aplicar a filtragem exponencial no valor amostrado e salvar o resultado da conversão filtrado, o estado é alterado para `ATUALIZACAO`, o canal é definido como 26, o modo de disparo é configurado para *hardware* e o contador do TPM1 é resetado. Assim, decorrido o período configurado no TPM1, a amostragem e conversão no canal 26 são disparadas. Durante o intervalo do período, é realizada a atualização do visor do LCD, do estado do *cooler* e do estado do LED R. Ao concluir a atualização, o estado é novamente definido como `AMOSTRA_TEMP`, reiniciando o ciclo.

Recomenda-se os seguintes passos de desenvolvimento:

3.a **Especificação funcional:** Complete a descrição do comportamento desejado do projeto `controle_cooler`, detalhando as condições que levam a cada estado, as atividades ou comportamentos realizados enquanto o sistema está nesse estado e as transições que podem ocorrer a partir desse estado.

3.b **Especificação implementacional:** Identifique os eventos de interrupção que podem ocorrer durante a operação de `controle_cooler`. Detalhe em diagramas de atividades, ou em uma representação equivalente, as ações dentro de cada estado, incluindo a habilitação e desabilitação desses eventos. Descreva ainda a sequência de interações entre os estados durante o tratamento de cada evento com o uso de diagramas de sequência, ou uma representação equivalente. Identifique blocos de instruções comuns nos estados e os parametrize em **funções** para ajudar a simplificar o código e facilitar a manutenção.

3.c **Implementação:** Escreva o código com base na especificação implementacional. Procure reusar as funções implementadas. Documente a interface de todas as **novas funções** implementadas seguindo sintaxe Doxygen [18].

3.d **Testes:** Realize testes para garantir que o sistema atenda aos requisitos especificados. Isso pode incluir testes de unidade e testes de integração. Registre os testes conduzidos e os resultados.

3.e **Depuração:** Identifique e corrija quaisquer problemas encontrados durante os testes. Habilite *Print Size* para uma simples análise do tamanho de memória ocupado.

RELATÓRIO

O relatório deve ser devidamente identificado, contendo a identificação do instituto e da disciplina, o experimento realizado, o nome e RA do aluno. O prazo para execução deste experimento é duas semanas. O relatório é dividido em duas partes. Na primeira semana, é necessário responder as questões do item 2 e realizar a especificação funcional e implementacional do projeto `controle_cooler`. **Isso inclui computar os valores a serem configurados nos registradores do KL25Z, descrever detalhadamente o comportamento desejado do projeto calculadora, identificar os eventos de interrupção que podem ocorrer durante sua operação, detalhar as ações dentro de cada estado e descrever as sequências de interações entre os estados.** Suba dois arquivos no sistema *Moodle*: um contendo as respostas do item 2 e outro, as especificações do projeto `controle_cooler`. Na segunda semana, deve-se fazer **uma descrição sucinta dos testes conduzidos, incluindo os resultados obtidos e quaisquer correções realizadas ao longo do**

desenvolvimento do projeto. Além disso, é necessário exportar **o projeto `controle_cooler` devidamente documentado** em um arquivo comprimido no IDE CodeWarrior e subir ambos os arquivos no sistema *Moodle*. **Não se esqueça de limpar o projeto (*Clean* …) e apagar as pastas html e latex geradas pelo Doxygen antes**.

REFERÊNCIAS

[1] Freescale. *KL25 Sub-Family Reference Manual*.
https://www.dca.fee.unicamp.br/cursos/EA871/references/ARM/KL25P80M48SF0RM.pdf
[2] Elliot Smith. Understanding the Successive Approximation Register ADC.
https://www.allaboutcircuits.com/technical-articles/understanding-analog-to-digital-converters-the-successive-approximation-reg/
[3] Temperature Sensor for the HCS08 Microcontroller Family
https://www.nxp.com/docs/en/application-note/AN3031.pdf
[4] Instituto Newton C. Braga. Como funcionam os conversores A/D?
http://www.newtoncbraga.com.br/index.php/como-funciona/1508-conversores-ad
(parte 1) e http://www.newtoncbraga.com.br/index.php/como-funciona/1509-conversores-ad-2
(parte 2)
[5] Freescale. *Kinetis L Peripheral Module Quick Reference (Rev. 0.09/2012)*.
https://www.dca.fee.unicamp.br/cursos/EA871/references/ARM/KLQRUG.pdf
[6] Nova versão do esquemático do shield FEEC
https://www.dca.fee.unicamp.br/cursos/EA871/references/complementos_ea871/Esquematico_EA871-Rev3.pdf
[7] Folha de dados técnicos - Kinetis KL25 Sub-Family 48 MHz Cortex-M0+ Based Microcontroller with USB.
https://www.dca.fee.unicamp.br/cursos/EA871/references/ARM/KL25P80M48SF0.pdf
[8] Azo Sensors. How Force Sensing Resistors Measure Force
https://www.azosensors.com/article.aspx?ArticleID=1718
[9] Resistance vs. Illumination curve
https://www.researchgate.net/figure/Resistance-vs-illumination-curve-LDRs-are-light-dependent-devices-whose-resistance_fig3_318029856
[10] Temperature sensor ICs simplify Design
https://www.maximintegrated.com/en/design/technical-documents/app-notes/6/694.html
[11] rot9_aula.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2024/codes/rot9_aula.zip
[12] rot7_aula.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2024/codes/rot7_aula.zip
[13] Doxygen
https://www.doxygen.nl/manual/docblocks.html
[14] Diagrams.net
https://www.diagrams.net/
[15] Analog Devices. The ABCs of Analog to Digital Converters: How ADC Errors Affect System Performance
https://www.analog.com/en/technical-articles/the-abcs-of-analog-to-digital-converters-how-adc-errors-affect-system-performance.html
[16] Direct Type ADCs
https://www.tutorialspoint.com/linear_integrated_circuits_applications/linear_integrated_circuits_applications_direct_type_adcs.htm

[17] rot9_example1.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2023/codes/rot9_example1.zip
[18] Successive-approximation ADC
https://en.wikipedia.org/wiki/Successive-approximation_ADC
[19] Wu, S.T. Ambiente de Desenvolvimento de Software
https://www.dca.fee.unicamp.br/cursos/EA871/references/apostila_C/AmbienteDesenvolvimentoSoftware_V1.pdf
[20] rot9_cooler.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2023/codes/rot9_cooler.zip

Revisado em Fevereiro de 2023
Revisado em Março e Outubro de 2022
Revisado em Maio e Julho de 2021
Revisado em Novembro de 2020
Revisado em Agosto de 2017
Elaborado com base no roteiro do Experimento 13 no Segundo Semestre de 2015.