EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 8 – TPM, DMA e DMAMUX

Profa. Wu Shin-Ting

OBJETIVO: Apresentação das funcionalidades PWM do módulo TPMx (*Timer/PWM*).

ASSUNTOS: Geração de sinais PWM (*Pulse Width Modulation*), programação do MKL25Z128 para processamento destes sinais via módulos TPMx.

O que você deve ser capaz ao final deste experimento?

Entender o princípio de funcionamento de TPMx.

Saber programar a base de tempo do contador de um módulo TPMx.

Saber configurar os canais de TPM para as funções *Input Capture* (IC), *Output Compare* (OC) e *Pulse Width Modulation* (PWM).

Programar KL25Z para capturar eventos de entrada com "marcadores de tempo".

Programar KL25Z para gerar sinais digitais específicos de saída condicionados a "instantes de tempo"

Programar KL25Z para gerar sinais PWM de ciclos de trabalho de interesse.

Entender o princípio de funcionamento dos circuitos *Direct Memory Access* (DMA) e Multiplexador de DMA (DMAMUX).

Saber configurar os módulos DMA, DMAMUX, a fonte e o destino para transferências via DMA.

Ter uma ideia do projeto de um filtro passa-baixo para converter sinais modulados por largura de pulso em sinais modulados por níveis de tensão.

Saber aplicar os contadores dos temporizadores para gerar números aleatórios.

Saber usar o tipo `struct` para organizar dados estruturados e parametrizá-los.

Saber diferenciar a aritmética de inteiros e de pontos flutuantes em C.

Ter uma noção sobre os problemas envolvidos no processamento de pontos flutuantes.

Ter uma noção da conversão de pontos flutuantes para inteiros em C.

Saber aplicar máquina de estados e mecanismo de interrupções na proteção de ações indevidas dos usuários.

Saber implementar os exemplos de aplicação dos módulos do microcontrolador apresentados em [14].

INTRODUÇÃO

Além do *timer* integrado ao núcleo, *SysTick* (Seção B.3.3/página 277 em [1]), e dos módulos PIT (*Periodic Interrupt Timer,* Cap. 32/página 573 em [2]), LPTMR (*Low-Power Timer*, Cap. 33, página 587, em [2]) e RTC (*Real Time Clock*, Cap. 34/página 597 em [2]), o microcontrolador KL25Z dispõe ainda de três módulos `TPM` (**Timer/PWM**, Cap. 31/página 547 em [2]), 1 com 6 canais e 2 com 2 canais, para temporização. Diferentemente de outros temporizadores, nos módulos TPM são integrados os circuitos dedicados de captura na entrada (*input capture*), saída por comparação (*compare output*) e modulação de largura de pulso (*pulse width modulation*). Esses circuitos têm capacidade de gerar sinais mais complexos, além de uma sequência de eventos periódicos de um temporizador típico, e suportam

pinos de comunicação com o mundo exterior do microcontrolador, oferecendo um melhor suporte à implementação de aplicações mais complexas.

Neste experimento vamos apresentar as funções configuráveis nos módulos TPMx através de três exemplos de projeto e aplicar o aprendizado no desenvolvimento de um projeto de medidor de tempos de reação, `tempo_reacao`. Entende-se como o **tempo de reação** o intervalo de tempo entre a geração de um estímulo e uma ação motora [16]. Os estímulos a serem aplicados neste projeto são audíveis (som de um *buzzer*). Veremos também uma possível forma de coletar os dados gerados nos periféricos de forma mais eficiente, com mínimas intervenções do processador, fazendo uso do módulo ***Direct Memory Access*** (DMA).

Módulo TPM

Em cada módulo TPMx há um contador LPTPM de 16 *bits*, mapeado no registrador `TPMx_CNT`, que conta ciclicamente de forma crescente (*up,* `TPMx_SC_CPWMS`=0) ou crecente-decrescente (*up-down,* `TPMx_SC_CPWMS`=1) (Figura 31-1/página 549 em [2]). Qualquer acesso de escrita em `TPMx_CNT` reseta o contador em zero. LPTPMs são pulsados por um sinal de relógio assíncrono `TPM clock` (Seção 5.7.5/página 124 em [2]), independente do sinal do barramento (*bus clock*) que coordena o restante dos circuitos do módulo. Assim, o contador pode manter a sua contagem mesmo em modos de baixo consumo energético.

A fonte de `TPM clock` é selecionável pelos campos `SIM_SOPT2_TPMSRC` e `SIM_SOPT2_PLLFLLSEL` a partir dos diferentes tipos de sinais de relógio gerados pelo módulo MCG, `MCGFLLCLK`, `MCGPLLCLK/2`, `OSCERCLK` e `MCGIRCLK` (Figura 24-1/página 370 em [2]). Os sinais de relógio de LPTPM podem ser também externos, através dos pinos devidamente multiplexados em `TPM_CLKIN0` ou `TPM_CLKIN1` (Seção 10.3.1/página 162 em [2]). Além da seleção da fonte, é necessário habilitá-la para os módulos TPM0, TPM1 e TPM2 pelos respectivos *bits* `SIM_SCGC6_TPM0`, `SIM_SCGC6_TPM1` e `SIM_SCGC6_TPM2` (Seção 12.2.10/página 207 em [2]). E, para ativar efetivamente a contagem em LPTPM, é necessário setar o *bit* `TPMx_SC_CMOD` em '1'. O estouro na contagem, ou seja quando `TPMx_CNT` atinge o valor setado em `TPMx_MOD`, seta a *flag* (de estado) `TPMx_SC_TOF`.

Um módulo TPMx contém múltiplos canais programáveis para uma das três funções: *input capture*, *output compare* ou *pulse width modulation*. Esses canais compartilham a mesma base de tempo definida por LPTPM, mas possuem seus próprios registradores de configuração `TPMx_CnSC`, registradores de dados `TPMx_CnV`, comparadores, pinos de comunicação com o mundo externo, e circuitos de controle dos pinos e de interrupções. Isso permite que um canal seja programado com uma função independente, porém sincronizada com as funções executadas em outros canais do mesmo módulo, e a sua *flag* `TPMx_CnSC_CHF` seja setada em função da ocorrência de um evento relacionado com a função programada nele.

É possível ainda configurar através do registrador de configuração `TPMx_CONF` uma base de tempo global/comum para os três módulos TPMx e o comportamento dos módulos nos modos de operação *Debug* e Espera (Seção 31.3.7/página 559 em [2] e Seção 12.3.8/página 126 em [14]).

Configuração de um Período em TPM

De acordo com as Seções 31.4.2/página 562 e 31.4.3/página 562 em [2], o período (contagem máxima) de LPTPM depende, além da frequência da fonte f_{clock} (`TPM clock`) selecionada, dos valores setados

em TPMx_MOD (valor de referência para contagem máxima, Seção 31.3.3/página 554 em [2]) e em TPMx_SC_PS (divisor *prescaler*, Seção 31.3.1/página 552 em [2])

$$Periodo = TPMx\_MOD \; x \frac{2^{TPMx\_SC\_PS}}{f_{clock}} x \left(1 + TPMx\_SC\_CPWMS\right) \qquad (1)$$

Funções Programáveis nos Canais de TPMx

A principal característica do módulo TPM é que cada um dos seus canais pode ser configurado para operar num dos três modos através dos *bits* TPMx_CnSC_MSnB e TPMx_CnSC_MSnA, do registrador de controle do canal TPMx_CnSC: *Input Capture* (Seção 31.4.4/página 564 em [2]), *Output Compare* (Seção 31.4.5/página 565 em [2]) e *PWM* (Seções 31.4.6 e 31.4.7/página 566 em [2]). Os níveis ou as bordas de interesse em cada um dos três modos de operação são configurados pelos *bits* TPMx_CnSC_ELSnB e TPMx_CnSC_ELSnA, do mesmo registrador. Usando como base de tempo o contador TPMx_CNT do módulo TPMx, um canal configurado com a função:

Input Capture (MSnB:MSnA==00) associa precisamente ao evento de interrupção detectado (borda de subida – ELSnB:ELSnA==01, borda de descida – ELSnB:ELSnA==10, ou ambas as bordas – ELSnB:ELSnA==11) o valor de contagem registrado em TPMx_CNT. O circuito captura o valor de TPMx_CNT no registrador TPMx_CnV no momento em que um evento pré-especificado ocorre. Em paralelo à captura, é setada a *flag* TPMx_CnSC_CHF em '1' (Seção 31.4.4/página 564, em [2]). Podemos dizer que é um circuito que rotula os eventos de interrupção com os "dados" que nos permitem inferir instantes de tempo de forma acurada. Para que o sinal de entrada seja corretamente amostrado, a sua frequência (mais alta) deve ser 2 vezes menor que a frequência de contagem do relógio do módulo TPMx (teorema de amostragem Nyquist-Shannon [17]).

Output Compare (MsnB:MsnA==01 ou 11) gera um sinal de saída pré-configurado (alterna – ELSnB:ELSnA==01, reseta – ELSnB:ELSnA==10, seta – ELSnB:ELSnA==11, pulso positivo – ELSnB:ELSnA==X1 ou pulso negativo – ELSnB:ELSnA==10) quando o valor de contagem no contador TPMx_CNT se iguala ao valor em TPMx_CnV. A periodicidade de atualização dessa saída é alinhada com o valor TPMx_CnV (Figuras 31-82 a 31-84/página 566 em [2]). No momento em que TPMx_CNT==TPMx_CnV, é setada a *flag* TPMx_CnSC_CHF em '1' (Seção 31.4.4/página 564, em [2]). Podemos dizer que é um circuito que faz contagens cíclicas a partir de um valor de contagem e permite pré-programar, de forma acurada, uma saída condicionada a um valor de contagem que pode corresponder a um instante específico de tempo.

Pulse Width Modulation (MsnB:MsnA==10) gera um sinal de saída com a largura de pulso e polaridade controláveis pelos *bits* de configuração TPMx_SC_CPWMS e pelo registrador de dados TPMx_CnV. A periodicidade dessa saída é alinhada com TPMx_CNT==0. Quando TPMx_SC_CPWMS==0, o alinhamento é com uma borda do pulso (*edge-aligned* PWM ou EPWM, Figura 31-87/página 568 em [2]). O nível lógico de saída assume '1' quando TPMx_CNT==0 e ELSnB:ELSnA==10, e '0' quando ELSnB:ELSnA==X1. Esse nível é alternado quando TPMx_CNT==TPMx_CnV. Quando TPMx_SC_CPWMS==1, o alinhamento é feito com o centro do pulso (*center-aligned* PWM ou CPWM, Figura 31-88/página 569 em [2]). O nível lógico de saída é '1' quando TPMx_CNT== TPMx_CnV na contagem decrescente se ELSnB:ELSnA==10, e '0' quando ELSnB:ELSnA==X1. O sinal é alternado quando TPMx_CNT== TPMx_CnV na contagem crescente. Sempre que TPMx_CNT==TPMx_CnV, a *flag* TPMx_CnSC_CHF é setada em '1' (Seção 31.4.4/página 564, em [2]). Podemos dizer que é um circuito que gera uma forma de onda com

período fixo, definido por TPMx_MOD, e larguras configuráveis pelo registrador TPMx_CnV. A razão cíclica das larguras dos pulsos em relação ao período do sinal é conhecida como **ciclo de trabalho** (*duty cycle*, Figura 1). Note que o incremento para m unidades sempre ocorre na transição de m-1 para m**. Para que tenhamos um ciclo de trabalho 100% no modo EPWM, o valor setado em TPMx_CnV deve ser pelo menos uma unidade maior do que o valor setado em TPMx_MOD** (Seção 31.4.6/página 567 em [2]).

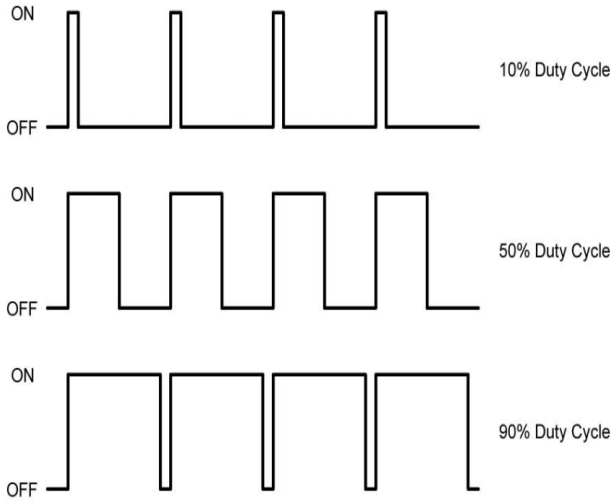

Figura 1: Modulação de largura de pulso [3]

Alocação de Pinos para TPMx

Diferentemente dos temporizadores básicos, os módulos TPMx têm a capacidade de capturar sinais externos e gerar sinais com características específicas através de seus canais. Para interagir com o mundo externo por meio desses sinais, é necessário alocar um pino físico para cada canal TPMx_CHn. Essa alocação deve ser com base na consulta da tabela de multiplexação dos pinos, encontrada na Seção 10.3.1/página 162 em [2], onde são listados os pinos que podem ser multiplexados para a função TPMx_CHn. Por exemplo, os pinos PTA4 e PTB18 podem ser designados para servir, respectivamente, ao canal TPM0_CH1 e ao canal TPM2_CH0, se estiverem devidamente multiplexados para as funções TPM0_CH1 e TPM2_CH0, ou seja, se PORTA_PCR4_MUX==0x03 e PORTB_PCR18_MUX==0x03. No entanto, TPMx podem ser usados como simples temporizadores básicos, apenas para a contagem de tempos por períodos, sem a necessidade de alocar um pino para o canal, assim como os temporizadores básicos SysTick (Cap. B3.3/página 275 em [1]) e PIT (Cap. 32/página 573 em [2]).

Disparos Externos para TPMx_CNT

O TPMx oferece a capacidade de reiniciar a contagem do contador LPTPM em 0 de forma assíncrona por meio de eventos externos, conhecidos como **disparadores** (***triggers***). As fontes válidas desses disparadores estão identificadas na Tabela 3-38/página 86 em [2]. Uma vez que um desses identificadores é configurado no campo TPMx_CONF_TRGSEL, ele passa a controlar os instantes em que LPTPM altera o modo de operação. Por exemplo, se for setado 0b1000 nesse campo, o conteúdo de TPMx_CNT será modificado quando a *flag* de TPM0_SC_TOF fique em '1'. A forma como essa modificação ocorre depende dos valores configurados nos *bits* de configuração TPMx_CONF_CROT (TPMx_CNT é resetado em '0'), TPMx_CONF_CSOO (TPM_CNT pára a contagem na ocorrência de um estouro e só retoma a contagem quando rehabilitado ou quando TPMx_CONF_CSOT==1 e recebe um disparo) e TPMx_CONF_CSOT (a contagem só é iniciada quando recebe um disparo).

O projeto `rot8_example1` ilustra a aplicação desses disparadores para delimitar, dentro de um período de tempo, as ocorrências das bordas de subida RE e de descida FE de pulsos PWM com larguras variáveis de forma cíclica. Isso simplifica o cômputo dessas larguras, reduzindo-o à diferença entre dois valores de contagem capturados pela função *Input Capture* configurada nos canais `TPM1_CH0` e `TPM1_CH1`. Ao multiplicarmos essa diferença pelo período dos pulsos do contador `TPM1_CNT`, obtemos o tempo decorrido entre as duas bordas.

A Figura 2 ilustra os intervalos entre os instantes CSOT e CSOO, nas quais a contagem do TPM1 é efetivamente ativada. Vale destacar aqui que a configuração dos disparadores periódicos é realizada apenas uma vez, durante a inicialização do módulo TPM1. Cada vez que um disparador é ativado, a contagem do `TPM1_CNT` é reiniciada automaticamente pelo *hardware*. O controle para a interrupção dos incrementos quando `TPM1_CNT` atinge a contagem máxima também realizado pelo *hardware*. Porisso, não são necessárias outras instruções além das configurações iniciais para processar a forma de onda TPM1 (na terceira linha da Figura 2).

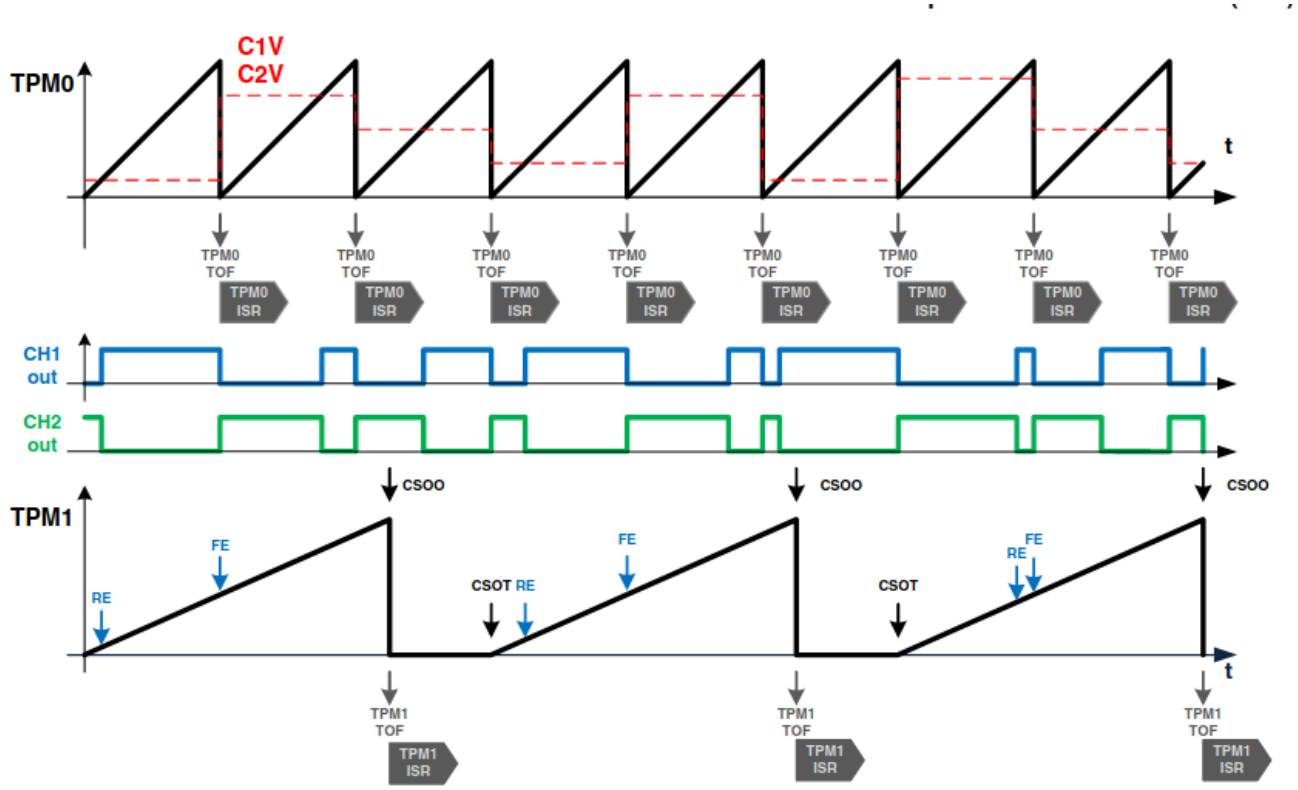

Figura 2: Disparos para controlar a contagem de `TPMx_CNT` no tempo (cópia da Fig. 12-4 em [14]).

Processamento de Interrupções em TPMx

Os contadores nos módulos TPMx são de corrida livre (*free running*), o que significa que eles percorrem todos os estados em um ciclo completo de contagem, indo de 0 até o valor máximo de contagem, definido por `TPMx_MOD`. Como acontece com todos os temporizadores, quando o valor do contador `TPMx_CNT` atinge o valor de `TPMx_MOD`, o TPMx gera um evento de *Overflow*. Além do estado de estouro do contador LPTPM (*overflow*) mostrado no *bit* `TPMx_SC_TOF`, cada canal tem um *bit* de estado `TPMx_CnSC_CHF` associado, que indica a ocorrência de um evento de interrupção configurado.

Quando os *bits* de habilitação de interrupção, `TPMx_SC_TOIE` e `TPMx_CnSC_CHIE`, correspondentes a cada um desses *bits* de estado estão em '1', uma requisição de interrupção IRQ é gerada, com o número de vetor associado ao módulo, assim que o *bit* de estado fiquem se torna '1'.

Esse número de vetor é igual a 33 para TPM0 (IRQ=17), a 34 para TPM1 (IRQ=18) e a 35 para TPM2 (IRQ=19) (Tabela 3-7/página 52 em [2]). Se as IRQ17, IRQ18 e IRQ19 estiverem habilitadas no lado do controlador NVIC (Seção B3.4/página 281 em [1]), o fluxo de controle é, automaticamente desviado para as rotinas de serviço pré-declaradas no arquivo `Project_Settings/Startup_Code/kinetis_sysinit.c`, gerado pelo IDE CodeWarrior. Os nomes declarados para as rotinas que tratam os eventos de TPM0, TPM1 e TPM2 são, respectivamente, `FTM0_IRQHandler`, `FTM1_IRQHandler` e `FTM2_IRQHandler`.

Note que há apenas uma linha de requisição associada a cada TPMx. As solicitações provenientes de diversas fontes são combinadas por uma lógica OU dentro do módulo. Para identificar a origem de uma interrupção nas rotinas de serviço, deve-se avaliar os *bits* de estado individualmente. Para otimizar o acesso a esses *bits*, que estão espalhados por diferentes registradores de controle/estado, há um espelho desses *bits* num único registrador de estado `TPMx_STATUS`. Para evitar probleas de "reentrâncias", todos os *bits* de estado devem ser resetados em '0' por meio de um acesso de escrita de '1' (*write-1-to-clear*) após o atendimento .

Uma vez que muitas funções suportadas por TPMx são executadas inteiramente por *hardware* e ele pode gerar uma variedade de eventos de interrupção funcionalmente distintos, a programação das tarefas a serem executadas pode ser simplificada com pequenas intervenções por *software* dentro das rotinas de serviço, como exemplificado no projeto `rot8_example1`. A tarefa desse projeto é fazer medições dos intervalos de tempo entre pares de eventos de interrupção gerados pelas capturas das bordas RE e FE (Figura 2). A forma mais precisa de ler as contagens capturadas é através da rotina de serviço `FTM1_IRQHandler`. Dado que o processamento dos dois valores capturados é bastante simples, a diferença é calculada na mesma rotina. O laço principal vazio da função `main` simplesmente mantém o processador aguardando as solicitações de interrupções.

Módulo DMA

O circuito de **Acesso Direto à Memória** (***Direct Memory Access***, DMA) é um controlador que permite a transferência de dados entre dispositivos sem a intervenção direta do processador,o que melhora a eficiência do sistema e permite que o processador fique livre para executar outras tarefas [13]. Em KL25Z todos os registradores dos módulos-periférico são mapeados no espaço de endereços de 32 *bits* do processador. O módulo DMA contém 4 canais independentes para transferência de 8-, 16- e 32-*bits*, cuja prioridade de atendimento segue a ordem canal 0 > canal 1 > canal 2 > canal 3. Cada canal `n` tem

- Um registrador de dados `DMA_SARn` (Seção 23.3.1/página 353 em [2]) para armazenar o endereço do rementente/fonte (*source*).
- Um registrador `DMA_DARn` (Seção 23.3.2/página 354 em [2]) para o endereço do destinatário.
- Um registrador de estado `DMA_DSRn` (Seção 23.3.3/página 355 em [2]) que indica o estado de uma transferência, incluindo erros de configuração (`DMA_DSR_BCRn_CE`), erros no barramento (`DMA_DSR_BCRn_BES`, `DMA_DSR_BCRn_BED`), estado de pendência (`DMA_DSR_BCRn_REQ`), estado ocupado (`DMA_DSR_BCRn_BSY`) e estado concluído (`DMA_DSR_BCRn_DONE`).
- Um contador de 24 *bits* `DMA_DSR_BCRn_BCR` (Seção 23.3.3/página 355 em [2]), que contém a quantidade de *bytes* a serem transferidos e é decrementado da quantidade de *bytes* transferidos após cada transferência bem sucedida.
- Um registrador de controle/configuração `DMA_DCRn` (Seção 23.3.4/página 357 em [2]), que habilita a interrupção ao completar uma transferência (`DMA_DCRn_EINT`), solicitação de

periférico para uso do canal (DMA_DCRn_ERQ), mecanismo de roubos de ciclos (DMA_DCRn_CS), auto-alinhamento com base nos endereços e no tamanho de dados (DMA_DCRn_AA), e requisições assíncronas (DMA_DCRn_EADREQ). Através deste registrador, configuram-se os tamanhos dos *buffers* circulares (DMA_DCRn_SMOD e DMA_DCRn_DMOD no remetente e no destinatário, respectivamente), o tamanho de dados por transferência (DMA_DCRn_SSIZE e DMA_DCRn_DSIZE no remetente e no destinatário, respectivamente) e modo de atualização dos endereços para a próxima transferência (DMA_DCRn_SINC e DMA_DCRn_DINC, respectivamente).

Antes de iniciar uma transação por DMA, é necessário carregar os endereços iniciais dos blocos de dados do remetente e do destinatário, respectivamente em DMA_SARn e DMA_DARn, além de definir os tamanhos dos dados, o tamanho dos *buffers* circulares a serem alocados e o modo de atualização dos endereços para a próxima transferência no registrador de configuração DMA_DCRn. Deve-se também especificar a quantidade total de *bytes* a serem transferidos na transação em DMA_DSR_BCRn_BCR (Seção 23.4.2.2/página 362 em [2]). Uma transação pode ser iniciada por *software*, configurando o *bit* DMA_DCRn_START, ou por requisição de um módulo periférico se o *bit* DMA_DCRn_ERQ esteja setado em '1'. É importante habilitar a requisição de transferência por DMA individualmente em cada módulo que solicitará a requisição. Por exemplo, para os módulos PORTx/GPIOx, a habilitação é feita através dos *bits* PORTx_PCRn_IRQC; para o módulo UART0 é pelos *bits* UART0_C5_TDMAE e UART0_C5_RDMAE; e para um módulo TPMx, pelo *bit* de configuração TPMx_CnSC_DMA.

Processamento de Interrupções em DMA

Quando os *bits* DMA_DSR_BCRn_DONE e DMA_DCRn_EINT estão ambos definidos como '1', um evento de interrupção IRQ é gerado, com o número de vetor associado ao canal correspondente. Esse número de vetor é 16 para o canal 0 (IRQ=0), 17 para o canal 1 (IRQ=1) e 18 para o canal 2 (IRQ=2) e 19 para o canal 3 (IRQ=3) (Tabela 3-7/página 52 em [2]). Se as IRQ0, IRQ1, IRQ2 e/ou IRQ3 estiverem habilitadas no controlador NVIC (Seção B3.4/página 281 em [1]), o fluxo de controle é automaticamente desviado para as rotinas de serviço pré-declaradas no arquivo Project_Settings/Startup_Code/kinetis_sysinit.c gerado pelo IDE CodeWarrior. Os nomes declarados para as rotinas que tratam os eventos dos canais 0, 1, 2 e 3 são, respectivamente, DMA0_IRQHandler, DMA1_IRQHandler, DMA2_IRQHandler e DMA3_IRQHandler. Note que há apenas uma linha de requisição associada a cada canal. A conclusão de uma transação de dados ou a ocorrência de algum erro faz o *bit* DMA_DSR_BCRn_DONE ficar em '1'. Para identificar a origem de uma interrupção nas rotinas de serviço, deve-se avaliar os *bits* de estado individualmente. O *bit* DMA_DSR_BCRn_DONE deve ser resetado por meio de um acesso de escrita (*write-1-to-clear*) para remover o evento de interrupção dentro da rotina de serviço. Esse acesso de escrita reseta todos os *bits* de estado do DMA, garantindo que o controlador DMA esteja em um estado limpo e pronto para novas operações.

Módulo DMAMUX

O circuito DMAMUX é um multiplexador que permite a seleção de várias fontes/*slots* de dados para serem transferidas pelo controlador DMA. Juntos, eles permitem que múltiplos dispositivos compartilhem o uso do DMA para transferir seus dados. É importante frisar que a função de DMAMUX é apenas a de multiplexagem de uma rota; todos os controles de transferência pela rota selecionada são de responsabilidade do módulo DMA.

Em KL25Z, o módulo DMAMUX é capaz de rotear 63 *slots* habilitáveis nos respectivos módulos, além de 6 *slots* sempre-habilitáveis para os 4 canais disponíveis. As transferências por DMA entre os módulos GPIO e as unidades de memória são sempre habilitáveis. No modo de roteamento normal, os dados (remetentes) endereçados são roteados diretamente para o canal pré-determinado em cada requisição pré-especificada. Os códigos de todas as possíveis requisições para uma transferência DMA estão listados na Tabela 3-20/página 64 em [2].

Para melhorar a **previsibilidade**, os dois primeiros canais, 0 e 1, podem ter suas requisições de transferências sobrepostas pelos disparos periódicos automáticos gerados pelo temporizador PIT (Seção 22.4.1/página 341 em [2]). A seção 22.5.2/página 344 em [2] apresenta os exemplos de configuração e habilitação dos *slots* habilitáveis, mais especificamente do *slot* #5 (Transmissor de UART1), para o canal 2 do DMA com e sem disparos automáticos de requisições (Seção 22.4.1/página 341 em [2]).

No projeto `rot8_example2,` é ilustrada a transferência dos dados de SRAM (sempre-habilitável) para o registrador de dados de 16 *bits* `TPM1_C1V` do canal `TPM1_CH1` que é configurado com a função EPWM, usando o canal 0 do módulo DMA. Para habilitar esse canal, é necessário setar em '1' o *bit* `TPM1_C1SC_DMA`, e especificar os instantes em que os dados devem ser roteados para o canl 0 do DM, donfigurando o código da fonte de requisição em `DMAMUX0_CHCFG0_SOURCE.` No caso apresentado, o código é #55, que corresponde ao evento de *Overflow* do TPM0. Os valores pré-carregados na SRAM determinam diferentes larguras de pulso no sinal de saída de `TPM1_CH1`. Esses sinais, ao passarem por um filtro passivo RC, resultam num **sinal analógico** com variações nos níveis de tensão.

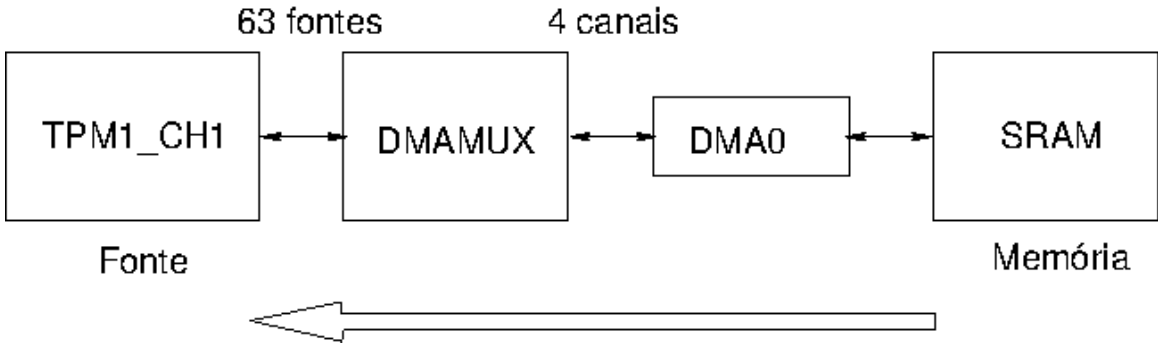

Figura 3: Projeto `rot8_example2`: transferência, via o canal 0 de DMA, dos dados de SRAM (sempre-habilitável) para TPM1_CH1 (habilitável por `TPM1_C1SC_DMA`) através do sinal de requisição 55 (TPM1 *Overflow*).

Conversão de Sinal Modulado por Largura de Pulso em Sinal Modulado por Nível de Tensão

A técnica de PWM, juntamente com filtrso passivos, é amplamente empregada em sistemas embarcados para converter sinais digitais em formas de onda analógicas, variando o ciclo de trabalho do sinal PWM. No entanto, é importante compreender que esse processo de conversão é realizado de forma indireta, uma vez que **os componentes passivos** não geram diretamente formas de onda analógicas, mas sim modificam o sinal digital para produzir uma saída que se assemelha a uma forma de onda analógica quando filtrada por um circuito RC, por exemplo.

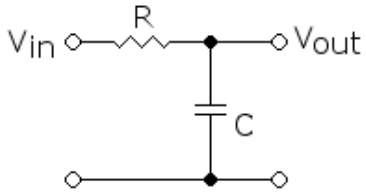

Figura 4: Filtro passivo passa-baixas.

O valor da resistência R, em Ohms, e da capacitância C, em Farads, é determinado em função da constante de tempo do circuito. A **contante de tempo** RC é o tempo, em segundos, necessário para carregar um capacitor em série com um resistor até atingir 63.2% do valor da tensão (de alimentação contínua) aplicada sobre ele. Para um sinal PWM de período T em que a menor largura seja T/N, podemos usar um filtro RC tal que

$$RC \geq N \times T.$$

Por exemplo se T = 0.001s (frequência de amostragem em 1kHz) e as larguras dos pulsos são múltiplos de T/64, então podemos escolher para R=100Ω um capacitor de capacitância

$$C \geq \frac{(64 \times 0.001)}{100} = 640\,uF$$

para atenuar as ondulações entre os pulsos gerados pelo circuito PWM. Essa escolha dos valores de R e C permite que o filtro RC atenue as variações de tensão entre os **pulsos do sinal PWM**, resultando em um **sinal mais suave e contínuo**.

Cômputo de Intervalo de Tempo entre dois Eventos de Interrupção de *Input Capture* por *Overflows*

No projeto `rot8_example1` (Figura 2), se não forçarmos a reinicialização do contador `TPMx_CNT` e definirmos um período de TPMx que cubra o intervalo de tempo entre as ocorrências RE (borda de subida em CH1) e FE (borda de descida em CH1), as duas ocorrências podem acontecer em períodos diferentes. É necessário, então, considerar as contagens dos períodos completos que transcorreram entre as duas ocorrências por meio de detecção dos eventos de estouros (o*verflows)*. Com a contagem de `N` estouros, pode-se computar o intervalo de tempo `t` entre as duas contagens C_{T1} e C_{T2} nos instantes T1 e T2 usando as seguintes equações

T1 > T2:

$$t = \left(\frac{(TPMx\_MOD + C_{T2} - C_{T1})}{TPMx\_MOD} + (N-1) \right) \times Período \quad (2)$$

T1 ≤ T2:

$$t = \left(\frac{(C_{T2} - C_{T1})}{TPMx\_MOD} + N \right) \times Período \quad , \qquad (3)$$

onde `TPMx_MOD` é o valor máximo de um ciclo completo de contagem e o *Período*, o intervalo de tempo deste ciclo completo (Figura 12-1 em [14]). O *Período*, por sua vez, pode ser calculado utilizando a Eq. (1) com f_{clock} =20,97152MHz em nossos projetos, em que usamos o sinal de relógio MCGFLLCLK.

A captura por *hardware* dos eventos de *overflows* e dos eventos de entrada, junto com os seus "marcadores de tempo" detectados de "forma imediata", aumenta a precisão do cômputo do intervalo

de tempo. Se resetarmos sincronamente o valor do contador LPTPM em 0 no instante T1, C_{T1} na Eq. (3) assumirá o valor 0. Isso simplificará ainda mais o cômputo.

Segue-se um pseudo-código do cálculo de `t` na rotina de serviço que trata as capturas de um canal TPMx_CHn

```
Se (flag de interrupção de TPMx_CHn  estiver em 1) então
  Se for a primeira captura então
     C_T1 ← TPMx_CnV;
     N ← 0;
     Habilitar a interrupção de Overflow de TPMx;
  Se for a segunda captura então
     C_T2 ← TPMx_CnV;
     Computar Eq. (2) ou (3);
Se (flag de overflow de TPMx estiver em 1) então
  N← N+1;
```

<u>Cômputo de Intervalo de Tempo entre dois Eventos de Interrupção de *Input Capture* por *Output Compare*</u>

A função *Output Compare* permite gerar, por *hardware*, um evento de interrupção quando o valor do contador `TPMx_CNT` se iguala ao valor configurado no registrador `TPMx_CnV`. Isso aumenta a precisão e a velocidade da resposta do sistema em relação a uma referência pré-especificada. Em conjunto com a função *Input Capture*, ela simplifica o cômputo do intervalo de tempo entre os dois instantes capturados, T1 e T2, se ambas as funções compartilharem a mesma base de tempo.

Analogamente à contagem da distância percorrida em volta de um circuito fechado, o procedimento consiste em setar no registrador `TPMx_CnV` do canal de função *Output Compare* o valor C_{T1} capturado pelo registrador `TPMx_CnV` do canal de função *Input Capture* e contar a quantidade de ciclos de contagem do contador em relação a C_{T1}. Essa contagem pode ser implementada habilitando a interrupção do canal *Output Compare* para que ele gere um evento de interrupção cada vez que o contador `TPMx_CNT` passe por C_{T1}, realizando uma contagem de ciclos de forma análoga à contagem de *overflows*.

Com a contagem por *hardware* da quantidade *M* de ciclos, juntamente com as contagens C_{T1} no instante inicial T1 e C_{T2} no instante final T2, podemos estimar com precisão o intervalo de tempo t entre as duas contagens C_{T1} e C_{T2} através das expressões:

T1 ≥ T2:

$$t = \left(M + \frac{\left(TPMx\_MOD - C_{T1}\right) + C_{T2}}{TPMx\_MOD}\right) \times Periodo \qquad (4)$$

T1 < T2

$$t = \left(M + \frac{\left(C_{T2} - C_{T1}\right)}{TPMx\_MOD}\right) \times Periodo \quad . \qquad (5)$$

Em relação à contagem de intervalos de tempo por *overflows*, requer-se nessa alternativa a alocação de um canal configurado com a função *Output Compare* para contar M, como mostra o seguinte pseudo-

código de contagem de quantidade de ciclos completos de TPMx na rotina de serviço que trata as capturas de um canal TPMx_CHn

```
Se (flag de interrupção do canal de Input Capture  estiver em 1) então
  Se for a primeira captura então
     C_T1 ← TPMx_CnV;
     M ← 0;
     Ativar e habilitar um canal de Output Compare de TPMx;
  Se for a segunda captura então
     C_T2 ← TPMx_CnV;
     Computar Eq. (4) ou (5);
Se (flag de interrupção do canal de Output Compare estiver em 1) então
  M← M+1;
```

PWM em Geração de Sinais Audíveis

As frequências de modulação para as notas musicais variam de acordo com a escala musical e a temperatura utilizada. Para a escala de 440Hz, temos as seguintes frequências:

- Dó (C) : 261.63 Hz (T=0,0038s)
- Ré (D) : 293.66 Hz (T=0,0034s)
- Mi (E) : 329.63 Hz (T=0,003s)
- Fa (F) : 349.23 Hz (T=0,0029s)
- Sol (G) : 392.00 Hz (T=0.0026s)
- Lá (A) : 440.00 Hz (T=0.0023s)
- Si (B) : 493.88 Hz (T=0.002s)
- Dó (C) : 523.25 Hz (T=0.0019s)

Podemos gerar sinais de áudio de notas musicais usando a função PWM de TPM. **Para cada nota musical, configuramos como o período do TPM o período da nota.** Em seguida, setamos no registrador de dados `TPMx_CnV` um valor que resulte numa forma de onda retangular de frequência da nota musical com um ciclo de trabalho adequado para produzir um tom audível. Por exemplo, se configurarmos MCGFLLCLK (20.971.520 Hz) como a fonte de sinais de relógio e 32 como divisor *prescaler* do contador LPTPM, podemos gerar um som audível da nota Lá num *buzzer* conectado no pino PTE21, se

- multiplexarmos o pino PTE21 para o canal `TPM1_CH1`,
- configurarmos o a contagem máxima `TPM1_MOD` em ((20971520)/(32*440),
- configurarmos a função EPWM para o canal `TPM1_CH1` via o registrador de configuração TPM1_C1SC, e
- setamos no seu registrador de dados `TPM1_C1V` uma percentagem menor que 100% da contagem máxima.

Parametrização de Blocos de Dados

KL25Z dispõe de 3 módulos de TPM cujos registradores são mapeados no espaço de memória 0x40038000-0x003A088:

```
#define TPM0_BASE_PTR                ((TPM_MemMapPtr)0x40038000u)
/** Peripheral TPM1 base pointer */
#define TPM1_BASE_PTR                ((TPM_MemMapPtr)0x40039000u)
/** Peripheral TPM2 base pointer */
#define TPM2_BASE_PTR                ((TPM_MemMapPtr)0x4003A000u)
```

No ambiente CodeWarrior este espaço é abstraído em três blocos do tipo de dado `struct TPM_MemMap` definido no arquivo `MKL25Z.h` [11]. Através desse tipo de dados são definidas as macros que nos permitem acessar os registradores pelos mesmos nomes usados no manual de referência [2].

```
typedef struct TPM_MemMap {
  uint32_t SC;                /**< Status and Control, offset: 0x0 */
  uint32_t CNT;               /**< Counter, offset: 0x4 */
  uint32_t MOD;               /**< Modulo, offset: 0x8 */
    struct {                  /* offset: 0xC, array step: 0x8 */
      uint32_t CnSC;  /**< Channel (n) Status and Control, array offset: 0xC, array step: 0x8 */
      uint32_t CnV;           /**< Channel (n) Value, array offset: 0x10, array step: 0x8 */
   } CONTROLS[6];
    uint8_t RESERVED_0[20];
    uint32_t STATUS;          /**< Capture and Compare Status, offset: 0x50 */
   uint8_t RESERVED_1[48];
    uint32_t CONF;            /**< Configuration, offset: 0x84 */
} volatile *TPM_MemMapPtr;
```

Além disso, há uma série de macros pré-definidas que nos permite "parametrizar" os blocos de dados referentes aos três módulos TPMx Se definirmos um vetor de ponteiros ao tipo de dado `TPM_MemMapPtr` no nosso código, como

```
TPM_MemMapPtr moduloTPM[] = TPM_BASE_PTRS,
```

podemos usar `TPM[x]→SC`, com `x` variando de 0 a 2, para acessarmos o registrador `SC` de cada módulo `TPMx` ao invés de usarmos separadamente as macros `TPM0_SC`, `TPM1_SC` e `TPM2_SC`. Os dois registradores `CnSC` e `CnV` referentes a cada um dos 6 canais de um módulo são agrupados, por sua vez, numa outra `struct` de cujo tipo foi declarada um vetor CONTROLS de 6 elementos.

Com o uso das seguintes macros, também definidas no arquivo `MKL25Z.h`,

```
#define TPM_CnSC_REG(base,index)   ((base)->CONTROLS[index].CnSC)
#define TPM_CnV_REG(base,index)    ((base)->CONTROLS[index].CnV),
```

os acessos individuais aos registradores de cada canal `n` podem também ser parametrizados. Em vez de usar `TPM1_C0SC`, `TPM1_C0V`, `TPM1_C1SC` e `TPM1_C1V`, podemos utilizar as alternativas `TPM_CnSC_REG (moduloTPM[1], 0)`, `TPM_CnV_REG (moduloTPM[1], 0)`, `TPM_CnSC_REG (moduloTPM[1], 1)` e `TPM_CnV_REG (moduloTPM[1], 1)`, respectivamente.

Podemos usar também os membros das estruturas definidas para acessar os mesmos registradores:
`moduloTPM[1]->CONTROLS[0].CnSC`
`moduloTPM[1]->CONTROLS[0].CnV`
`moduloTPM[1]->CONTROLS[1].CnSC`

moduloTPM[1]->CONTROLS[1].CnV.

Observe que nas duas últimas alternativas, fazemos referências aos mesmos membros de dois blocos de dados distintos por meio de parametrização. Essa abordagem simplifica a parametrização das funções, como na implementação das funções `TPM_config_especifica` e `TPM_CH_config_especifica` nos projetos `rot8_example1`, `rot8_example2` e `rot8_aula_PWM` e `rot8_aula_ICOC`.

Geração de Números Aleatórios

Um gerador de números aleatórios produz uma sequência de números sem seguir uma ordem determinística. Esse processo pode ser implementado tanto por *hardware* quanto por *software*. A imprevisibilidade desses valores os torna muito atraente para aplicações em que comportamentos aleatórios são desejáveis, como em jogos de azar, simulações, criptografia e geração de dados para teste de *software* e treinamento de máquina.

A função `rand` disponível na biblioteca-padrão de C, `stdlib.h`, é capaz de gerar **números inteiros pseudoaleatórios** em um intervalo especificado. Eles são chamados pseudoaleatórios porque são gerados por um procedimento determinístico a partir de uma semente inicial. Para aumentar a imprevisibilidade da sequência de números gerados, é comum usar a função `srand` para gerar diferentes sementes à função `rand` [18].

Outra abordagem para explorar a imprevisibilidade é utilizar os números lidos dos contadores integrados nos temporizadores dos microcontroladores. Essa imprevisibilidade surge de diversos fatores externos, como interrupções, variações de frequência, flutuações de temperatura, oscilações de tensão e até mesmo a execução de instruções que acessam o contador.

Um algoritmo simples para gerar um valor aleatório de `n` *bits* envolve extrair os `n` *bits* menos significativos do valor lido de um contador que é pulsado por um sinal de relógio de alta frequência. Por exemplo, podemos utilizar o contador `TPM1_CNT` para gerar um valor aleatório entre 0x3E8 e 0xFFF, amostrando **repetidamente** o conteúdo do contador até encontrar um valor maior que 0x3E8.

```
aleatorio = (TPM1_CNT & 0xFFF);
while (aleatorio < 0x3E8) aleatorio = (TPM1_CNT & 0xFFF);
```

Proteção de Ações Indevidas dos Usuários usando Máquinas de Estado

No roteiro 7 [7], discutimos como modelar um sistema como uma máquina de estados, definindo cuidadosamente as ações permitidas em cada estado para proteger as regiões críticas. Podemos aplicar uma estratégia semelhante para proteger nosso sistema de ações inadequadas por parte dos usuários. Ao implementar as regras de restrição nas ações dos usuários e nas transições entre os estados do sistema, podemos previnir que ações potencialmente prejudiciais levem o sistema a estados não previstos, evitando danos inesperados. O projeto `tempo_reacao` exige que as reações das pessoas, por meio de pressionamento no botão IRQA12, sejam sincronizadas com os estímulos gerados pelo microcontrolador. Para proteger o sistema do processamento das ações não-sincronizadas sobre o botão, podemos desabilitar as interrupções geradas por tais entradas e, quando não for possível, definir as ações permitidas para cada estado e implementar nas rotinas de serviço que tratam tais entradas as regras de validação do estado do sistema. Isso permite que o processamento de uma entrada indevida seja interrompido. No entanto, é importante que todos os tratamentos sejam cuidadosamente ponderados para evitar descartes equivocados dos eventos de interrupção e para garantir a transparência e a fluidez na operação do sistema.

EXPERIMENTO

Neste experimento vamos desenvolver o projeto de tempos de reação audível, chamado de `tempo_reacao`, que segue a seguinte dinâmica:

a) São gerados intervalos de tempo de espera aleatórios e exibida a mensagem "Pressione IRQA12" no visor do LCD (estado `PREPARA_INICIO`).

b) Aguarde o acionamento de IRQA12 (estado `INICIO`). Ao pressionar o botão IRQA12, a mensagem "Teste Auditivo" é mostrada no visor (estado `PREPARA_AUDITIVO`) e o sistema entra no estado de preparação para o testo auditivo, aguardando um intervalo de tempo **aleatório** (estado `ESPERA_ESTIMULO_AUDITIVO`).

c) Após o término do intervalo aleatório, é gerado **um estímulo auditivo** e o valor do contador C_{T1} é armazenado no instante em que o estímulo é iniciado. O *buzzer* (Figura 6), conectado no pino 1 do *header* H7 do *shield* FEEC-EA871 [4], emite um som audível e aguarda a reação do usuário através do pressionamento do botão IRQA5 (estado `ESPERA_REACAO_AUDITIVA`).

Figura 6: *Buzzer* piezoelétrico.

d) Ao pressionar IRQA5, a contagem C_{T2} é capturada. O tempo de reação medido é mostrado no visor do LCD (estado `RESULTADO`) e o sistema entra em um estado de espera de **aproximadamente** 3 segundos para que os valores possam ser lidos (estado `LEITURA`). Nesse intervalo de 3 segundos, o *buzzer* reproduz uma música de sua escolha.

e) Volta para (a), reiniciando o ciclo.

Figura 7 sintetiza os 7 estados do sistema que foram usados para modelar a dinâmica do sistema.

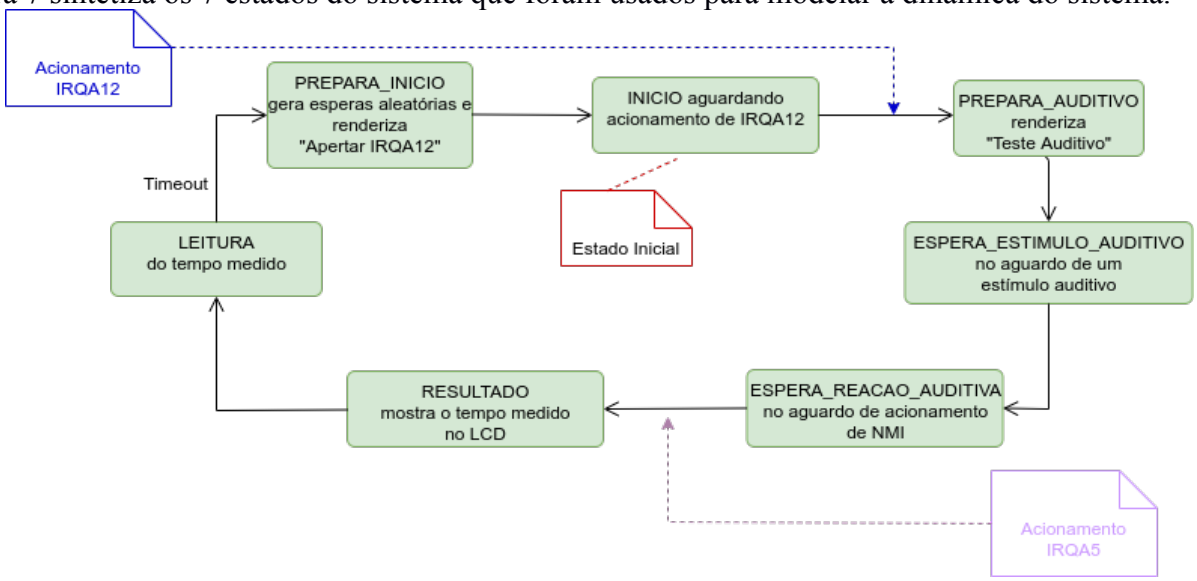

Figura 7: Diagrama de máquina de estados do projeto `temp_reacao` (editado em [12]).

A Figura 8 mostra os componentes em *hardware* (vermelho) e *software* (preto) necessários para a implementação do projeto no *kit* disponível no nosso laboratório.

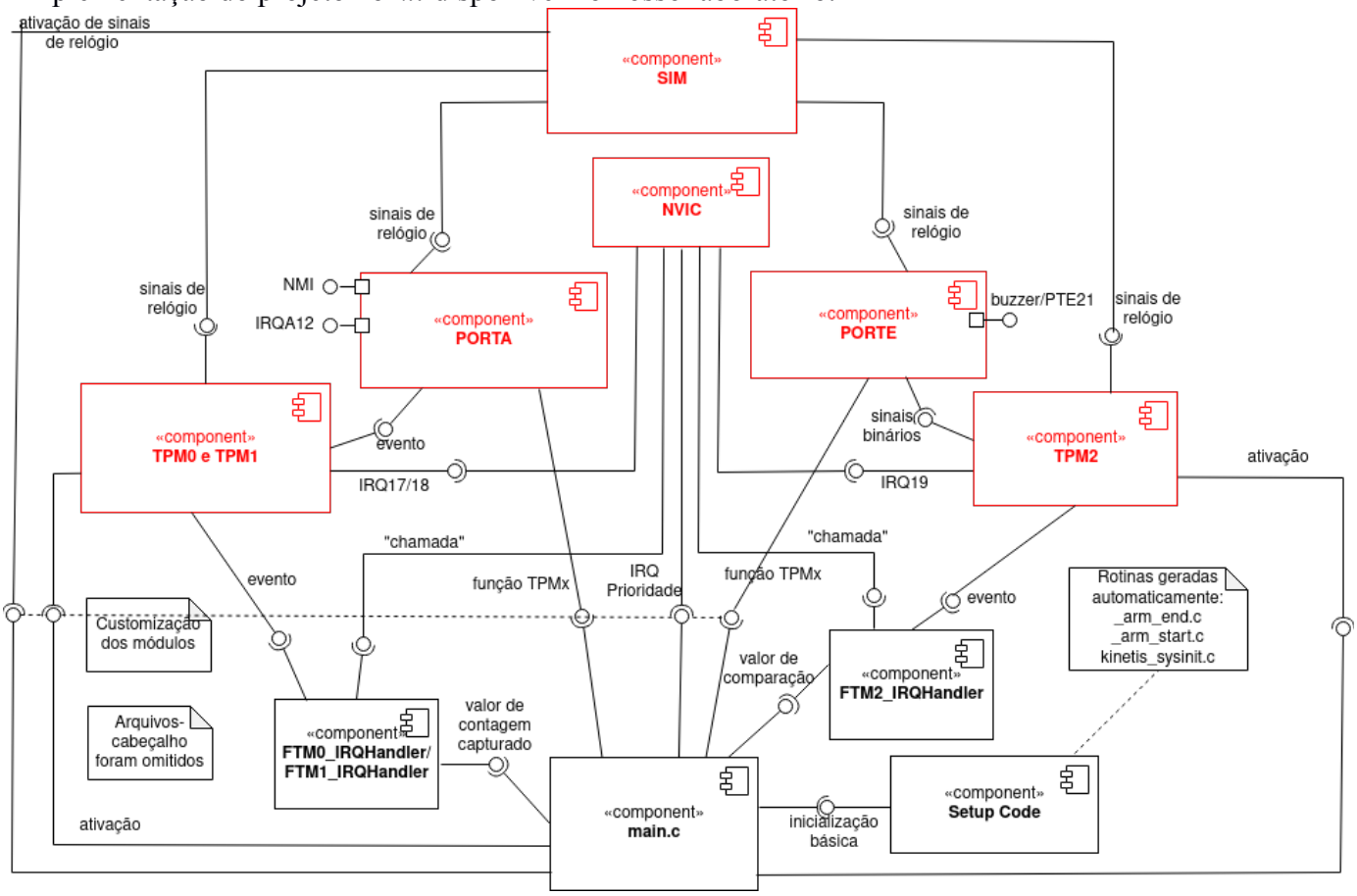

Figura 8: Diagrama de componentes do projeto `temp_reacao` (editado em [12]).

A fonte de sinais de relógio é MCGFLLCLK 20.971.520Hz. No estado `ESPERA_REACAO_AUDITIVA`**, a frequência de TPM1, com o divisor *prescaler* setado em 128, é configurado em 440Hz para gerar um sinal audível correspondente à nota LÁ da escala de 440Hz. No estado** `LEITURA`**, a frequência de TPM1 varia como no** `rot8_aula_PWM` [5] **para a reprodução de uma música.**

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 `rot8_aula_PWM` [5] demonstra o uso do módulo TPM1 para gerar um sinal PWM no seu canal `TPM1_CH1`, que é direcionado ao pino 3 do *header* H5 do *shield* FEEC-EA871 [4] . O período do módulo TPM1 é alterado ciclicamente de acordo com uma lista predefinida de períodos, correspondentes às notas musicais. Simultaneamente, o valor no registrador `TPM1_C1V` é modificado para manter o ciclo de trabalho em 50%. Ao conectar o LED RGB externo ao *header* H5, o LED ligado ao pino 3 se alternará entre estar aceso durante meio período e apagado durante o outro meio período. Se trocarmos o LED pelo *buzzer* mostrado na Figura 6, conectando-o aos pinos 3, 4 e 5 do mesmo *header*, ouviremos um som com variações.

O projeto `rot8_aula_ICOC` [6] exemplifica o uso do par *Input Capture* (`TPM0_CH1`) e *Output Compare* (`TPM0_CH2`) para calcular intervalos de tempo com precisão. No circuito, o canal `TPM0_CH1` está associado ao pino `PTA4`, onde está conectada o botão `NMI`. Devido à indisponibilidade de acesso aos pinos que servem o canal `TPM0_CH2`, refletimos o sinal deste canal no canal `TPM1_CH1`, conectado ao pino 3 do *header* H5 do *shield* FEEC-EA871 [4]. Nesse

contexto, a interrupção do canal TPM0_CH2 é ativada no primeiro pressionamento do botão NMI e desativada no segundo. Durante o intervalo entre esses dois pressionamentos, o canal TPM0_CH2 gera interrupções periódicas para contagem de tempo, mantém a saída em estado '1'; caso contrário, retorna ao estado '0'. E esse comportamente é refletido no canal TPM1_CH1. Se ligarmos o LED RGB externo ao *header* H5, onde o canal R do LED está conectado ao pino 3, podemos ver o LED R piscando. Para verificar o período configurado para TPM0 e TPM1, configuramos o pino PTE22 (pino 2 do *header* H5) como GPIO, cujo estado é alternado a cada interrupção do canal TPM0_CH2.

1.a **Módulo TPM**: Como são configuradas as fontes de relógio e frequências para os sinais de barramento (*bus clock*) e os sinais de relógio dos módulos TPM0 e TPM1 no projeto rot8_aula_ICOC e TPM1 no projeto rot8_aula_PWM?

1.b **Configuração de um Período em TPM:** Quais são os valores configurados nos registradores TPMx_MOD e TPMx_SC_PS dos três módulos TPMx nos dois projetos? Use um par de marcadores do analisador lógico para certificar se os períodos do sinal gerado é condizente com as configurações. Qual é o tipo de contagem programado para os 3 módulos?

Pela forma de onda gerada pelo rot8_aula_PWM, explique a diferença na variabilidade da cor e do som gerados pelo mesmo projeto.

Ao observar as formas de onda geradas pelo rot8_aula_ICOC nos pinos 2 e 3, explique a alternância observada entre vermelho e amarelo, mesmo que os LEDs controlados são os R e G.

1.c **Funções Programáveis nos Canais de TPMx**: Quais são os valores configurados nos registradores TPMx_CnSC dos quatros canais utilizados nos dois projetos? São condizentes com as funções especificadas?

Para o canal TPM1_CH1, qual registrador é usado para configurar o ciclo de trabalho? Ao invés de configurarmos 50% do período, usarmos 85% do período ou 25% do período, o que acontecerá com a altura do som?

Para o canal TPM0_CH1, onde é armazenado o valor capturado? Para o canal TPM0_CH2, onde é armazenado o valor de comparação com o contador TPM0_CNT? Por quê é altamente recomendável usar canais de um mesmo módulo para constituir um par *input capture-output compare* na implementação de um cronômetro?

1.d **Alocação de Pinos para TPMx**: Quais pinos são alocados aos canais utilizados nos dois projetos? Como eles são configurados? É possível um canal operar sem pinos associados? Observe que todos os pinos de saída estão com o *bit* POTRx_PCRn_DSE (*drive strength enable*) setado em '1' para aumentar a corrente de saída.

1.e **Processamento de Interrupções em TPMx**: Os módulos TPMx são capazes de gerar eventos do contador quando a contagem atinge o valor máximo e eventos por canal. Apenas quando eles estiverem habilitados, geram-se solicitações de interrupção para NVIC desde que essas solicitações são liberadas no lado do NVIC. Analise os códigos dos dois projetos, Identifique os eventos habilitados para solicitação de interrupções. Quais dos 3 módulos tem suas solicitações liberadas do lado do NVIC? Como cada um desses eventos são tratados?

1.f **Cômputo de Intervalo de Tempo entre dois Eventos de Interrupção de *Input Capture* por *Output Compare***: Execute rot8_aula_ICOC e capture os sinais usando um analisador lógico por um período de 10 segundos enquanto o botão NMI é acionado seis vezes. Use 1 par de marcadores para mostrar, em unidade de tempo, a largura de um intervalo dos 3 pulsos mostrados no pino 3 do *header* H5. Compare o valor medido com o tempo correspondente à quantidade de períodos do TPM0 contabilizados no canal 2, espelhados no pino 2 do *header* H5. Qual é a diferença esperada entre os dois tempos? A diferença observada está condizente com a esperada.

Dica: Coloque um *breakpoint* no final do bloco de código de tratamento do segundo pressionamento da botoeria para acessar os valores computados das variáveis counter e valor1, e o valor do registrador TPM0_C2V.

1.g **Parametrização de Blocos de Dados**: Analise a implementação das funções TPM_config_especifica e TPM_CH_config_especifica que permitem configurar

qualquer módulo TPMx e qualquer canal TPMx_CHn. Como são acessados os registradores específicos nos canais e nos módulos nessas duas funções?

2 **Aprender com os Exemplos dos Manuais**: Na Seção 12.4/página 127 em [14] são fornecidos dois exemplos sobre a configuração de TPMx. O Exemplo 1 [8] é a implementação do comportamenteo ilustrado na Figura 2. É uma aplicação na qual os canais `TPM1_CH0` e `TPM1_CH1` são configurados com a função *Input Capture* (IC) e os canais `TPM0_CH1` e `TPM0_CH2` são configurados com a função *edge-aligned* PWM (EPWM). Adicionalmente, os eventos de estouro (*overflow*) de TPM0 são designados como disparadores para a reinicialização periódica do contador `TPM1_CNT`, garantindo que o início de cada ciclo de contagem seja sincronizado com um evento de estouro de TPM0.

Já o Exemplo 2 [9] trata de uma aplicação na qual o canal `TPM0_CH2` é configurado com a função *center-aligned* PWM (CPWM). Além disso, ocorre a transferência dos dados de 3 formas de onda distintas armazenadas na memória para seu registrador `TPM0_C2V` via DMA no modo de baixo consumo energético (*Very Low Power Stop*, VLPS). O botão PTA4 é configurada para "acordar" o sistema do modo de baixo consumo e alterar as formas de onda a serem transferidas para `TPM0_CH2`.

Devido às **diferenças de ambiente** (limitações físicas de acessos aos pinos alocados para TPM0 no *shield* FEEC871), **foi usado o módulo TPM2 (CH0 e CH1) e TPM1 no lugar de TPM0 em** `rot8_example1` e `rot8_example2`. Executar os projetos no modo *Debug* do IDE *CodeWarrior* pode ajudar na análise. Outras pequenas adaptções foram feitas, como parametrização das funções relacionadas com os registradores dos módulos TPMs e ajustes na **dependência do projeto** e na **compatibilidade de códigos** (nomes das rotinas de serviço, instruções adicionais) para que sejam compiláveis no IDE CodeWarrior.

No entanto, o que se destaca nestes exemplos é a sua ampla aplicabilidade. O Exemplo 1 em [14], realizado em `rot8_example1`, apresenta o controle preciso da operação do contador LPTPM por meio de disparos externos, possibilitando que o período da sua contagem seja sincronizado com eventos externos relevantes.

Já o Exemplo 2em [14], implementado em `rot8_example2`, ilustra o uso eficiente do DMA para realizar transferências diretas de dados entre a memória e um periférico. **O DMA é uma prática muito comum devido à sua capacidade de melhorar o desempenho geral do sistema, reduzir a sobrecarga da CPU, facilitar a transferência eficiente de dados, por exemplo dos sensores, em lote, oferecer suporte a operações assíncronas e contribuir para a eficiência energética do sistema, sem a necessidade de intervenção direta do processador.** O Exemplo 2 também demonstra a capacidade de um sistema digital em gerar sinais analógicos através do uso de componentes passivos, como resistores e capacitores, em conjunto com técnicas de modulação de largura de pulso (PWM).

2.a **Disparos Externos para TPMx_CNT:** Qual foi a configuração feita nos campos `TPM1_CONF_TRGSEL`, `TPM1_CONF_CROT`, `TPM1_CONF_CSOO` e `TPM1_CONF_CSOT` do Exemplo 1 para sinsronizar uma contagem no módulo TPMx sincronizar com um evento externo? Consulte na Tabela 3-38/página 86 em [2] a fonte de disparos setada? A configuração está condizente com o esboço de ondas mostradas na Figura 2? Há outras instruções, além das de configuração, para garantir que o comportamento se repita periodicamente? Justifique.

2.b **Transferrência DMA:** O Exemplo 2 transfere as amostras de três formas de onda, quandrada, triangular e senoidal, para o canal `TPM1_CH1`, configurado com a função CPWM. Os pulsos com larguras (ciclos de trabalho) definidas pelos valores das amostras são gerados num pino de saída. As configurações para essas transferências estão implementadas em `DMA0_MemoTPM1CH1_config_especifica` dentro do `rot8_example2`. Além disso, são inseridas instruções adicionais no código do projeto para garantir a operacionalidade do programa.

2.b.1 **Módulo DMAMUX: A** Figura 3 mostra que existem 63 fontes disponíveis para transferências DMA, juntamente com 4 canais que podem operar simultaneamente. Por

conseguinte, é crucial selecionar tanto a fonte quanto o canal antes de encaminhar os dados entre a fonte e a memória. Ao setar no código de `rot8_example2`

```
DMAMUX0_CHCFG0 |= DMAMUX_CHCFG_SOURCE(55);
```

qual é a fonte habilitável para transferências via DMA? E qual dos 4 canais do DMA é selecionado para transferência?

2.b.2 **Endereços de origem e destino**: Quais são os endereços iniciais dos blocos de dados configurados em `DMA_SAR0`, `DMA_DAR0`? Qual é o tamanho dos dados em *bytes* configurados em `DMA_DCR0_SSIZE` e `DMA_DCR0_DSIZE` para cada transferência?

2.b.3 **Atualização dos endereços de origem e destino em cada transferência:** Neste exemplo, os dados são retirados de uma fonte (um *buffer* de memória) e transferidos para um destino (`TPM1_C1V`), com o endereço de memória da fonte sendo incrementado automaticamente após cada transferência, mas o endereço de destino permanece o mesmo. Para isso, como o *bit* `DMA_DCR_SINC` e o *bit* `DMA_DCR_DINC` são configurados?

2.b.4 ***Bufffers* de transferência**: Essas áreas de armazenamento temporário permitem que os dados sejam transferidos de forma suave e eficiente entre periféricos que operam em velocidades diferentes, sem que o ritmo de operação individual da fonte e do destino seja muito comprometido. Foram usados os *buffers* circulares em `rot8_example2`? E no `Examplo 2`?

2.c **Processamento de Interrupções em DMA**: A implementação `rot8_example2` inclui duas rotinas de serviço, `PORTA_IRQHandler` e `DMA0_IRQHandler`. Ao acionar o botão NMI/PTA4 ou ao finalizar uma transferência, são gerados eventos de interrupção que fazem o microcontrolador acordar do seu estado VLPS para atendê-los.

2.c.1 Sob quais condições as formas de onda geradas na saída são alteradas entre senóide, triangular e quadrada?

2.c.2 Ao mudar a forma de onda, é nessário carregar no bloco cujo endereço está setado no canal 0 do DMA as amostras da nova forma de onda. Como a cópia pode ser "longa", o bloco de instruções de cópia foi deslocado para o laço principal da função `main`. Identifique este bloco de instruções.

2.c.3 Identifique o bloco de instruções que coloca o processador no modo VLPS enquanto o DMA transfere os dados da memória para `TPM1_CH1`. Como são somente 3 instruções, podemos executá-las dentro das rotinas de serviço, ou seja, a ordem da execução das instruções altera o comportamento do sistema?

2.d **Conversão de Sinal Modulado por Largura de Pulso em Sinal Modulado por Nível de Tensão**: Os sinais gerados no pino PTE21/`TPM1_CH1` em `rot8_example2` são modulados por larguras dos pulsos. Para recurperá-los no formato de modulação por nível de tensão, devemos colocar um circuito RC, como ilustra a Figura 9. Registre as formas de onda dos sinais filtrados num osciloscópio usando R = 100Ω e variando C entre 22nF, 10uF, 22uF e 68uF para as 3 ondas. **Ao conectar um capacitor eletrolítico, fique atento à sua polaridade**. Qual é a relação entre os valores dos capacitores e o grau de suavização dos sinais filtrados?

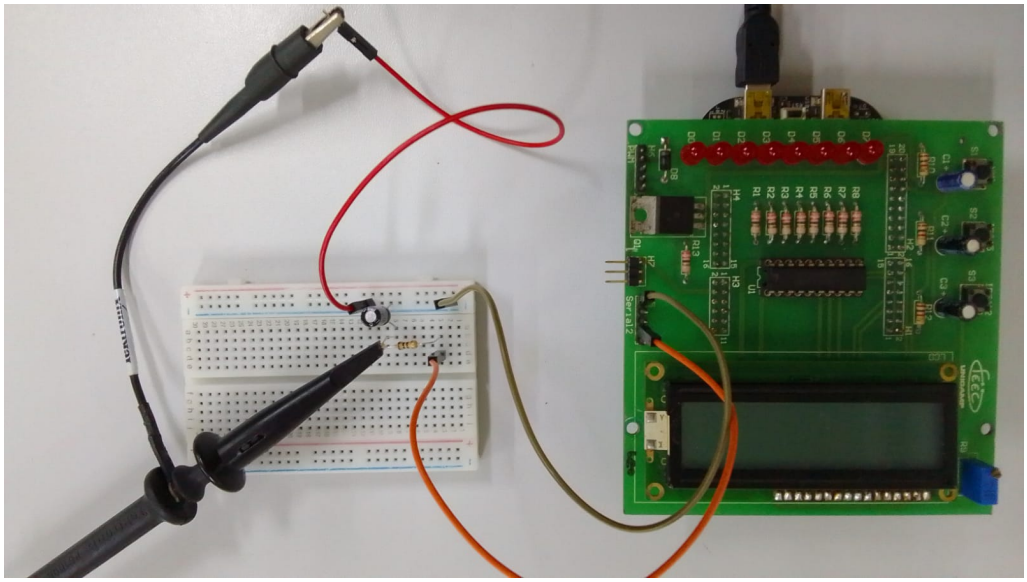

Figura 9: Filtro RC (a linha laranja ao pino 3 (o do meio) e a linha verde musgo ao pino 5 do *header* H5 do *shield* FEEC-EA871).

3 **Praticar as práticas:** Desenvolva o projeto `tempo_reacao`, em que o botão IRQA12 e o *buzzer* são conectados aos pinos PTA12 e PTE21, respectivamente, multiplexados para `TPM1_CH0` (IC) e `TPM1_CH1` (EPWM), e o pino em que o botão IRQA5 está conectado é multiplexado para `TPM0_CH2` (IC). Para **proteção de ações indevidas dos usuários**, os canais são normalmente desativados (`MsnB:MsnA:ELSnB:ELSnA` = 0b0000). Somente no estado `INICIO`, é ativado o modo IC (0b00), sensível a borda de descida (0b10), para o botão IRQA12. No estado `PREPARA_INICIO`, o canal `TPM0_CH4` (OC com saída setada em '0') é habilitado para controle de contagem de tempo a partir do instante em que IRQA12 foi pressionado.

Para **aumentar a aleatoriedade** dos momentos em que são produzidos os estímulos, um número aleatório `m` é gerado no estado `PREPARA_INICIO`. `TPM0_CH4` conta `m` ciclos antes de passar para o estado `ESPERA_ESTIMULO_AUDITIVO`. Os canais do botão IRQA5 (`TPM0_CH2`) e do *buzzer* só são ativados nos estados `ESPERA_ESTIMULO_AUDITIVO` e `ESPERA_REACAO_AUDITIVA`. Esta configuração garante que o instante do início do estímulo auditivo e o instante da reação sejam controlador por um mesmo contador `TPM0_CNT`. Para reproduzir uma música no estado `LEITURA`, aplique o procedimento implementado `rot8_aula_PWM`.

Recomenda-se os seguintes passos de desenvolvimento:

3.a **Especificação funcional:** Complete a descrição do comportamento desejado do projeto `tempo_reacao`, 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 `tempo_reacao`. 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.

Funções que podem ser úteis na implementação já se encontram implementadas em `GPIO_latch_lcd.c`, `TPM.c` e `util.c`:

```
GPIO_escreveStringLCD (uint8_t end, uint8_t *str)
TPM_habilitaInterrupTOF(uint8_t x)
ftoa(float n, char* res, uint8_t afterpoint)
```

Outra função que pode ajudar:

`void ISR_geraNumeroAleatorio(uint32_t *m)`: gerar um valor aleatório para contagem de ciclos iniciais de espera pelo estímulo.

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 de duas semanas, dividido em duas partes. Na primeira semana, é necessário responder as questões do item 2 e e realizar a especificação funcional e implementacional do projeto `tempo_reação`. **Isso inclui o cômputo dos 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 `tempo_reação`. 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 `tempo_reacao` 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] ARMv6-M Architecture Reference Manual – ARM Limited.
https://www.dca.fee.unicamp.br/cursos/EA871/references/ARM/ARMv6-M.pdf
[2] KL25 Sub-Family Reference Manual – Freescale Semiconductors (doc. Number KL25P80M48SF0RM), Setembro 2012.
https://www.dca.fee.unicamp.br/cursos/EA871/references/ARM/KL25P80M48SF0RM.pdf
[3] PWM – Modulação por Largura do Pulso
http://www.mecaweb.com.br/eletronica/content/e_pwm
[4] Nova versão do esquemático do shield FEEC
https://www.dca.fee.unicamp.br/cursos/EA871/references/complementos_ea871/Esquematico_EA871-Rev3.pdf
[5] rot8_aula_PWM.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2024/codes/rot8_aula_PWM.zip
[6] rot8_aula_ICOC.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2024/codes/rot8_aula_ICOC.zip
[7] Roteiro 7
http://www.dca.fee.unicamp.br/cursos/EA871/1s2024/roteiros/roteiro7.pdf
[8] rot8_example1.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2023/codes/rot8_example1.zip
[9] rot8_example2.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2023/codes/rot8_example2.zip
[10] Wu, S.T. Ambiente de Desenvolvimento de Software
https://www.dca.fee.unicamp.br/cursos/EA871/references/apostila_C/AmbienteDesenvolvimentoSoftware_V1.pdf
[11] Understanding embedded C: What Are Structures?

https://www.allaboutcircuits.com/technical-articles/understanding-embedded-C-what-are-structures/
[12] Diagrams.net
https://www.diagrams.net/
[13] Embedded Staff. Direct Memory Access (DMA)
https://www.embedded.com/introduction-to-direct-memory-access/
[14] Kinetis L Peripheral Module Quick Reference (Rev. 0.09/2012)
https://www.dca.fee.unicamp.br/cursos/EA871/references/ARM/KLQRUG.pdf
[15] Doxygen
https://www.doxygen.nl/manual/docblocks.html
[16] Tempo de Reação
https://mundoeducacao.uol.com.br/fisica/tempo-reacao.htm
[17] Elettroamici, Teorema de Nyquist-Shannon
https://www.elettroamici.org/pt/teorema-di-nyquist-shannon/
[18] Intellectuale. Valores aleatórios em C com a função rand
http://linguagemc.com.br/valores-aleatorios-em-c-com-a-funcao-rand/

Revisado em Fevereiro de 2024
Revisado em Fevereiro de 2023
Revisado em Março de 2022
Revisado em Maio e Julho de 2021
Revisado em Novembro de 2020