# EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

## EXPERIMENTO 6 – Relógio em Tempo Real

Profa. Wu Shin-Ting

**OBJETIVO:** Apresentação do princípio de funcionamento de temporizadores e uma aplicação em relógios.

**ASSUNTOS:** Configuração e programação do MKL25Z128 para processamento de eventos temporais.

**O que você deve ser capaz ao final deste experimento?**

Ter uma noção do módulo gerador de sinais de relógio de multipropósito.

Programar os temporizadores *PIT* e *RTC* integrados no MKL25Z128.

Entender a diferença entre diferentes formatos de representação de tempo e a conversão entre eles.

Saber converter valores inteiros em *strings* de algarismos em ASCII para serem renderizados no LCD.

Saber programar timeout com uso de um temporizador.

Conhecer uma forma de proteger as regiões críticas das interrupções.

Saber definir e usar uma função em C.

Saber reconstruir o fluxo de controle de um *software* com uso de um depurador.

Projetar um relógio digital com uso de um temporizador.

### INTRODUÇÃO

No roteiro 5 [5], foram abordados os conceitos fundamentais relativos aos temporizadores, destacando a aplicação do temporizador integrado ao núcleo/processador do KL25Z, o `SysTick` (Seção B3.3/página 275 em [2]), para controlar diversos intervalos de tempo. Neste experimento, vamos explorar dois módulos periféricos de temporizadores: `PIT` *(Periodic Interrupt Timer*, Capítulo 32/página 573 em [3]), e `RTC` (*Real Time Clock*, Capítulo 34/página 597 em [3]). Assim como o `SysTick`, esses módulos são capazes de gerar eventos que interrompem a execução do código. No entanto, ao contrário do SysTick, que gera sinais síncronos, os eventos gerados pelo PIT e RTC são assíncronos em relação à operação do processador. As seções de descrição funcional (*Functional Description*) em cada capítulo de [3] fornecem detalhes técnicos sobre o funcionamento de cada módulo. Demonstraremos a configuração e uso das funções configuradas de `RTC` e `PIT` na execução de uma tarefa específica, através da implementação de um cronômetro digital, que pode ser considerado um relógio digital de contagem regressiva.

No projeto `cronometro` o tempo desejado é setado e mostrado no meio da primeira linha do visor do LCD no formato padrão **HH:MM:SS** (24 horas) (estado `CRONOMETRPO`). Em seguida, a contagem regressiva é iniciada, decrementando o horário a cada segundo com base nas contagens feitas pelo `RTC`, até alcançar 0 e retornar ao estado `INATIVO` com `RTC` desabilitado. É possível ajustar o horário por meio de três botões, cujas **bordas de descida** são tratadas como potenciais eventos de interrupção. Sempre que se pressiona, respectivamente, os botões `NMI`, `IRQA5` e `IRQA12` no estado `INATIVO`, o sistema passa do INATIVO para os estados `INCREMENTA_HORA`, `INCREMENTA_MINUTO` e `INCREMENTA_SEGUNDO`, respectivamente.

O visor do LCD é atualizado a cada pressionamento. Se nenhum botão for pressionado dentro de um intervalo de 2,5 segundos, o RTC é habilitado e os registradores `RTC_TPR` e `RTC_TSR` resetados em '0'. Automaticamente, o `RTC` começa a contar a partir dos novos valores definidos até que a contagem atinja o valor ajustado. Em cada segundo, é atualizado no visor do LCD os valores restantes no formato HH:MM:SS. Um diagrama de máquina de estados do relógio digital especificado é mostrado na Figura 1. É importante observar que para cada estado de ajuste, há um estado de espera correspondente como uma decisão de projeto para evitar processamentos desnecessários do LCD.

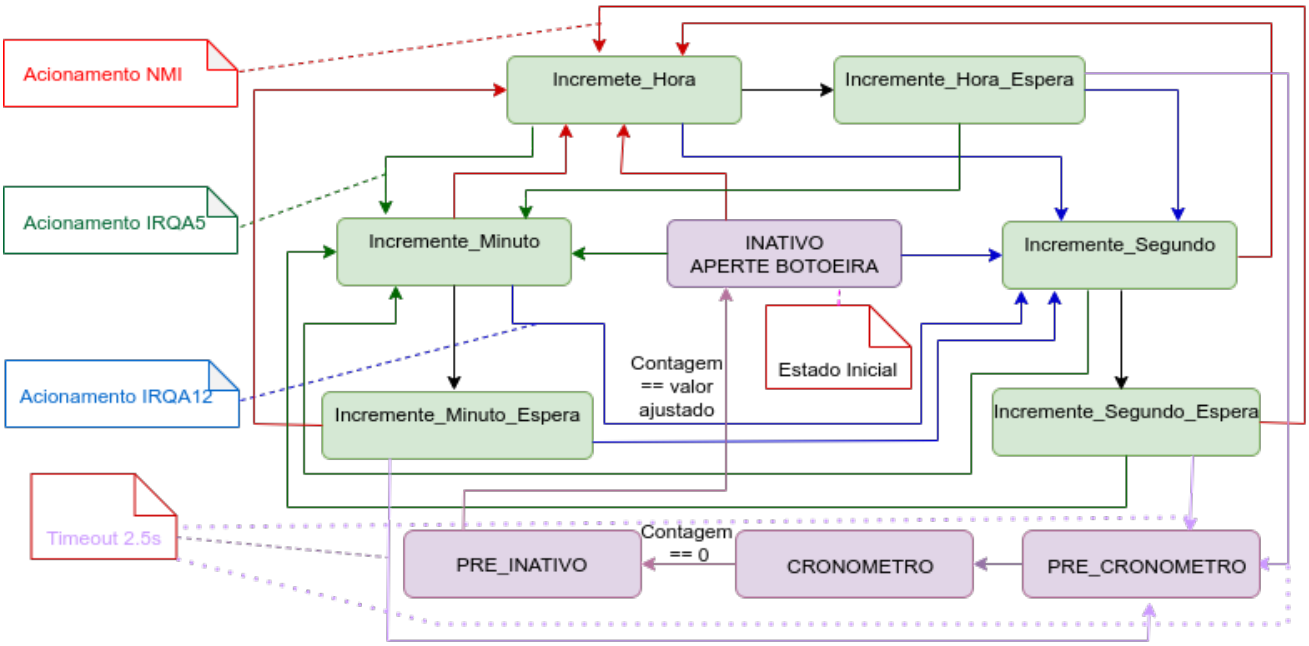


Figura 1: Máquina de estados de `cronometro` (editado em [14]).

## Módulo MCG: MCGOUTCLK

Para acomodar a variedade de velocidades de operação dos módulos periféricos integrados, o KL25Z possui o módulo *Multipurpose Clock Generator* (`MCG`), dedicado à geração das referências necessárias aos sinais de relógio desses módulos (Figura 24-1/página 370 em [3]). Um dos sinais gerados por esse módulo é o sinal **`MCGOUTCLK`** (*Multipurpose Clock Generator Output Clock*), usado para sincronizar as operações dos módulos integrados no microcotrolador. Em MKL25Z, a fonte do sinal `MCGOUTCLK` pode ser selecionada através do campo de configuração `MCG_C1_CLKS`, que oferece três opções distintas de fontes de relógio (Seção 24.3.1/página 372 em [3]): sinal de relógio de referência interna (*Multipurpose Clock Generator Internal Reference Clock,* `MCGIRCLK`), sinal de relógio de referência externa (*Oscillator External Reference Clock*, `OSCERCLK`), e sinal gerado por um laço de sincronismo, `MCGPLLCLK` (*Multipurpose Clock Generator Phase-Locked Loop Clock*) ou `MCGFLLCLK` (*Multipurpose Clock Generator Frequency-Locked Loop Clock*).

Os geradores de sinais por PLL (do inglês *Phase-Locked Loop*) ou FLL (do inglês *Frequency-Locked Loop*) são circuitos comumente aplicados para gerar um sinal de relógio estável e preciso a partir de uma fonte de relógio de referência. Eles funcionam comparando a frequência do sinal de referência com uma frequência de realimentação interna e ajustando a frequência de saída de forma a minimizar a diferença entre esses dois sinais. Isso é feito através de um circuito de controle que ajusta a frequência e a fase do sinal de saída do PLL ou FLL até que ele esteja sincronizado e bloqueado à frequência e fase do sinal de referência.

No caso do MKL25Z, um sinal por **FLL** é produzido a partir de um sinal de referência interna `MCGIRCLK` ou externa `OSCERCLK` do módulo `MCG`. Esse sinal é sincronizado com uma referência de entrada de frequência, e o circuito FLL compara a frequência do sinal de referência com a referência de enrada. O circuito ajusta então a frequência do sinal de referência para coincidir com a referência de entrada, assegurando a sincronização adequada. Por outro lado, um sinal por **PLL** é

controlado apenas pelo sinal de referência externa (OSCERCLK). Nesse caso, o circuito PLL compara a fase do sinal de entrada com a fase do sinal de saída gerado pelo circuito e ajusta a fase de saída conforme necessário para mantê-la sincronizada com a fase de entrada.

Por padrão, o sinal MCGOUTCLK é gerado por FLL, pois MCG_C1_CLKS é resetado em 0b00 (Seção 24.3.5/página 376 em [3]) e MCG_C5_PLLCKEN0 e MCG_C5_PLLSTEN0 são definidos como '0' (Seção 24.3.1/página 372 em [3])) na inicialização de MKL25Z. Nos nossos experimentos, usamos essa configuração padrão.

**Módulo PMC: LPO**

O *Power Management Controller* (PMC) é responsável por regular o fornecimento de energia para várias partes do microcontrolador e detectar baixa voltagem (Capítulo 14/página 237 em [3]). Ele coordena o *Low Power Oscillator* (LPO) que opera em frequências baixas, geralmente na faixa de kHz, adequadas para operações de baixa potência. O PMC gera um sinal de relógio de 1 kHz que está habilitado em todos os modos de operação, incluindo em quase todos os modos de baixo consumo (Seção 5.7.1/página 122 em [3]). Esse sinal de 1 kHz é comumente conhecido como sinal de relógio LPO de 1 kHz, ou sinal de relógio PMC 1kHz, ou simplesmente sinal de relógio LPO. Usamos neste experimento LPO como fonte de sinal de relógio do contador do módulo RTC.

**Módulo SIM: Distribuição de Sinais de Relógio**

Vimos nos roteiros anteiores que o módulo SIM (*System Integration Module*) é responsável pela habilitação dos sinais de relógio que controlam os módulos do KL25Z. De fato, a partir dos sinais de relógio LPO gerados pelo módulo PMC, o sinal de relógio de referência externa OSCERCLK e os sinais produzidos pelo módulo MCG, como MCGOUTCLK, MCGPLLCLK, MCGFLLCLK e MCGIRCLK, o SIM distribui uma variedade de sinais de relógio. Isso inclui sinais de relógio de interface (*bus interface clock*), como *platform clock*, *system clock*, *bus clock* e *flash clock*, bem como sinais de relógio internos (*internal clock*), como *core clock*, LPO e ERCLK32K (*External Reference Clock 32kHz*) (Tabela 5-2/página 121 em [3]). Figura 2 ilustra as opções de sinais de relógio disponíveis e como eles são gerados e conectados aos diferentes módulos do microcontrolador KL25Z.

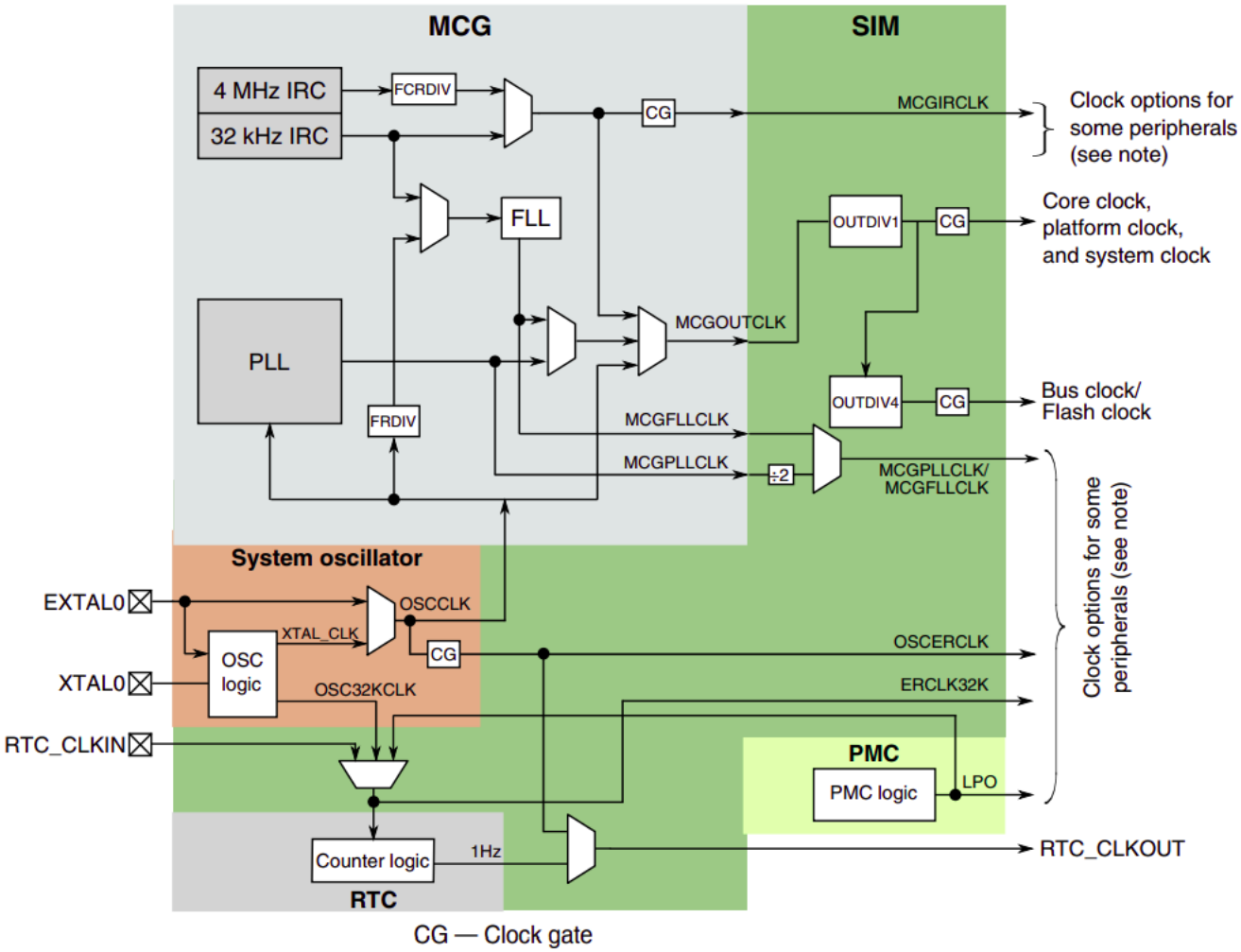


Figura 2: Diagrama de sinais de relógio disponíveis no KL25Z.

Os **sinais de relógio de interface** são destinados a sincronizar a comunicação entre os diversos módulos do KL25Z, garantindo que os dados sejam transmitidos de forma confiável e no tempo correto ao longo do barramento compartilhado. Cada tipo de módulo pode exigir uma frequência de clock específica para operar de maneira eficiente e coordenada com os outros módulos. Por outro lado, os **sinais de relógio internos** são essenciais para o funcionamento interno de certos módulos do microcontrolador. Cada módulo pode requerer uma frequência de relógio específica para suas operações internas, como a execução de instruções de processamento ou o controle de eventos temporais. Portanto, diferentes sinais de relógio permitem uma personalização mais precisa das frequências e temporizações para atender às necessidades específicas de cada módulo, otimizando assim o desempenho e a eficiência geral do microcontrolador.

Os módulos `PIT` e `RTC` compartilham o mesmo barramento cujo sinal de relógio de interface é *bus clock*. Projetados para atender propósitos distintos, eles diferem nos sinais de relógio internos que alimentam seus contadores. O `PIT` é projetado para gerar eventos de interrupções periódicas em frequências configuráveis, sem se preocupar com a manutenção do consumo de energia em estado de baixa potência. Por outro lado, o `RTC` é especificamente concebido para suportar a implementação de relógios digitais. Ele é capaz de gerar eventos de interrupções periódicas com uma precisão de 1Hz e pode garantir sua operação contínua mesmo no modo de baixo consumo de energia. Portanto, o `PIT` utiliza o mesmo *bus clock*, e o `RTC` oferece três alternativas, selecionáveis por *software*, para controlar seus contadores internos (Seção 5.7.3/página 123 em [3]).

Figura 2 mostra como o sinal de relógio de barramento (*bus clock*) é derivado do sinal `MCGOUTCLK` (20.971.520Hz). Essa derivação ocorre por meio de dois divisores localizados no módulo `SIM`, chamados de `OUTDIV1` e `OUTDIV4` (Figura 5-1/página 116 em [3]). A configuração desses divisores é realizada através do registrador de configuração `SIM_CLKDIV1` (Seção 12.2.12/página 210 em

[3]). No entanto, o divisor OUTDIV1 só pode ser configurado na inicialização, condicionado à configuração previamente definida no registrador de configuração FTF_FOPT[LPBOOT] (Seção 27.33.4/página 429 em [3]). Por padrão, a configuração inicial do divisor OUTDIV1 é definida como 1. No projeto rot6_aula [8], o valor 0b011 atribuído a OUTDIV4 configura a frequência de *bus clock* para 20.971.520Hz/4 = 5.242.880Hz.

As três opções de fontes de sinais para o sinal de relógio ERCLK32K, responsável por pulsar os contadores do RTC, são detalhadas na Seção 5.7.3/página 123 em [3]. Elas incluem o sinal OSC32KCLK gerado pelo módulo OSC (Seção 25.3/página 161 em [3]), um sinal externo RTC_CLKIN conectado no pino 1 da porta PTC (Seção 10.3.1/página 161 em [3]), e o sinal LPO de 1kHz gerado pelo controlador de gerenciamento de energia (P*ower Management Controller*, PMC) para operações nos modos de baixo consumo. Essas opções são selecionáveis pelo registrador de configuração SIM_SOPT1 (Seção 12.2.1/página 193 em [3]). A seção 4.1.3.1/página 38 em [11] demonstra como usar esse registrador para configurar as três alternativas de fontes de relógio para RTC.

**Módulo PIT**

**PIT** (*Periodic Interrupt Timer*) é um módulo com duas unidades de temporizador de 32 *bits*, as quais podem ser configuradas para operar como uma única unidade de 64 *bits* por meio do registrador de controle PIT_TCTRLn. A habilitação do módulo e sua operação no modo *Debug* são configuradas pelo registrador de controle PIT_MCR. Similar ao SysTick, a *s*ua **contagem no contador PIT_CVALn é decrescente**. Os valores máximos de contagem (REF) são carregados nos registradores de dados PIT_LDVALn. Quando as duas unidades estiverem encadeadas e os seus registradores PIT_LDVALn forem carregados com 0xFFFFFFFF, PIT operará como **temporizador vitalício** (*lifetime timer*) e os valores de contagem em 64 *bits* podem ser acessados pelos registradores de dados PIT_LTMR64H e PIT_LTMR64H.

Quando PIT_CVALn atinge 0, a *flag* de interrupção do registrador de estado PIT_TFLGn é setada em '1' e, se o *bit* de habilitação de interrupção no registrador de controle PIT_TCTRLn estiver em '1', o sinal de requisição de interrupção IRQ 22/vetor 38 (Tabela 3-7/página 52 em [3]) é ativado. A *flag* PIT_TFLGn_TIF só é resetada em '0' quando se escreve '1' nela (Seção 32.3.7/página 580 em [3]). **Para reiniciar uma contagem, é necessário desabilitar o temporizador, resetando o *bit* PIT_TCTRL_TEN para '0', e então reabilitado, setando PIT_TCTRL_TEN para '1'.** Para modificar o conteúdo de PIT_LDVALn, é opcional a desabilitação do temporizador.

Sem circuitos de *pre-* e *postscaler*, o período de interrupções $T_{PIT}$, em segundos, do PIT é derivado da frequência de *bus clock*:

$$T_{PIT} = \frac{PIT_{LDVAL}}{\frac{MCGOUTCLK}{OUTDIV1 \times OUTDIV2}} = \frac{PIT_{LDVAL} \times OUTDIV2}{MCGOUTCLK} = \frac{PIT_{LDVAL} \times OUTDIV2}{20.971.520}. \quad (1)$$

No entanto, similar ao módulo SysTick, podemos inserir, por *software*, *postscaler* e aumentar o período de $T_{PIT}$ como vimos no roteiro 5 [5].

Similar aos módulos PORTx, o sinal de relógio de PIT é desabilitado na inicialização do KL25Z (Seção 12.2.10/página 207 em [3]). É necessário habilitá-lo através do registrador de configuração SIM_SCGC6 (Seção 12.2.10/página 207 em [3]). Nas Seções 32.5/página 582, 32.6/página 583 e 32.7/página 584 em [3] há exemplos de configuração do PIT para diferentes modos de operação. Assim que o módulo for ativado, o seu circuito, em processamento paralelo do núcleo, gera eventos de interrupção periódicos na periodicidade $T_{PIT}$. Se o controlador NVIC estiver configurado para atender IRQ22, o fluxo de controle é desviado para a rotina de serviço. Consultando o arquivo

`Project_Settings/Startup_Code/kinetis_sysinit.c` gerado pelo IDE *CodeWarrior*, o nome da rotina de serviço declarado para a IRQ 22 é `PIT_IRQHandler`.

No KL25Z os eventos de interrupção gerados pelo `PIT` podem ser aproveitados como gatilhos para diversos módulos, como o `ADC` (*Analog-Digital-Converter*, Capítulo 28/página 457 em [3]), `TPM` (*Timer/PWM*, Capítulo 31/página 547 em [3]), `DAC` (*Digital-to-Analog Converter*, Capítulo 30/página 537 em [3]) e o controlador de `DMA` (*Direct Memory Access*, Capítulo 23/página 349 em [3]). A tabela 3-1/página 45 em [3] sintetiza as interconexões entre os módulos presentes em KL25Z. A habilitação dos gatilhos pode ser controlada por *software,* através de registradores de configuração localizados nos próprios módulos interconectados ou externamente a eles. Por exemplo, no caso do módulo `TPM` essa configuração é feita diretamente em `TPM` (Tabela 3-38/página 86 em [3]). Porém, a habilitação de interconexões entre o `PIT` e o módulo `ADC` é realizada por meio de um registrador específico no módulo `SIM` (Seção 12.2.6/página 200 em [3]). O Capítulo 7/página 67 em [1] apresenta uma aplicação do DMA nas transferências diretas de dados entre a memória e um periférico usando `PIT` em gatilhos de transferências.

## Módulo RTC

**O módulo RTC é um temporizador dedicado para contar os segundos de forma crescente, com a condição de que a frequência da fonte de *clock* seja 32,768kHz**. Figura 3 apresenta um diagrama de blocos do módulo `RTC`.

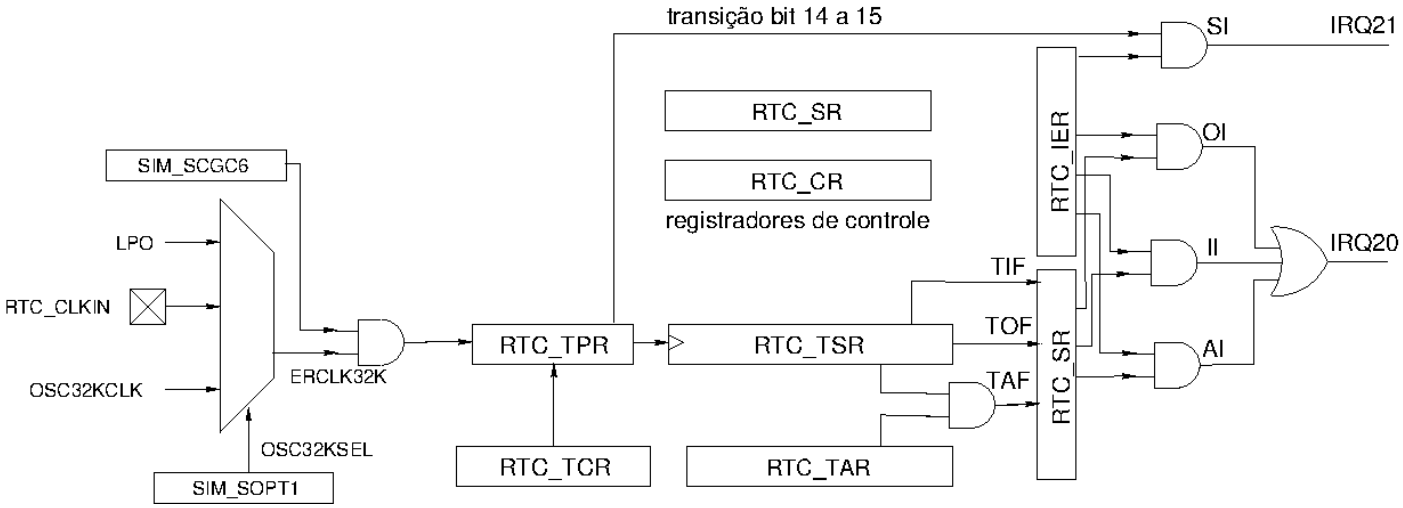


Figura 3: Diagrama de blocos do módulo `RTC`.

O seu **contador de segundos** `RTC_TSR` de 32 *bits* é atualizado a cada $2^{15}$ tiques de relógio por meio de um divisor *prescaler* cuja contagem só acontece se o contador estiver habilitado e as *flags* de *Overflow* (`RTC_SR_TOF`) e *Time Valid* (`RTC_SR_TIF`) do registrador de estado `RTC_SR` estiverem em '0'. O valor da contagem em *prescaler* pode ser acessado através do registrador de dados `RTC_TPR`. Porisso, com um sinal de relógio de frequência 32,768kHz, o período $T_{RTC}$ do circuito de *prescaler* é

$$T_{RTC} = 2^{15} \times \frac{1}{f} = 2^{15} \times \frac{1}{32768} s = 1 s. \quad (2)$$

Quando a contagem em `RTC_TSR` chega em $2^{32}$-1, pára-se o incremento. O *bit* `RTC_SR_TOF` é setado em '1' e os registradores `RTC_TSR` e `RTC_TPR` resetados em '0'. Esses dois registradores podem ser também resetados, escrevendo '1' no *bit* `RTC_SR_TIF`. Os conteúdos de `RTC_TPR` e `RTC_TSR` podem ser modificados, sempre na sequência `RTC_TPR` e `RTC_TSR`, quando o *bit* `RTC_SR_TCE` do registrador de controle/configuração `RTC_SR` estiver em '0' (contadores desabilitados) (Seção 34.3.2/página 607 em [3]).

Para aprimorar a precisão das medições temporais, o RTC possui um circuito de compensação integrado, projetado para corrigir desvios em relação aos valores nominais dentro de uma ampla faixa, variando de 0,12ppm (*parts per million*) a 3906ppm. Essa compensação é ajustável por *software*, via o registrador de configuração RTC_TCR. Além disso, o RTC é equipado com a capacidade de gerar um sinal de 1Hz, que pode ser acessado através do pino 3 da porta C (RTC_CLKOUT) (Seção 10.3.1/página 163 em [3]). Esse sinal de 1Hz pode auxiliar na calibração da compensação, garantindo uma maior precisão nos tempos medidos.

Projetado para operar com sinais de 32,768kHz, se selecionarmos a alternativa LPO de 1kHz para pulsar os contadores do RTC, o período $T_{RTC\text{-}LPO}$ do circuito de *prescaler* passa a ser

$$T_{RTC-LPO} = 2^{15} \times \frac{1}{f} = 2^{15} \times \frac{1}{1000}\, s = 32{,}768\, s. \quad (3)$$

e os incrementos no registrador de dados RTC_TSR só acontecem em cada 32,768s. É, portanto, necessário reinterpretar, por *software*, os valores lidos de RTC_TSR e RTC_TPR para extrair os segundos conforme a seguinte relação

$$Segundos = \frac{TSR \times 32768}{f_{clock}} + \frac{TPR}{f_{clock}} = \frac{TSR \times 32768 + TPR}{f_{clock}}, \quad (4)$$

onde $f_{clock}$ é a frequência do sinal de relógio. Para $f_{clock}$ = 1000Hz, um segundo corresponde a 1000 incrementos no registrador RTC_TPR (e não 32768 incrementos) e um incremento no registrador RTC_TSR acontece em cada 32,768s (e não em 1s). Em outras palavras, a contagem em segundos passa a ser

$$Segundos = \frac{TSR \times 32768}{1000} + \frac{TPR}{1000} = \frac{TSR \times 32768 + TPR}{1000}, \quad (5)$$

ou seja, a representação de *Segundos* é desmembrada em duas partes, uma no registrador RTC_TSR e outra no registrador RTC_TPR:

$$\begin{aligned} TSR &= (Segundos \times f_{clock})/32768 = (Segundos \times 1000)/32768 \\ TPR &= (Segundos \times f_{clock})\,\%\,32768 = (Segundos \times 1000)\,\%\,32768 \end{aligned} \quad (6)$$

**Observe que, embora a segunda e a terceira expressões nas Eqs. (4) e (5) sejam matematicamente equivalentes, elas produzem resultados distintos em operações inteiras dos processadores. Na segunda expressão, temos a soma de dois termos truncados, enquanto na terceira expressão, ocorre o truncamento da soma. O erro acumulado na segunda expressão é maior do que na terceira.**

Assim como os módulos PORTx e PIT, o sinal de relógio de RTC é desabilitado na inicialização do KL25Z (Seção 12.2.10/página 207 em [3]). É necessário habilitá-lo através do registrador de configuração SIM_SCGC6 (Seção 12.2.10/página 207 em [3]). Assim que o módulo for ativado com as *flags* RTC_SR_TOF e RTC_SR_TIF em '0', o seu circuito inicia a contagem em processamento paralelo do núcleo, mantendo os valores de tempos atulizados em RTC_TSR e RTC_TPR.

Há no RTC um circuito de alarme que seta o *bit* RTC_SR_TAF em '1' quando a contagem em RTC_TSR alcança o valor configurado no registrador de dados RTC_TAR (Seção 34.3.4/página 608 em [3]). Através do registrador de controle RTC_IER, habilita-se solicitações de interrupções para as *flags* de interrupção RTC_SR_TOF (RTC_IER_TOIE), RTC_SR_TIF (RTC_IER_TIIE) e RTC_SR_TAF (RTC_IER_TAIE), e também para os incrementos em RTC_TSR (RTC_IER_TSIE). Este é o único módulo que tem mais de uma IRQ associada, sendo uma dedicada para incrementos em segundos e a outra para o restante de solicitações de interrupções. As IRQs correspondentes são 20 (*Alarm*) e 21 (*Seconds*) (Tabela 3-7, página 53, em [3]). Se o controlador NVIC estiver configurado

para atender a linha de interrupção solicitante, o fluxo de controle é desviado para a rotina de serviço. Consultando o arquivo `Project_Settings/Startup_Code/kinetis_sysinit.c`, gerado pelo IDE *CodeWarrior*, os nomes das rotinas de serviço declarados para as IRQs 20 e 21 são, respectivamente, `RTC_Alarm_IRQHandler` e `RTC_Seconds_IRQHandler`.

**Incremento de Segundos por *Polling***

Para um mesmo *hardware*, pode-se programar `RTC` para ter comportamentos distintos que demandam diferentes fluxos de controle. Por exemplo, se for selecionado o sinal `LPO` para ser a fonte do sinal de relógio do `RTC`, a resolução dos eventos de interrupção *Seconds* e de `RTC_SR_TAF` passa de 1s para 32,768s. Não podemos mais aproveitar o circuito de interrupção por segundos implementado no módulo `RTC` para atualizar os horários por segundo. Uma solução é fazer esta atualização por *polling*, amostrando periodicamente os valores nos registradores `RTC_TSR` e `RTC_TPR` e compará-los com o valor anterior até que a diferença entre o valor lido e o último valor salvo seja 1s, como se faz na função `main` do projeto `rot6_aula` [8].

**Conversão entre Segundos e HH:MM:SS**

Embora representar os horários em segundos simplifique o circuito do `RTC`, é importante considerar que os usuários estão mais familiarizados com os formatos de hora convencionais, tanto em 24 horas como em 12 horas (formato (DIA:)HH:MM:SS [4]). Nos ajustes dos horários de um relógio, geralmente pensamos em ajustar separadamente as unidades de tempo em um dia, ou seja, horas (HH), minutos (MM) e segundos (SS), em vez de lidar diretamente com segundos como no módulo `RTC`. Portanto, é essencial criar interfaces que permitam uma transição suave entre essas representações para tornar o funcionamento da máquina o mais transparente possível para os usuários. Isso não apenas facilita a curva de aprendizado, mas também melhora a aceitabilidade do produto.

Para simplificar a interação dos usuários, converter o formato de segundos para o formato HH:MM:SS é fundamental para criar uma interface amigável e intuitiva. Isso permite que os usuários ajustem os horários por unidade de tempo que é **cognitivamente mais acessível**. No entanto, é importante garantir que os horários ajustados sejam convertidos de volta para o formato compreendido pelo módulo `RTC`, utilizando esses valores para atualizar os registradores `RTC_TPR` e `RTC_TSR` conforme necessário. Graças à aritmética modular, essa conversão entre segundos e o formato (DIA:)HH:MM:SS pode ser realizada de forma direta:

- de `segundos` para (DIA:)HH:MM:SS (a implementação em [4] assume, por padrão, que o valor de segundos seja menor ou igual a 86400)
  DIA = `segundos`/86400;
  sec = `segundos`%86400;
  SS = sec%60;
  MM = (sec/60)%60;
  HH = sec/3600.
- de (DD:)HH:MM:SS para `segundos`
  `segundos` = DD*86400+HH*3600+MM*60+SS;

No projeto `rot6_aula` [8] foi declarado em `ISR.c` um vetor de 4 elementos `hor` para simplificar o processamento individual das 4 unidades de tempo, DIA, HH, MM e SS. No arquivo `util.c`, temos a função `ConvertSectoDay` que implementa a conversão de segundos para formato HH:MM:SS e a função `ConvertDaytoSec`, que faz o inverso, convertendo o formato HH:MM:SS para segundos.

**Conversão de Inteiros para *Strings* na Base Decimal**

A exibição dos horários no visor do LCD requer a conversão dos valores das horas, minutos e segundos em *strings* de dígitos ASCII, para garantir que sejam apresentados de forma legível. No

processo, é importante lembrar que o **LCD renderiza apenas os *bitmaps* da Tabela de Fontes (CGROM) cujos endereços estão na memória DDRAM**. Portanto, o projetista deve decidir como esses valores serão escritos na DDRAM para produzir o resultado desejado no visor.

Para converter números inteiros em *strings* de caracteres, um algoritmo típico envolve duas etapas: primeiro, extrair os dígitos do valor original por meio de divisões sucessivas por 10 [7]; e segundo, adicionar o valor numérico de cada dígito ao valor numérico de '0'==0x31==48 para obter o código ASCII correspondente. Uma implementação desse algoritmo em C é apresentada em [10], onde é necessário especificar previamente a quantidade de dígitos que o número inteiro possui.

No entanto, considerando que as unidades de tempo em um horário sempre são representadas por dois dígitos, mesmo quando o valor é menor que 10, foi desenvolvida uma implementação personalizada para aproveitar essa peculiaridade. Essa implementação está disponível no arquivo `util.c`, na função `ConvertSectoDayString`.

## Regiões Críticas

Nos relógios digitais, os ajustes de horário são frequentemente realizados por meio do pressionamento dos botões, como demonstrado no projeto `rot6_aula` [8]. Esses eventos assíncronos podem interromper o fluxo de controle durante a execução de instruções de máquina, por exemplo, entre as instruções que atualizam o visor do LCD. Essa interrupção pode resultar na modificação do conteúdo do vetor `hor` durante o tratamento de uma interrupção, fazendo com que os valores exibidos no visor se tornem inconsistentes ao retornar para o fluxo de controle principal.

Esse problema ocorre porque tanto o bloco de instruções para tratamento de interrupções quanto o bloco de instruções principal acessam o mesmo vetor `hor`. Para evitar que as ações de um bloco interfiram indevidamente nas operações do outro, é essencial garantir que seus **processamentos** sejam **mutuamente exclusivos**. Esses blocos de instruções são conhecidos como **regiões críticas**. Proteger essas regiões críticas contra interrupções indesejadas é um desafio comum na programação de microcontroladores [13].

Uma solução comum na programação de microcontroladores é desabilitar as interrupções indesejadas durante a execução de uma região crítica. Isso pode ser feito nos circuitos que geram e tratam as interrupções nos módulos do KL25Z.

Os mecanismos de interrupção implementados na maioria dos módulos do KL25Z envolvem dois circuitos principais: o **circuito de geração de interrupção**, responsável por configurar a solicitação de interrupção, e o **circuito de tratamento de interrupção**, que configura o atendimento da interrupção. Portanto, a desabilitação das interrupções pode ser realizada em dois pontos: na geração dos eventos de interrupção pelos módulos e no tratamento desses eventos pelo controlador `NVIC` (Seção B3.4/página 281 em [2]). Por exemplo, no caso de eventos gerados pelos sinais digitais de propósito geral, como os dos botões, podemos descartá-los na fonte resetando o campo `PORTA_PCRn_IRQC` em 0b0000, ou podemos desabilitar a linha de requisição de interrupção IRQ30 que chega em `NVIC` escrevendo '1' no *bit* 30 do registrador de configuração `NVIC_ICER`.

Em `rot6_aula` [8] as duas instruções de ler o estado e o horário atual do sistema definem uma região crítica. Para garantir que as duas informações, estado e horário, se mantenham consistentes, a região é protegida de interrupções com a desabilitação de interrupções antes de iniciar a execução da região. Após a execução, as interrupções são reativadas.

## Frequência em Atualizações do RTC

Como os registradores só podem ter seus valores modificados quando o contador de tempo estiver desabilitado (Seção 34.3.2/página 607, em [3]), sempre que formos atualizar os horário no `RTC` precisamos executar duas instruções de acesso ao registrador de controle `RTC_TCR`, um para desabilitar e outro para reabilitar as contagens. Portanto, por desempenho, deve-se minimizar a

frequência de atualização dos registradores RTC_TPR e RTC_TSR sem comprometer a responsividade do sistema.

Em rot6_aula [8] processamos os ajustes nas unidades de horas, minutos e segundos somente no vetor hor enquanto um usuário interage com o relógio. Postergamos as atualizações dos registradores do RTC, em segundos, para a rotina de serviço PIT_IRQHandler, após decorridos 1.5s desde o último pressionamento em NMI. No entanto, por uma questão de usabilidade, o programa mostra na função main os valores intermediários inseridos para que os usuários saibam como o sistema interpretou suas ações. Essa abordagem proporciona uma experiência mais amigável ao usuário, permitindo uma interação mais intuitiva com o sistema.

**Frequência em Atualizações do LCD**

Para reduzir o tempo de processamento das rotinas de serviço, seguimos a **estratégia de máquina de estados** delineada no roteiro 5 [5], em que deslocamos todas as instruções com maior tempo de processamento e não críticas, como aquelas relacionadas ao LCD, para o fluxo de controle principal representado pela função main. Além disso, distinguimos os estados de regime, INCREMENTE_HORA_ESPERA, INCREMENTE_MINUTO_ESPERA e INCREMENTA_SEGUNDO_ESPERA., e os respectivos estados de transição, INCREMENTE_HORA, INCREMENTE_MINUTO e INCREMENTA_SEGUNDO, para evitar renderizações desnecessárias. Os estados de transição são responsáveis por atualizar o visor do LCD na função main, enquanto os estados de regime incrementam as unidades de tempo na rotina PORTA_IRQHandler.

Existe ainda um estado de regime adicional NORMAL, encarregado de atualizar o horário exibido no visor do LCD a cada segundo. Devido à frequência da fonte de relógio do RTC selecionada ser 1kHz, a atualização é realizada por meio de *polling*, ou seja, amostrando periodicamente os dados do RTC e renderizando esses dados no visor. No entanto, a frequência de amostragem dentro do laço de controle implementado na função main é muito mais alta do que 1Hz, resultando na renderização repetida dos mesmos valores inúmeras vezes. Para evitar essas renderizações redundantes, adotamos a **estratégia de memorização** dos dados amostrados recentemente para comparação com os valores atuais. Introduzimos em rot6_aula [8] duas variáveis para armazenar o valor amostrado anterior e o valor amostrado atual. Assim, somente quando ocorrem diferença nesses valores é que o visor do LCD é atualizado, reduzindo a quantidade de renderizações desnecessárias e melhorando a eficiência do sistema como um todo.

***Timeout***

***Timeout*** é o intervalo de tempo definido em um sistema antes que um evento específico ocorra, a menos que outro evento predefinido aconteça primeiro; nesse caso, o período de espera é encerrado assim que qualquer um dos eventos ocorre. Em rot6_aula [8] a confirmação dos valores entrados em hor é implementada usando PIT. Em vez de exigir uma tecla de confirmação adicional (mais um movimento mecânico de digitação), optamos por uma abordagem baseada na ausência de novas entradas. Se decorrer um período de tempo superior ao limite permitido para uma nova entrada, o sistema retorna ao estado do relógio, atualizando o valor do horário a cada segundo. Cada interação com o botão IRQA5 reinicia a contagem de tempo no PIT.

**Definição e Uso de uma Função**

Todas as funções em C podem retornar um valor, que pode ser o conteúdo de um endereço ou um endereço, após a sua execução. Para que um compilador possa reservar um espaço de memória adequado para o valor retornado, é necessário declarar o tipo de dado do retorno seguindo a mesma convenção de declaração das variáveis: <tipo_de_dado> para um valor do tipo_de_dado e <tipo de dado *> para um endereço de um valor do tipo_de_dado. Em rot6_aula [8], as declarações das funções em util.h e ISR.h, respectivamente

```
char *ConvertSectoDayString (uint32_t seconds, char *string)

tipo_estado ISR_LeEstado
```

indicam que `ConvertSectoDayString` retorna um endereço do tipo `char` e `ISR_LeEstado`, um valor do tipo `tipo_estado`. Para acessar os valores retornados, adota-se a mesma convenção aplicada para as variáveis: sem operador para o valor retornado; operador "endereço-de" **&** para o endereço do valor retornado, e operador "endereço-de" ***** para o conteúdo do valor retornado.

Uma função é declarada como do tipo de dado `void`, quando ela não retorna nenhum valor. Todas as rotinas de serviço pré-declaradas no IDE CodeWarrior são do tipo `void`. Em muitas implementações é comum usar a função para retornar códigos de erros detectados na execução da função, permitindo que a função seja inserida diretamente num comando condicional.

Em `rot6_aula` [8] podemos, por exemplo, verificar o estado do sistema chamando diretamente a função `ISR_LeEstado` que retorna o valor do estado do sistema em `main` (`main.c`)

```
switch (ISR_LeEstado()) {

}
```

inserimos a função `ConvertSectoDayString` como um argumento da chamada de `GPIO_escreveStringLCD` dentro da rotina `atualizaHorarioLCD` em `main.c`:

```
GPIO_escreveStringLCD (0x00, (uint8_t *)
ConvertSectoDayString(segundos, hh_mm_ss)).
```

Demonstramos ainda a aplicação do retorno de uma função na atribuição de duas variáveis `anterior` e `atual` num único comando em `main` (`main.c`):

```
anterior = RTClpo_getTime (&atual);
```

**Processamento de Argumentos passados po Valor**

Na definição da função `ConvertSectoDay` em `util.c` do projeto `rot6_aula` [8] o primeiro argmento é passado por valor. Embora esse valor seja modificado dentro da rotina, o valor da variável `seconds` que é passado para ele não é modificado no escopo de `ConvertSectoDayString`. Mais uma demonstração da aplicação de passagens por valor numa chamada de função.

**Engenharia Reversa de *Software***

Engenharia reversa é uma técnica utilizada para compreender o funcionamento de um sistema por meio da análise de sua estrutura e comportamento. Pode ser aplicada tanto na análise de códigos-fonte abertos para entender os algoritmos e estratégias aplicadas, quanto na investigação de sistemas de código fechado [17]. Nesta disciplina, iremos praticar a engenharia reversa em projetos que disponibilizam os códigos-fonte, utilizando as ferramentas de depuração disponíveis no IDE CodeWarrior (Seção 2.5/página 25 em [18]). Geralmente, o processo envolve rastrear o fluxo de controle, executando as instruções passo a passo a partir de uma linha de comando ou de uma chamada de função específica. Ao analisar como os dados são manipulados e as decisões são tomadas no código, é possível reconstruir o algoritmo original. O ponto inicial desse rastreamento é frequentemente definido como um ponto de parada, conhecido como *breakpoint*, no modo `Debug` do IDE.

## EXPERIMENTO

Neste experimento vamos implementar o diagrama de máquina de estados do `cronometro` mostrado na Figura 1. O módulo principal na implementação do `cronometro` é `RTC`, tendo como fonte de clock `LPO` para seus contadores internos. Para detectar o *timeout* dos acionamentos dos botões foi utilizado o temporizador `PIT`. Assim que um acionamento válido de um botão for detectado, é habilitado o temporizador que reinicia a contagem. Como o período máximo de um `PIT` individual é $2^{32}/20971520 = 204.8$s para um sinal de barramento (***bus clock***) **em 20,97MHz**, podemos configurá-lo, **sem *postscaler* por *software*,** para cronometrar *timeouts* de 2,5s.. Adicionalmente, são necessários os módulos `GPIOC/PORTC` para interfacear com o LCD e `GPIOA/PORTA/NVIC` para interfacear com os eventos de interrupção gerados pelos botões. Figura 3 mostra um diagrama de componentes utilizados na implementação do cronômetro digital proposto.

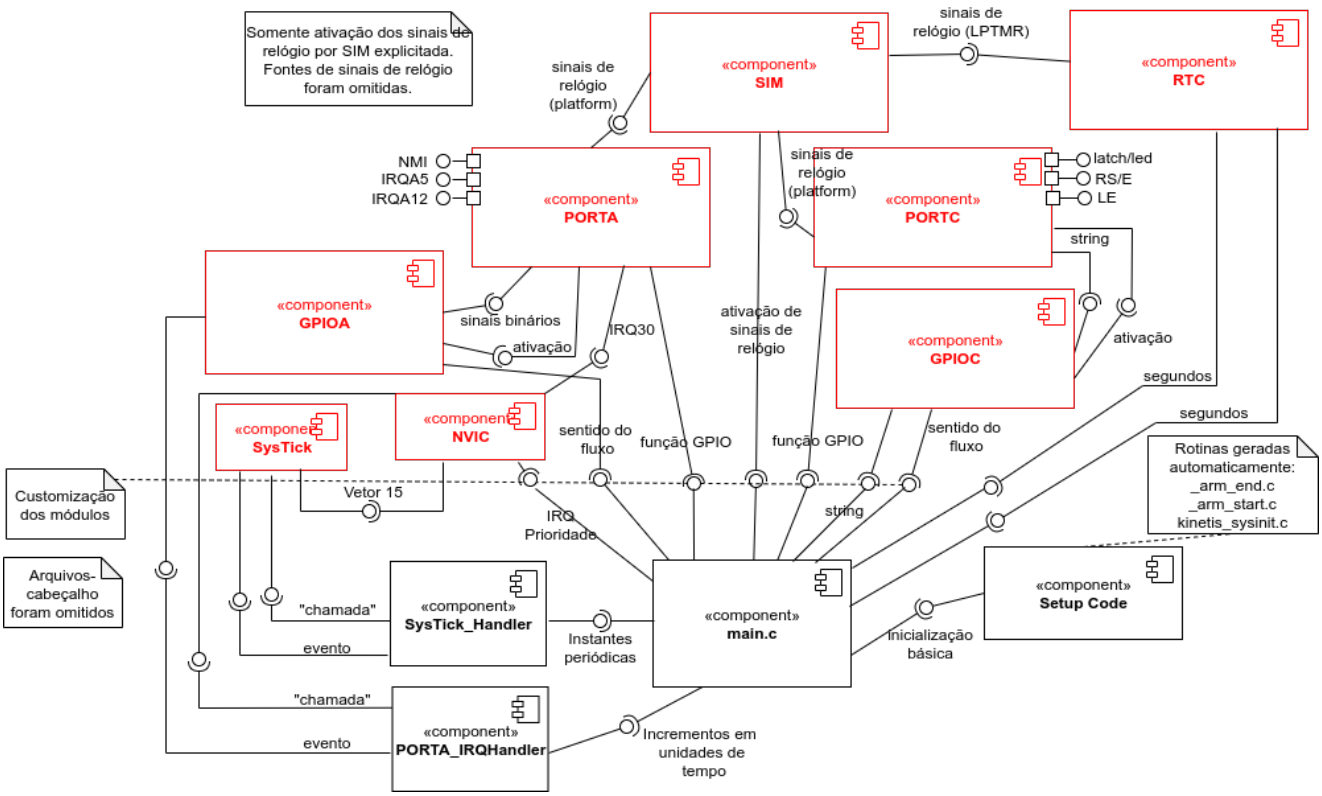


Figura 3: Diagrama de componentes do `cronometro` (editado em [14]).

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:** Carregue o projeto `rot6_aula` [8] no IDE CodeWarrior, substitua a função `espera_2us` definida em `util.c` pela função que você implementou no roteiro 3 [15] e gere o executável. Conectando o *kit* LED RGB nos pinos de H5 pode ver efeitos dos sinais gerados pelas cores dos LEDs. Conecte quatro canais do analisador lógico Saleae [13] nos pinos `PTE20` (sinal com período igual a *timeout* dos pressionamentos), `PTE21` (sinal com período configurado para `PIT`), `PTE22` (sinal com período dos eventos IRQ21) e `PTE23` (sinal com período de 1s) do microcontrolador, e o seu terra no pino 5 de H5. A pinagem do *shield* FEEC871 é mostrado em [16] e os pinos no *kit* LED RGB, da direita (LED) para esquerda (pinos), correspondem aos pinos 0 a 5 de H5.

1.a Execute o programa no modo `Run`, acione o botão IRQA5 algumas vezes e faça uma breve descrição do que você observou ao longo da execução do programa.

1.b **Módulo MCG**: Reseta a execução do projeto (Seção 2.5/página 25 em [18]) com um ponto de parada na primeira instrução da função `main`. Ao parar o fluxo neste ponto, certifique a

configuração da fonte do sinal MCGOUTCLOK através dos valores setados em MCG_C1 e MCG_C5.

1.c **Modulo SIM:** Qual é o valor assumido pelo campo SIM_CLKDIV1_OUTDIV4 na inicialização padrão (Seção 12.2.12/página 210 em [3])? Qual é o valor configurado em rot6_aula [8]? Quais dos módulos, SysTick, PIT e RTC, têm as frequências dos seus contadores afetados por esta configuração?

1.d **Módulo PIT:** Registre as larguras dos pulsos dos sinais nos pinos PTE20 e PTE21.

1.d.1 Certifique os valores setados nos registradores de configuração do PIT na sua inicialização (aba Registers), colocando um *breakpoint* na primeira instrução após a função de configuração PIT_initTimer0.

1.d.2 Será que a largura dos pulsos registrados no pino PTE21 corresponde às configurações estabelecidas em SIM_CLKDIV1 e PIT_LDVAL0? Como essas configurações influenciam diretamente a frequência dos pulsos gerados? Alterar os valores em SIM_CLKDIV1_OUTDIV4 ou PIT_LDVAL0 terá impacto nas larguras dos pulsos dos sinais observados no analisador lógico?

1.d.3 Será que a largura dos pulsos registrados no pino PTE20 corresponde às configurações estabelecidas em SIM_CLKDIV1 e PIT_LDVAL0 e o valor atribuído a POSTSCALER no arquivo ISR.h? Alterar os valores em POSTSCALER terá impacto nas larguras dos pulsos de qual dos dois sinais, PTE20 e PTE21, observados no analisador lógico?

1.d.4 Setando o divisor SIM_CLKDIV1 em 1, ou seja SIM_CLKDIV1_OUTDIV4 setado em 0b001 e SIM_CLKDIV1_OUTDIV1 pré-fixado em 0b0000, qual valor deve ser configurado em PIT_LDVAL0 para gerar interrupções periódicas de 2,5s? Certifique, reexecutando o programa após alterações pertinentes e geração de um novo código executável.

1.e **Módulo RTC:** Ajuste a escala da saída do analisador de forma que possa ver um pulso completo em PTE22. Registre as larguras dos pulsos dos sinais nos pinos PTE22 e PTE23.

1.e.1 Certifique os valores setados nos registradores de configuração do RTC na sua inicialização (aba Registers), colocando um *breakpoint* na primeira instrução após a função de configuração RTClpo_init.

1.e.2 A largura dos pulsos no pino PTE23 está condizente com a programação? Justifique com base no seu conhecimento sobre o módulo RTC, lembrando que os valores computados pela Eq. (5) e acessados pela função RTClpo_getTime são truncados em segundos.

1.e.3 Se subistituirmos a frequência do sinal de relógio de 1kHz por 10kHz, quais seriam as larguras dos pulsos em PTE22 e PTE23 se mantivermos o mesmo programa? Justifique com base no seu conhecimento sobre o módulo RTC.

1.f **Incremento de Segundos por *Polling***: Diferentemente do sinal medido em PTE23, que é amostrado por *polling* (*software*), o sinal medido em PTE22 é gerado por interrupção (*hardware*). Procure compreender a diferença entre amostragem por *polling* e por interrupção comparando os códigos das instruções que lidam com os sinais em PTE22 e PTE23.

1.g **Conversão de Inteiros para *Strings***: Os horários no formato HH:MM:SS contém sempre dois dígitos mesmo que os valores sejam menores que 10. Procure compreender como é implementada a inserção de '0' nos dígitos mais significativos na função ConvertSectoDayString quando os valores numéricos são menores que 10.

1.h **Regiões Críticas**: Em rot6_aula [8], desabilitamos as interrupções pelo NVIC para proteger a região crítica

```
estado = ISR_LeEstado ();
ISR_leHorario (&dias, &horas_atual_local, &minutos_atual_local,
&segundos_atual_local);
```

Identifique nos códigos as instruções de desabilitação e reabilitação das interrupções.

1.i **Frequência em Atualizações do RTC:** Para simplificar processamentos da interface com usuários, os horários são armazenados e processados via vetor hor.

1.i.1 Em qual função é inicializado o conteúdo de `hor` quando se chaveia do estado `NORMAL` para outros estados?

1.i.2 Nma "sessão" de interação, os valores armazenados em `hor` e os valores em `RTC_TSR` e `RTC_TPR` são equivalentes?

1.i.3 Em qual função os valores em `hor` são trensferidos para `RTC_TSR` e `RTC_TPR`? Quais são as condições necessárias para alterar valores nesses 2 registradores?

1.i.4 Identifique a função que atualiza `RTC_TSR` e `RTC_TPR`.

**1.j Frequência em Atualizações do LCD:** Para evitar renderização repetida dos mesmos valores inúmeras vezes, adotou-se a estratégia de inserção de estados de transição e a estratégia de memória de valores anteriores. Identifique a implementação dessas 2 estratégias em `rot6_aula` [8].

1.k ***Timeout*:** Para implementar o *timeout* dos acionamentos do botão IRQA5 foi usado o `PIT`.

1.k.1 Identifique o ponto do fluxo em que a contagem por `PIT` começa e o ponto em que a contagem termina.

1.k.2 Qual função em `PORTA_IRQHandler` garante que toda contagem em `PIT` inicia a partir do "zero"?

2 **Praticar as práticas:** Desenvolva o projeto `cronometro` com as unidades de tempo, HH, MM e SS, ajustados por `NMI`, `IRQA5` e `IRQA12`, respectivamente. Recomenda-se os seguintes passos que procuram reusar os códigos do projeto `rot6_aula` [8] por ter um fluxo de controle similar ao do projeto `cronometro`. As diferenças são o estado adicional `INATIVO`, os dois botões adicionais, a configuração de *timeout*, e os processamentos do estado `NORMAL` e do estado `CRONOMETRO`.

2.a Crie um novo projeto (Seção 2.1/página 4 em [29]).

2.b Sobreescreva o arquivo `main.c` do projeto `rot6_aula` sobre `main.c` do novo projeto e faça os **testes funcionais** para certificar o porte (Seção 2.2.3/página 14 em [29]).

2.c Remova as funções de inicialização dos pinos de PTE e as instruções de espelhamento dos eventos nos pinos PTE. Faça os **testes funcionais** para certificar a remoção.

2.d Remova a função `RTC_ativaSegundoIRQ` que ativa IRQ21.

2.e **Três novos estados**: `INATIVO`, o estado inicial do dispositivo em que o visor do LCD é resetado com 2 mensagens ("00:00:00" na primeira linha e "APERTE BOTOEIRA" na segunda linha), e `RTC` é desabilitado (RTC_SR_TCE em '0'), `PRE_INATIVO`, o estado de transição do final de uma cronometragem para `INATIVO` em que são renderizadas as 2 mensagens e é **desabilitado `RTC`**, e `PRE_CRONOMETRO`, o estado de transição para `CRONOMETRO` em que muda o visor do LCD para a segunda linha é apagada. Os estados `PRE_INATIVO` e `PRE_CRONOMETRO` são tratados na função `main`.

2.f **Botões adicionais (bordas de descida)**:

2.f.1 Implemente a função `void GPIO_initSwitches (uint8_t NMI_IRQC, uint8_t IRQA5_IRQC, uint8_t IRQA12_IRQC, uint8_t prioridade)` em `GPIO_switches.c` para inicializar as 3 botões com interrupção (borda de descida) habilitada. Substitua `GPIO_initSwitchIRQA5` por essa nova função. Gere um executável e faça **testes de unidade** do atendimento de solicitações das 3 botões, pressionando aleatoriamente os 3 botões.

2.f.2 Adicione o tratamento dos eventos de `NMI` (`PTA4`) no processamento de horas (HH) em `PORTA_IRQHandler` (`ISR.c`) e `main` (`main.c`) de forma similar ao botão `IRQA5`. Insira nas funções `GPIO_desativaSwitches*` e `GPIO_reativaSwitches*` o tratamento de `PTA4`. Faça **testes de unidade** do processamento dos eventos de `PTA4`.

2.f.3 Adicione o tratamento dos eventos de `IRQA12` (`PTA12`) no processamento de segundos (SS) em `PORTA_IRQHandler` (`ISR.c`) e `main` (`main.c`) de forma similar ao botão `IRQA5`. Insira nas funções `GPIO_desativaSwitches*` e

`GPIO_reativaSwitches*` o tratamento de `PTA12`. Faça **testes de unidade** do processamento dos eventos de `PTA12`.

2.g **Processamento dos eventos de botões:** Diferentmente do `rot6_aula` [8], os ajustes não são relativos ao horário corrente. Eles são em relação ao valor inicial 00:00:00. Além disso, no `cronometro` a `contagem` é limitada a um valor de cronometragem especificado pelo usuário.

2.g.1 Habilite e resete o contador do `PIT` usando a função `PIT_resetaCVAL0` em `PORTA_IRQHandler` para o controle de *timeout*.

2.g.2 Substituir em `PORTA_IRQHandler` a chamada

```
ISR_carregaHorario();
```

por

```
hor[3] = hor[0] = hor[1] = hor[2] = 0;
```

2.g.3 Ao pressionar um botão no estado `INATIVO`, podemos considerar esse pressionamento como um incremento na unidade de tempo correspondente ao botão. Para isso, devemos inserir na função `PORTA_IRQHandler` uma linha no processamento do evento do botão `hor[x]++`, onde x é a unidade de tempo correspondente, como mostra o seguinte bloco de códigos:

```
case INATIVO:

    if (PORTA_ISFR & PORT_ISFR_ISF(GPIO_PIN(5))) {
         hor[1]++;
         estado = INCREMENTE_MINUTO;
    }
    break;
```

2.g.4 A partir de um estado de regime, `INCREMENTE_HORA_ESPERA`, `INCREMENTE_MINUTO_ESPERA` ou `INCREMENTE_SEGUNDO_ESPERA`, o sistema pode passar para qualquer um dos 3 estados de transição, `INCREMENTE_HORA`, `INCREMENTE_MINUTO` ou `INCREMENTE_SEGUNDO`, dependendo do botão pressionado. Codifique esta ideia no tratamento dos estados de regime, `INATIVO`, `INCREMENTE_HORA_ESPERA`, `INCREMENTE_MINUTO_ESPERA` ou `INCREMENTE_SEGUNDO_ESPERA`, dentro da função `PORTA_IRQHandler` em `ISR.c`. É importante garantir que os incrementos em horas sejam feitos no módulo de 24 horas, enquanto os incrementos em minutos e segundos sejam feitos no módulo de 60.

2.g.5 Os estados de transição, `INCREMENTE_HORA`, `INCREMENTE_MINUTO` ou `INCREMENTE_SEGUNDO`, são tratados no laço do fluxo principal da função `main`. Para recuperar o valor atualizado das unidades de tempo incrementadas pelo usuário, use a funçao `ISR_leHorario`.

2.g.6 **Não se esqueça de baixar *flags* dos eventos que não satisfazem as condições estabelecidas, para evitar novas interrupções.**

2.h ***Timeout***: Configure `PIT` com os valores calculados em 1.d.4. Ajuste a rotina de serviço `PIT_IRQHandler` para desabilitar `PIT` e mudar o sistema para o estado `PRE_CRONOMETRO` a ser processado no laço do fluxo principal da rotina `main`. No estado `PRE_CRONOMETRO`, o `RTC` é **habilitado**, os conteúdos de `RTC_TSR` e `RTC_TPR` são resetados, e a `contagem` em segundos é computada a partir das entradas do usuário, antes de direcionar o sistema para o estado `CRONOMETRO`.

2.i **Estado Normal → Estado CRONOMETRO:** Além de detectar os incrementos `atual`, em segundos, por *polling*, precisa-se ajustar esses valores para serem renderizados no visor do LCD em contagem regressiva, ou seja deve-se mostrar no visor do LCD o valor `contagem - atual`. Quando `contagem - atual == 0`, direcione o sistema para o estado `PRE_INATIVO` que **desabilita `RTC`** e atualiza o visor do LCD.

2.j Habilite *Print Size* para uma simples análise do tamanho de memória ocupado. Gere um executável e faça **testes funcionais** do projeto ajustando os horários aleatoriamente para ver se a resposta está condizente com a especificação.

2.k Revise a documentção das funções nos arquivos-cabeçalho. Gere uma documentação do projeto com Doxygen [9].

## RELATÓRIO

O relatório deve ser devidamente identificado, contendo a identificação do instituto e da disciplina, o experimento realizado, o nome e RA do aluno. Para este experimento, registre os testes de unidade realizados no item 2, destacando os testes malsucedidos e as suas ações corretivas num arquivo em pdf. Exporte o projeto `pomadoro` **devidamente documentado** num arquivo comprimido no IDE CodeWarrior. Suba os dois arquivos no sistema *Moodle*. **Não se esqueça de limpar o projeto (*Clean* …) e apagar as pastas html e latex geradas pelo Doxygen antes da exportação**.

## REFERÊNCIAS


[1] Freescale. Kinetis L Peripoeral Module Quick Reference
https://www.dca.fee.unicamp.br/cursos/EA871/references/ARM/KLQRUG.pdf
[2] ARMv6-M Architecture Reference manual
https://www.dca.fee.unicamp.br/cursos/EA871/references/ARM/ARMv6-M.pdf
[3] *KL25 Sub-Family Reference Manual – Freescale Semiconductors (doc. Number KL25P80M48SF0RM)*, Setembro 2012.
https://www.dca.fee.unicamp.br/cursos/EA871/references/ARM/KL25P80M48SF0RM.pdf
[4] Convert seconds to HH:MM:SS
https://www.csestack.org/online-tool-to-convert-seconds-to-hours-minutes-hhmmss/
[5] Roteiro 5
http://www.dca.fee.unicamp.br/cursos/EA871/1s2024/roteiros/roteiro5.pdf
[6] Tutorialspoint. Convert an int to ASCII character in C/C++.
https://www.tutorialspoint.com/convert-an-int-to-ascii-character-in-c-cplusplus
[7] Split a number into digits
https://www.log2base2.com/c-examples/loop/split-a-number-into-digits-in-c.html
[8] rot6_aula
http://www.dca.fee.unicamp.br/cursos/EA871/1s2024/codes/rot6_aula.zip
[9] Documentação com Doxygen
https://www.doxygen.nl/manual/docblocks.html
[10] Keil Forum. Conversion of integer to ASCII for display.
https://community.arm.com/support-forums/f/keil-forum/17118/conversion-of-integer-to-ascii-for-display
[11] Freescale. Kinetis L Peripoeral Module Quick Reference
https://www.dca.fee.unicamp.br/cursos/EA871/references/ARM/KLQRUG.pdf
[12] *Datasheet* do *display* LCD
https://www.dca.fee.unicamp.br/cursos/EA871/references/datasheet/AC162A.pdf
[13] Erich Styger. EnterCritical() and ExitCritical(): Why Things are Failing Badly.
https://mcuoneclipse.com/2014/01/26/entercritical-and-exitcritical-why-things-are-failing-badly/
[14] Diagrams.net
https://www.diagrams.net/
[15] Roteiro 3
http://www.dca.fee.unicamp.br/cursos/EA871/1s2024/roteiros/roteiro3.pdf
[16] Esquemático do *shield* FEEC871
https://www.dca.fee.unicamp.br/cursos/EA871/references/complementos_ea871/Esquematico_EA871-Rev3.pdf
[17] Engenharia Reversa: como usá-la no desenvolvimento de *software*?

https://www.ivoryit.com.br/2022/05/06/engenharia-reversa-como-pode-ser-usada-no-desenvolvimento-de-software/
[18] Wu, S.T. Ambiente de Desenvolvimento de Software
https://www.dca.fee.unicamp.br/cursos/EA871/references/apostila_C/AmbienteDesenvolvimentoSoftware_V1.pdf

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