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] apresentamos os conceitos básicos relacionados com temporizadores e aplicamos o temporizador integrado ao núcleo/processador do KL25Z, `SysTick` (Seção B3.3/página 275 em [2]), no controle de diferentes intervalos de tempo. Neste experimento, vamos apresentar dois módulos-periférico de temporizadores, `PIT` *(Periodic Interrupt Timer*, Capítulo 32/página 573 em [3]), e `RTC` (*Real Time Clock*, Capítulo 34/página 597 em [3]), que, como `SysTick`, só geram eventos capazes de interromper fluxos de execução. Porém, diferentes do `SysTick` que geram sinais síncronos, os eventos gerados por `PIT` e `RTC` são assíncronos em relação à operação do processador. A seção de descrição funcional (*Functional Description*) em cada capítulo de [3] apresenta os detalhes técnicos do respectivo módulo. Mostramos, através da implementação de um relógio digital, a configuração e o uso das funções configuradas de `RTC` e `PIT` na execução de uma tarefa específica.

No projeto `relogio_digital` o horário é mostrado no meio da primeira linha do visor do LCD no formato padrão HH:MM:SS (24 horas) (**estado normal**). O horário é ajustado a cada segundo com base nas contagens feitas pelo RTC. É possível ajustar o horário por meio de três botoeiras cujas **bordas de subida** são tratadas como potenciais eventos de interrupção. Sempre se passa **do estado NORMAL** para o **estado incremente_horas**, **incremente_minutos** e **incremente_segundos** quando se aciona, respectivamente, a botoeira `NMI`, `IRQA5` e `IRQA12`. **Se dentro de um *timeout*=2.75s nenhuma chave for acionada, o relógio volta de qualquer um dos estados de ajuste ao estado NORMAL. Somente neste instante, as contagens nos registradores `RTC_TPR` e `RTC_TSR` são, de fato, atualizadas**. Automaticamente, o RTC passa a fazer a contagem a partir dos novos valores setados e o LCD passa a mostrar o novo horário. Um diagrama de máquina de estados do relógio digital especificado é mostrado na Figura 1. Note que para cada estado de ajuste, há um estado **espera**

correspondente. Vamos ver que essa foi uma decisão de projeto para evitar processamentos desnecessários do LCD.

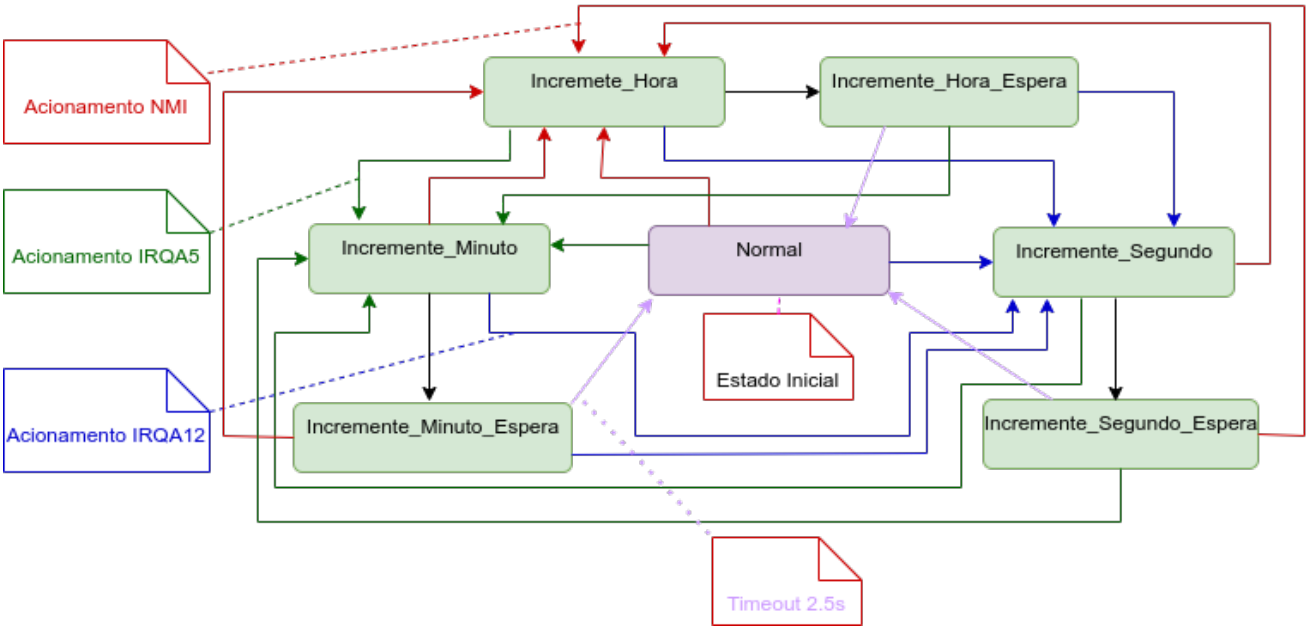

Figura 1: Máquina de estados de `relogio_digital` (editado em [14]).

Módulo MCG: MCGOUTCLK

O **sinal `MCGOUTCLK`** é gerado pelo módulo *Multipurpose Clock Generator* (MCG) e usado para sincronizar as operações dos módulos integrados num microcotrolador. Em MKL25Z, a sua fonte é selecionável via o campo de configuração `MCG_C1_CLKS` dentre as 3 fontes (Seção 24.3.1/página 372 em [3]): sinal de relógio de referência interna, sinal de relógio de referência externa e sinal gerado por um laço de sincronismo, PLL ou FLL.

Os geradores de sinais por um laço de sincronismo de fase (*Phase-Locked Loop*, PLL) ou de frequência (*Frequency-Locked Loop*, FLL) são circuitos comumente aplicados em sincronização de relógios e geração de sinais. Eles geram, respectivamente, sinais sincronizados com uma fase ou uma frequência de entrada. Em MKL25Z, um sinal por FLL é gerado a partir de um sinal de referência interna (`MCGIRCLK`) ou externa (`OSCERCLK`) do módulo MCG sincronizado com uma referência de entrada de frequência. O **FLL** compara a frequência do sinal de referência com a referência de entrada e ajusta a frequência do sinal de referência para corresponder ao sinal de referência de entrada. Um sinal por PLL é controlado por um sinal de referência externa (`OSCERCLK`). O **PLL** compara a fase de entrada com a fase de saída gerada pelo circuito e ajusta a fase de saída de acordo para mantê-la sincronizada com a de entrada. No IDE *CodeWarrior*, o sinal `MCGOUTCLK` é gerado por FLL (`MCG_C1_CLKS` resetado em 0b00 (Seção 24.3.5/página 376 em [3]), `MCG_C5_PLLCKEN0` e `MCG_C5_PLLSTEN0` em '0' (Seção 24.3.1/página 372 em [3])) na inicialização de MKL25Z. Nos nossos experimentos, usamos essa configuração padrão.

Distribuição de Sinais de Relógio

Os contadores integrados em `PIT` e `RTC` são pulsados por sinais de relógio de frequências distintas em função dos seus propósitos. O primeiro circuito é para gerar eventos de interrupções periódicas em frequências configuráveis e o segundo para gerar acuradamente eventos de interrupções periódicas em 1Hz. Para atender diferentes velocidades de operação dos módulos-periférico integrados, KL25Z dispõe do módulo *Multipurpose Clock Generator* (`MCG`) dedicado à geração das referências aos sinais de relógio dos módulos (Figura 24-1/página 370 em [3]). Em conjunto com o módulo *System Integration Module* (`SIM`), são derivados os sinais de relógio de barramentos (*bus interface clock*) que conectam os módulos (*platform clock*, *system clock*, *bus clock* e *flash clock*) e os sinais de relógio (*internal clock*) requeridos por alguns módulos para a sua operação interna (`core clock`, `LPO`,

MCGOUTCLK, MCGPLLCLK, MCGFLLCLK, MCGIRCLK, OSCERCLK, ERCLK32K) (Tabela 5-2/página 121 em [3]).

O *bus interface clock* é utilizado para sincronizar a comunicação entre componentes através dos barramentos, os *internal clocks* são usados para sincronizar as operações internas de cada componente do sistema, como os contadores dos temporizadores. PIT e RTC compartilham o mesmo barramento cujo sinal de relógio (*bus interface clock*) é *bus clock*. Eles diferem nos sinais de relógio internos que alimentam seus contadores: PIT usa o mesmo *bus clock*, enquanto RTC tem três alternativas, selecionáveis por *software*, para pulsar os seus contadores internos (Seção 5.7.3/página 123 em [3]).

O sinal de relógio de barramento (*bus clock*) resulta da divisão do sinal MCGOUTCLK (20.971.520Hz) por dois circuitos divisores no módulo SIM, OUTDIV1 e OUTDIV4 (Figura 5-1/página 116 em [3]). Os dois divisores são configuráveis 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 condicionada à configuração setada no registrador de configuração FTF_FOPT[LPBOOT] (Seção 27.33.4/página 429 em [3]). Por configuração padrão do IDE, OUTDIV1 = 1.

As 3 alternativas de fontes de sinais para o sinal de relógio ERCLK32K que pulsa os contadores do RTC são (Seção 5.7.3/página 123 em [3]): 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. Elas 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 o uso desse 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*, configurável para uma unidade de 64 *bits* por meio do registrador de controle PIT_TCTRLn. Através do registrador de controle PIT_MCR, configura-se a sua habilitação e a sua operação no modo *Debug*. Como SysTick, a *s*ua contagem é decrescente. Os temporizadores carregam no contador PIT_CVALn a contagem máxima configurada no registrador de dados PIT_LDVALn. Em particular, 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. O valor é decrementado até atingir 0, quando 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', é ativado o sinal de requisição de interrupção IRQ 22/vetor 38 (Tabela 3-7/página 52 em [3]). 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 em '0', e reabilitá-lo, setando PIT_TCTRL_TEN em '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].

Como nos 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 usados como gatilhos dos módulos 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 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. As habilitações dos gatilhos são controláveis, por *software,* através de registradores de configuração localizados nos módulos interconectados, como no módulo TPM (Tabela 3-38/página 86 em [3]), ou no módulo SIM em interconexões com o módulo ADC (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 segundos, sob a condição de que a frequência da fonte de *clock* seja 32,768kHz. Figura 2 apresenta um diagrama de blocos do módulo RTC.

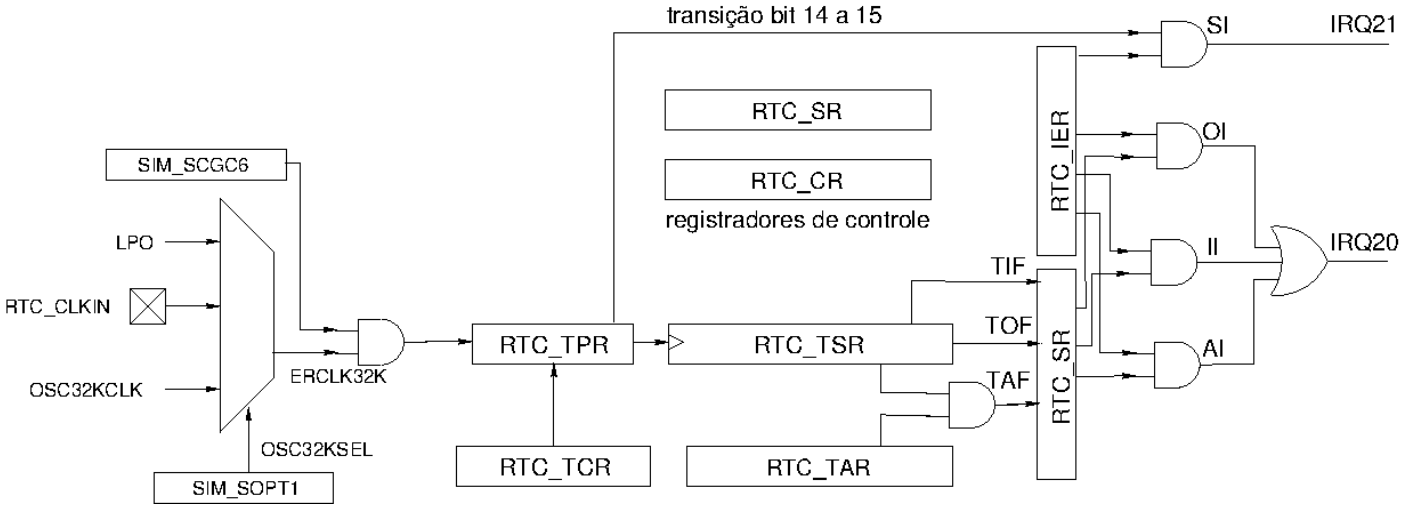

Figura 2: Diagrama de blocos do módulo RTC.

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

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

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 aumentar a acurácia dos tempos medidos, há um circuito de compensação integrado no RTC, capaz de compensar desvios em relação aos valores nominais no intervalo entre 0,12ppm (*parts per million*) a 3906ppm. O valor de compensação deve ser configurado, por *software*, via o registrador de configuração RTC_TCR. RTC só auxilia na definição do valor de compensação, gerando um sinal de 1Hz, acessível pelo pino 3 da porta C (RTC_CLKOUT) (Seção 10.3.1/página 163 em [3]).

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:

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

Note-se que, embora a segunda e a terceira expressões nas Eqs. (4) e (5) sejam equivalentes matematicamente, elas geram resultados distintos em operações inteiras dos processadores. A segunda expressão é a soma de dois termos truncados enquanto a terceira expressão é o truncamento da soma. O erro acumulado na segunda expressão é maior do que a da terceira expressão.

Como nos 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, em processamento paralelo do núcleo, inicia a contagem, mantendo os "tempos" em RTC_TSR e RTC_TPR atualizados.

Há no RTC um circuito de alarme que seta o *bit* RTC_SR_TAF em '1' quando a contagem em RTC_TSR se igualar com 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). É o único módulo que tem mais de uma IRQ associada, sendo uma dedicada para incrementos em segundos, conforme mostra a figura 1: IRQs 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 segundo 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 no projeto `rot6_aula` [8].

Conversão entre Segundos e HH:MM:SS

Representar os horários em segundos simplifica o circuito do RTC. Porém, o formato com que os potenciais usuários são familiarizados, seja no formato de 24 horas seja de 12 horas, é (DIA:)HH:MM:SS [4]. Nos ajustes dos horários de um relógio, pensa-se sempre em ajustar separadamente as unidades de tempo num dia (HH horas, MM minutos e SS segundos), e não em segundos como no módulo `RTC`. Precisamos atentar a essas diferenças em representações e desenvolver interfaces entre elas para que o modo de operação da máquina fique mais transparente possível para usuários. Isso aumenta a inclinação da curva de aprendizagem e a aceitabilidade do produto.

Converter o formato em segundos para o formato HH:MM:SS torna mais fácil implementar uma interface amigável e intuitiva para usuários, permitindo que estes façam ajustes por unidade de tempo. Isso pode reduzir a quantidade de instruções executadas em cada interação. Por outro lado, precisamos assegurar que os horários ajustados sejam convertidos para o formato entendível pelo módulo `RTC` e usar estes valores para atualizar os registradores `RTC_TPR` e `RTC_TSR` após os ajustes feitos. Aplicando a aritmética modular, a conversão entre `segundos` e os no formato (DIA:)HH:MM:SS é 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.

Conversão de Inteiros para *Strings* na Base Decimal

Outro ponto a destacar é a conversão de um valor inteiro das unidades de tempo, horas, minutos e segundos, numa *string* de dígitos em ASCII para que o valor possa ser mostrado no visor do LCD [12] na forma como estamos familiarizados. É importante sempre manter em mente que **um LCD só renderiza os *bitmaps* dos endereços da Tabela de Fontes (CGROM) que estão escritos na**

memória DDRAM. Cabe ao projetista decidir o que escrever em DDRAM para que o resultado renderizado no visor seja condizente com a expectativa. Faz parte da exibição de valores inteiros no LCD a conversão de inteiros para *strings* de caracteres. Um algoritmo popular de conversão consiste de 2 partes: (1) extrair a sequência de dígitos do valor de interesse pelas divisões sucessivas por 10 [7], e (2) somar o valor numérico de cada algarismo com o valor numérico de '0'==0x31==48 para obter o código ASCII correspondente [6]. Em [10] é apresentada uma implementação do algoritmo em C em que deve ser especificada a priori a quantidade de dígitos que um valor inteiro contém. A implementação `UlToStr` é incluída no arquivo `util.c` em `rot6_aula` [8]. Porém, pelo fato das unidades de um horário do dia serem sempre representadas por 2 dígitos, mesmo quando o valor é menor que 10, foi usada uma implementação "personalizada" para as unidades de tempo em `ConvertSectoDayString` (`util.c`).

Regiões Críticas

Os ajustes nos horários de um relógio digital são feitos tipicamente através dos pressionamentos de botoeiras, como no projeto `rot6_aula` [8]. Esses eventos assíncronos podem interromper, na resolução de instruções de máquina (*assembler*), um fluxo de controle em qualquer ponto, como entre as instruções que atualizam o visor do LCD. Neste caso, o conteúdo do vetor `hor` pode ser modificado no tratamento de uma interrupção e fazer com que, ao retornar para as instruções de renderização do LCD, o restante dos valores mostrados não sejam mais os que eram para ser exibidos. Isso acontece porque o bloco de instruções no tratamento de interrupções e o bloco as instruções do fluxo de controle principal fazem acessos ao mesmo vetor `hor`. Para que as ações de um bloco não interfiram de forma indesejada nos resultados gerados pelo outro, uma solução é assegurar que os seus **processamentos** sejam **mutuamente exclusivos**. Denominamos de **regiões críticas** esses blocos de instruções. Como protegê-las de interrupções indevidas é um dos desafios para programação de microcontroladores [13].

Uma solução mais simples, muito encontrada na programação de microcontroladores, é manter as interrupções indesejadas desabilitadas durante a execução de uma região crítica. Os mecanismos de interrupção implementados na maioria dos módulos do KL25Z envolve dois circuitos principais: o circuito que configura a solicitação de interrupção (também chamado de circuito de geração de interrupção) e o circuito que configura o atendimento da interrupção (também chamado de circuito de tratamento de interrupção). Portanto, essa desabilitação pode ser feita em dois pontos: na geração dos eventos de interrupção pelos módulos e no atendimento de eventos de interrupção pelo controlador `NVIC` (Seção B3.4/página 281 em [2]). No caso de eventos gerados pelos sinais digitais de propósito geral, como os das bo~~b~~toeiras, 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 diminuir 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 `e` postergamos as atualizações do conteúdo

dos registradores do RTC, em segundos, para após a confirmação dos valores entrados em hor. No entanto, por amigabilidade, renderizamos os valores intermediários entrados para que usuários saibam o que o sistema entendeu das suas ações.

Timeout

Timeout é um período de tempo permitido em um sistema antes que um evento ocorra, a menos que outro evento especificado ocorra primeiro; em ambos os casos, o período termina quando um dos eventos ocorre. Em rot6_aula [8] a confirmação dos valores entrados em hor é implementada com uso de PIT. Ao invés de mais um movimento mecânico de digitação de uma "tecla de confirmação", adotamos a estratégia de confirmação por falta de acionamentos. Se transcorrer um intervalo de tempo maior do que o tempo permitido para uma nova digitação, o sistema retorna para o estado de relógio atualizando o valor do horário em cada segundo. É reiniciada a contagem de tempo no PIT cada vez que haja uma interação com a botoeira IRQA5.

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 retornaado.

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 de descobrir o princípio de funcionamento de um sistema através da análise da sua estrutura e do seu comportamento. Ela pode ser aplicada na "dissecação" de códigos abertos para descobrirmos os algoritmos aplicados e os "pulos de gato", como também na descoberta do funcionamento de códigos fechados [17]. Nesta disciplina vamos praticar a engenharia reversa sobre projetos que temos códigos-fonte usando as ferramentas de depuração disponíveis no IDE CodeWarrior (Seção 2.5/página 25 em [18]). Na maioria das vezes, o procedimento consiste em rastrear o fluxo de controle executando as instruções, passo a passo, a partir de uma linha de comando ou de uma chamada de função de interesse. Através da análise da forma como os dados são transformados e as bifurcações dos fluxos de controle, pode-se tentar reconstruir o algoritmo original. O ponto inicial de um rastreamento é tipicamente definido como um ponto de parada (*breakpoint*).

EXPERIMENTO

Neste experimento vamos implementar o diagrama de máquina de estados do relógio digital mostrado na Figura 1. O módulo principal na implementação do relógio digital é `RTC`, tendo como fonte de clock `LPO` para seus contadores internos. Para detectar o *timeout* dos acionamentos das botoeiras foi utilizado o temporizador `PIT`. Assim que um acionamento válido de uma botoeira for detectado, é habilitado o temporizador que reinicia a contagem. Como o período máximo de `PIT` 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 2,75s.. Adicionalmente, são necessários os módulos `GPIOC/PORTC` para interfacear com o LCD e `GPIOA/PORTA/NVIC` para interfacear com os eventos de interrupção gerados pelas botoeiras. Figura 3 mostra um diagrama de componentes utilizados na implementação do relógio digital proposto.

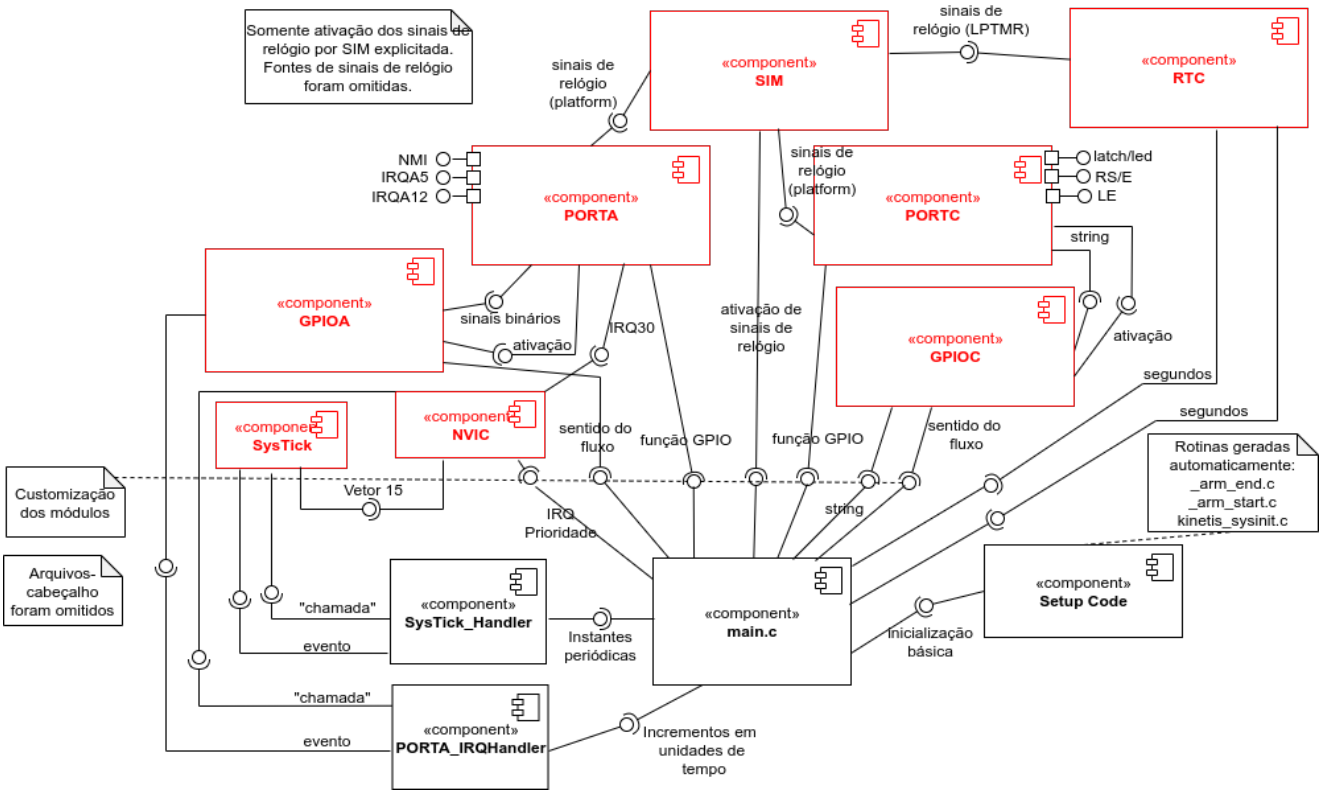

Figura 3: Diagrama de componentes do `relogio_digital` (editado em [14]).

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

1 Carregue o projeto `rot6_aula` [8] no IDE CodeWarrior, substitua a função `espera_5us` 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.

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

1.2 **Distribuição de Sinais de Relógio:** 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`? Quais dos módulos, ~~GPIOx,~~ SysTick, PIT e RTC, ~~são~~ têm as frequências dos seus contadores afetados por esta configuração?

1.3 **Módulo PIT:** Registre as larguras dos pulsos dos sinais nos pinos `PTE20` e `PTE21` com *printscreen*.

1.3.1 A largura dos pulsos registrados no pino `PTE21` está condizente com as configurações feitas em `SIM_CLKDIV1` e `PIT_LDVAL0`? Se aumentarmos ou diminuirmos os valores no campo `SIM_CLKDIV1_OUTDIV4` ou em `PIT_LDVAL0`, o que acontecerá com as larguras dos pulsos dos sinais visualizados pelo analisador lógico?

1.3.2 A largura dos pulsos registrados no pino `PTE20` está condizente com as configurações feitas em `SIM_CLKDIV1` e `PIT_LDVAL0` e `POSTSCALER`? O que acontecerá com as larguras dos pulsos dos sinais em `PTE20` e `PTE21`, se aumentarmos ou diminuirmos somente os valores de POSTSCALER?

1.3.3 Setando o divisor `SIM_CLKDIV1` em 1, ou seja `SIM_CLKDIV1_OUTDIV4` setado em 0b000 e `SIM_CLKDIV1_OUTDIV1` pré-fixado em `0b0000`, qual valor deve ser

configurado em `PIT_LDVAL0` para gerar interrupções periódicas de 2,75s? Certifique, reexecutando o programa após alterações pertinentes e geração no novo código executável.

1.4 **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` com *printscreen*.

1.4.1 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.

1.4.2 Se subistituirmos a frequência do sinal de relógio de 1kHz para 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`.

1.4.3 Lembrando que a resolução do alarme passa para 32,768s com LPO como a fonte de pulsos de relógio, propõe uma forma de implementar, por *software,* um alarme com resolução em segundos?

1.5 **Incremento de Segundos por *Polling***: Em qual função do arquivo RTC.c o sinal no pino `PTE22` é gerado? A largura dos pulsos está condizente com a programação? Se subistituirmos a frequência do sinal de relógio de 1kHz para 10kHz, quais adaptações deveremos fazer nas Eqs. (5) e (6)? Obs.: Diferente do sinal medido em PTE23 que é amostrado por *polling* (*software*), o sinal medido em PTE22 é gerado por interrupção (*hardware*).

1.6 **Conversão de Inteiros para *Strings***: Os horários no formato HH:MM:SS contém sempre dois dígitos mesmo que os valores sejam menores que 10. Descreva sucintamente como são inseridos os '0' na função `ConvertSectoDayString` quando os valores são menores que 10.

1.7 **Regiões Críticas**: Em `rot6_aula` 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);
```

Substitua as desabilitações por NVIC, `GPIO_desativaSwitchesNVICInterrupt` e `GPIO_reativaSwitchesNVICInterrupt`, pelas desabilitações por PORTx, `GPIO_desativaSwitchesIRQA5Interrupt` e `GPIO_reativaSwitchesIRQA5Interrupt`. Reexecute o novo código executável. Registre o comportamento do sistema observado. Se incluirmos as 2 outras botoeiras, NMI e IRQA12, no processamento, haverá diferença na percepção da fluidez das respostas do sistema?

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

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

1.8.2 Durante uma "sessão" de interações, os valores armazenados em `hor` e os valores em `RTC_TSR` e `RTC_TPR` são equivalentes?

1.8.3 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?

1.8.4 Identifique a função que atualiza `RTC_TSR` e `RTC_TPR`.

1.9 ***Timeout*:** Para implementar o *timeout* dos acionamentos da botoeira IRQA5 foi usado o `PIT`.

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

1.9.2 O que garante que toda contagem inicia a partir do "zero"?

2 **Engenharia Reversa:** Execute o projeto `rot6_aula` no modo *Debug* (Seção 2.5/página 25 em [18]).

2.1 **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` com os valores setados em `MCG_C1` e `MCG_C5`.

2.2 Certifique os valores setados nos registradores de configuração do `PIT` e `RTC` na inicialização desses dois módulos (aba `Registers`), colocando um *breakpoint* na primeira instrução após as respectivas funções de configuração `PIT_initTimer0` e `RTClpo_init`.

2.3 Descreva sucintamente o fluxo de controle programado para gerar os sinais nos pinos `PTE20`, `PTE21`, `PTE22` e `PTE23`.

3 Desenvolva o projeto `relogio_digital` 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` por ter um fluxo de controle similar ao do projeto `relogio_digital`. As diferenças são as duas botoeiras adicionais e a configuraçãoo de *timeout*.

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

3.2 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]).

3.3 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.

3.4 Remova a função `RTC_ativaSegundoIRQ` que ativa IRQ21.

3.5 **Botoeiras adicionais (bordas de subida)**:

3.5.1 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 botoeiras com interrupção habilitada. Substitua `GPIO_initSwitchIRQA5` por essa nova função. Gere um executável e faça **testes de unidade** do atendimento de solicitações das 3 botoeiras, pressionando aleatoriamente as 3 botoeiras.

3.5.2 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 à botoeira `IRQA5`. Insira nas funções `GPIO_desativaSwitches*` e `GPIO_reativaSwitches*` o tratamento de PTA4. Faça **testes de unidade** do processamento dos eventos de PTA4,.

3.5.3 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 à botoeira `IRQA5`. Insira nas funções `GPIO_desativaSwitches*` e `GPIO_reativaSwitches*` o tratamento de PTA12. Faça **testes de unidade** do processamento dos eventos de PTA12.

3.5.4 Com as 3 botoeiras integradas, faça **testes comparativos de unidade** da fluidez das respostas do sistema e implemente a melhor alternativa para desabilitar as interrupções antes de entrar na região crítica com base na resposta ao item 1.7. Se você não perceber diferenças, escolha uma das duas alternativas.

3.6 ***Timeout***: Configure `PIT` com os valores calculados em 1.3.3. Ajuste a rotina de serviço `PIT_IRQHandler` para processamento sem `POSTSCALER`.

3.7 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.

3.8 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, responda, num arquivo em pdf,

as perguntas nos itens 1 e 2. Exporte o projeto `relogio_digital` **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**.

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/EA871/1s2023/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/EA871/1s2023/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/EA871/1s2023/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 Janeiro de 2023
Revisado em Fevereiro e Agosto de 2022

Maio e Julho de 2021
Novembro de 2020