EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 6 – Exceções e Interrupções

Profa. Wu Shin-Ting

OBJETIVO: Apresentação do mecanismo de interrupção.

ASSUNTOS: Configuração e programação do MKL25Z128 para processamento de exceções e interrupções.

O que você deve ser capaz ao final deste experimento?

Entender o procedimento de tratamento de exceções e de interrupções do MKL25Z128.

Programar temporizador *SysTick* para processamento de exceções.

Programar temporizador *PIT* para processamento de interrupções.

Programar temporizador *RTC* para implementação de um relógio digital.

Programar os pinos das portas para processamento de interrupções externas via controlador NVIC (*Nested Vectored Interrupt Controller*).

Usar diagrama de transição de estados para descrever a sequência de estados pelos quais um programa pode passar.

INTRODUÇÃO

Nos experimentos 4 e 5, para saber se as botoeiras PTA4, PTA5 ou PTA12 foram pressionadas, tivemos que testar periodicamente os seus estados. Quando o nível lógico nos respectivos pinos for 0, consideramos que se fechou o circuito [8]. Essa prática de espera por um evento acontecer, ocupando o processador com a leitura de um registrador de estado, é conhecida por ***polling***.

Na maioria dos sistemas digitais, espera-se que o núcleo de processamento seja capaz de responder apropriadamente aos diferentes estados de um dispositivo do mundo exterior (teclado, sensores, contadores de tempo, etc.). No entanto, para monitorar os estados de um dispositivo, a técnica *polling* tem várias desvantagens, sendo as principais o desperdício de tempo de processamento na varredura das entradas e o tempo de resposta a uma entrada crítica. Porisso, a estratégia por **interrupção** é a mais recomendada para um microcontrolador responder a um evento externo. E, o nosso microcontrolador dispõe de um *hardware* específico, o NVIC (*Nested Vectored Interrupt Controller*), para monitorar e processar as entradas assíncronas externas **por módulo**, desviando a execução do programa para uma função chamada *handler* ou **rotina de serviço** (ISR - *Interrupt Service Routine*) no processador (Cap.3 de [1]). Quando se conclui a execução da sequência de instruções da rotina de serviço, o processador retorna ao fluxo original. Operando de forma cooperativa com o núcleo de processamento, o NVIC proporciona uma forma eficiente com uma interface de programação simples para tratar as múltiplas e aninhadas (*nested*) solicitações de interrupção. Figura 3-2 em [2] mostra a relação do NVIC com outros módulos do nosso microcontrolador.

No momento de criação de um novo projeto, o ambiente CodeWarrior gera automaticamente os arquivos `Project_Settings/Startup_Code/kinetis_sysinit.c` e `Project_Settings/Linker_Files/MKL25Z128_flash.ld`. Estes contém, respectivamente, o conteúdo do segmento `.vectortable` (tabela de vetores) e a posição da memória onde a tabela de vetores será relocada. Esta relocação ocorre automaticamente antes da

execução do nosso programa. O conteúdo da tabela de vetores é, de fato, os ponteiros/endereços a todas as rotinas de serviço correspondentes às exceções processáveis pelo nosso microcontrolador. O ponteiro à rotina de serviço da exceção ativa é carregada automaticamente no contador de programa (PC) para que a rotina seja executada. Quando há mais de uma solicitação de interrupção, o NVIC arbitra automaticamente a exceção de maior prioridade de atendimento com base no nível de prioridade das solicitações (Seção B.1.5 em [3]).

Com tantas funções relativas à interrupção já implementadas no nosso microcontrolador, **o trabalho do programador se reduz a configurar os módulos do nosso MCU para que eles operem no modo de interrupção e a customizar a forma de tratamento de cada exceção através da reprogramação das rotinas de serviço**. E, quando se trata de um evento externo, é necessário configurar o NVIC para ele arbitrar a prioridade de atendimento de múltiplas e aninhadas interrupções externas.

EXPERIMENTO

1 *Vamos entender como se programa o nosso microcontrolador para processar interrupções?* Leia atentamente a apostila sobre o modelo de exceção do nosso microcontrolador [4].

2 *Vamos ver como os conceitos são traduzidos na prática através de um programa* [6] que utiliza uma biblioteca de rotinas? Este programa possui as mesmas funcionalidades do projeto do roteiro 5 [7], com a diferença de que as facilidades de interrupção do nosso microcontrolador são exploradas em [5] tanto (a) para obter medições mais acuradas de tempo, quanto (b) para capturar variações nos estados das botoeiras PTA4, PTA5 e PTA12 com maior precisão. Para isso, além de utilizarmos dois módulos de temporização do nosso micro-controlador, SysTick (*System Timer*) e PIT (*Periodic Interrupt Timer*), habilitamos a capacidade de gerar sinais de interrupção dos pinos PTA4, PTA5 e PTA12 (Seção 11.5.1 em [2]).
Em termos de códigos, adicionamos três novos conjuntos de arquivos na biblioteca, `nvic.*`, `pit.*` e `systick.*`, e atualizamos o arquivo `pushbutton.*`. Inserimos ainda um arquivo de rotinas de serviço `handler.*`. E, em termos de fluxo de controle, tivemos que reestruturar os três blocos de controle sequenciais PTA12, PTA5 e PTA4 em [7] a fim de que os eventos não sequenciais de interrupção possam ser tratados de maneira coordenada. O código implementado em [6] foi baseado num diagrama de transições entre 5 estados identificados no sistema: temporizador parado mantendo LCD com informação do estado do temporizador configurado (0), temporizador em configuração (1), cronômetro operando (2), estado do cronômetro é exibido no LCD (3), e temporizador operando (4). Variáveis de estado, `modo`, `flag_muda_cor`, `flag_muda_estado`, `flag_muda_tempo`, foram adicionadas para controlar o fluxo de execução.

Veja na rotina de serviço `PORTA_IRQHandler` no arquivo `handler.c` como o **diagrama de transições foi traduzido para linguagem C** (note que alguns detalhes de controle foram omitidos por clareza):

1. Se o evento de interrupção é gerada pelo pino 4 da `PORTA`, então
   1.1 Se a botoeira `PTA4` for solta (e estiver no estado 2), então muda para estado 3 exibindo o estado do cronômetro;
   1.2 Senão

1.2.1 Se estiver no estado 0 ou 3 então inicia a cronometragem com uso do `PIT` e muda para o estado 2;

1.2.2 Se estiver no estado 2 então finaliza a cronometragem e muda para o estado 3;

Abaixe a bandeira correspondente ao pino 4 da `PORTA`;

2. Se o evento de interrupção é gerada pelo pino 5 da `PORTA`, então

2.1 Se a botoeira `PTA5` for solta e estiver no estado 1, então muda para o estado 0, parando o `SYSTICK`;

2.2 Senão, e estiver no estado 0 ou 3 então inicia a configuração do temporizador com uso do `SYSTICK` e muda para estado 1;

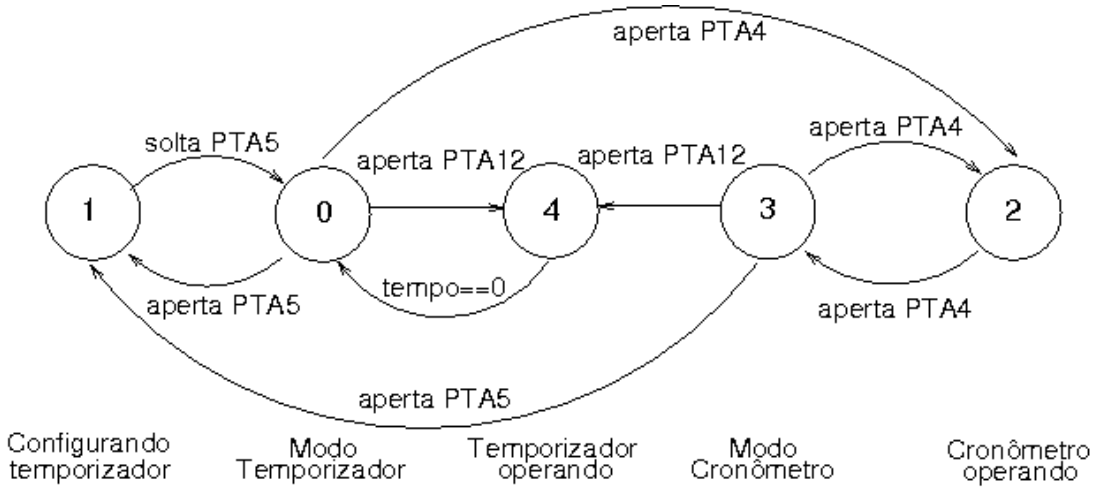
Abaixe a bandeira correspondente ao pino 5 da `PORTA`;

3. Se o evento de interrupção é gerada pelo pino 12 da `PORTA`, então

3.1 Se a botoeira `PTA12` for pressionada e estiver no estado 0 ou 3, então carrega o valor configurado do temporizador, inicia a contagem regressiva (`tempo`) com uso do `PIT`, e muda para o estado 4.

Abaixe a bandeira correspondente ao pino 12 da `PORTA`.

Observe que, no caso (3), a transição do estado 4 para 0 ocorre automaticamente na rotina de serviço `PIT_IRQHandler` quando a contagem regressiva atinge o valor 0 (`tempo == 0`).



Para que as mudanças dos estados das botoeiras PTA4, PTA5 e PTA12 disparem automaticamente a chamada da rotina `PORTA_IRQHandler`, precisamos **habilitar o mecanismo de interrupção dos pinos** em que elas estão ligadas. Estas botoeiras são ligadas nos pinos da porta A. Portanto, adicionamos a rotina `enablePushbuttonIRQA` no arquivo `pushbutton.c` para habilitar o mecanismo de interrupção dos pinos 4 (sensível à borda de descida), 5 (sensível às duas bordas) e 12 (sensível à borda de descida) da porta A (Seção 11.5.1 em [2]) . O módulo PORTA é um módulo externo ao núcleo (Seção 3.3.2.3 em [2]). Isso significa que suas interrupções serão somente atendidas se o NVIC estiver configurado para tal (Seção B3.4.2 em [3]), como mostra a rotina `enableNVICPTA` em `nvic.c` que setou o nível de prioridade da PORTA em 3. Você deve ter notado que a rotina trata as três interrupções, PTA4, PTA5 e PTA12, pois, sob o ponto de vista do `NVIC`, elas são agrupadas num mesmo vetor de interrupção IRQ=30. Como queremos processamento diferenciado para cada evento, foi elaborada uma **estrutura de fluxo de controle** de condições. É importante observar que no nosso microcontrolador a *flag* de interrupção do

módulo PORTA não é limpa automaticamente após o seu atendimento. É preciso limpá-la expicitamente escrevendo 1 (*write-1-to-clear*) no campo correspondente (Seção 11.5.4 em [2]).

Um bom observador deve perceber que a execução correta do procedimento implementado na rotina `PORTA_IRQHandler` **depende do funcionamento dos dois módulos de temporização, `SysTick` e `PIT`**. Estes módulos integrados no nosso micro-controlador são de fato contadores. Quando habilitamos a sua capacidade de gerar sinais de interrupção (Seção B3.3.3 em [3] e Seção 32.3.6 em [2]), eles geram tais sinais toda vez que os seus contadores internos atingem um valor previamente programado/definido. As rotinas de configuração destes dois módulos foram organizados em dois arquivos do programa [6], `systick.c` e `pit.c`, respectivamente.

Como `SysTick` é considerado um módulo do núcleo (Seção 3.3.2.3 em [2]), as suas interrupções entram diretamente na lista de arbitragem da prioridade de atendimento e quando chegar a sua vez de atendimento, o fluxo de controle é desviado para a rotina de serviço pré-definida no arquivo `kinetis_sysinit.c`: `SysTick_Handler`. De acordo com o procedimento dado acima, este temporizador deve contar ciclicamente `indice5`, o índice do vetor de intervalos de tempo `cores_estaticas`, a cada 0.25s. Configurando o módulo `SysTick` para gerar interrupções periódicas de 0.25s com `initSysTick`, e habilitando a sua interrupção com `enableSysTickInterrup`, a rotina `SysTick_Handler` só precisa incrementar ciclicamente `indice5`, pois sabemos que o fluxo de controle só entrará nesta rotina em cada 0.25s.

O `PIT`, por sua vez, é um módulo considerado externo ao núcleo (Seção 3.3.2.3 em [2]), portanto é necessário configurar o `NVIC`. No programa foi setado o nível de prioridade de atendimento em 1 na rotina `enableNVICPIT` localizada no arquivo `nvic.c`. Quando atendidas suas interrupções, segue-se o fluxo de controle similar ao do `SysTick`, desviando para a rotina de serviço `PIT_IRQHandler`. Como o temporizador `PIT` é utilizado para marcar intervalos de 100ms, a rotina somente incrementa a variável global de tempo `tempo`, usado tanto para fazer a contagem regressiva do temporizador (modo de operação = 4) como permitir fazer a contagem progressiva do cronômetro do sistema (modo de operação = 2) até que se aperte novamente `PTA4` para párar o cronômetro (modo de operação = 3) . Para facilitar este controle, o `PIT` foi configurado para gerar interrupções periódicas de 0.1s com uso de `initPIT`. Note que precisamos também limpar explicitamente a *flag* de interrupção deste módulo após o seu atendimento (Seção 32.3.7 em [2]).

Você deve estar se questionando onde estão as chamadas de rotinas que enviam mensagens para LCD. Veja o laço principal no arquivo `main.c` . Em cada iteração, o processador verifica o estado em que se encontra o sistema através da variável `modo` e envia mensagens correspondentes ao LCD. Alguém poderia responder **por quê não enviar estas mensagens dentro da rotina de serviço?** E **por quê repetir indefinitamente envios num laço**?

3 *Vamos praticar o que aprendemos?* Substitua "xxxx" nos códigos em [6] para que o programa funcione conforme a descrição dada no item (2). Você precisa pensar

3.1 no valor de contagem máxima a ser setado no módulo `SYSTICK` através do argumento `periodo` da função `initSysTick(unsigned int periodo)` para atender a especificação de interrupções periódicas de 250ms.

3.2 no valor de contagem máxima a ser setado no módulo `PIT` através do argumento `periodo` da função `initPIT(unsigned int periodo)` para atender a especificação de interrupções periódicas de 100ms.

3.3 na forma como determinar o valor `indice5` quando a botoeira PTA5 é pressionada e solta.

3.4 na forma como detectar o fim do intervalo de tempo configurado após pressionar a botoeira PTA12 usando a variável `tempo`.

3.5 na forma como atualizar a contagem progressiva do cronômetro quando a botoeira PTA4 é pressionada.

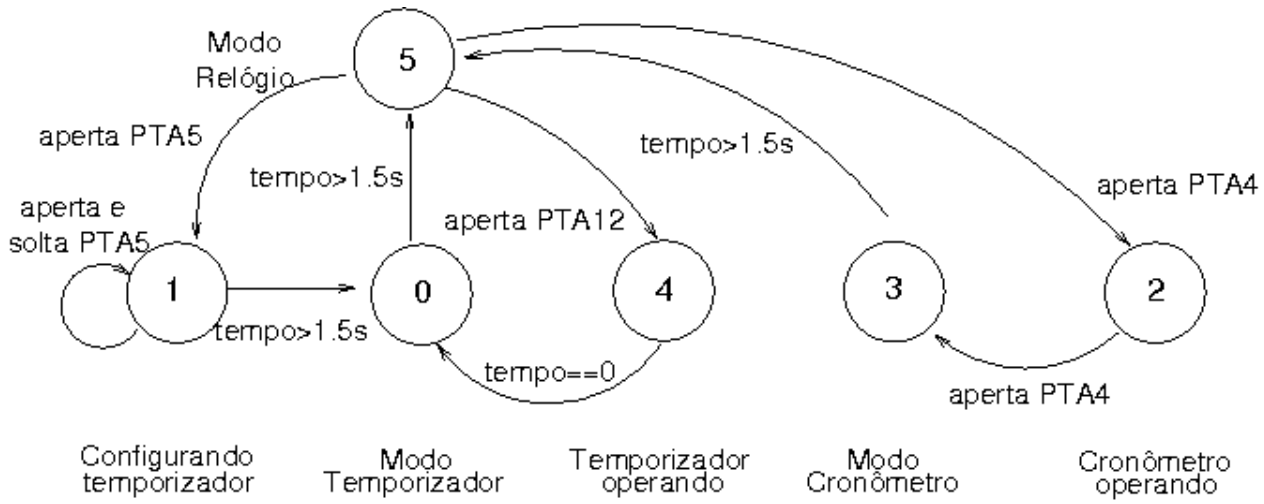
3.6 na configuração dos registradores de controle dos módulos `PORTA`, `NVIC`, `PIT` e `SYSTICK` para que os módulos `PORTA`, `PIT` e `SYSTICK` tratam corretamente as interrupções geradas pelas botoeiras e pelos temporizadores internos.

4 *Há um módulo dedicado para contagem de tempo de relógio no nosso microcontrolador?* O módulo `RTC` (*Real Time Clock*) contém um registrador `RTC_TPR` de 16 *bits* que incrementa um segundo registrador `RTC_TSR` de 32 *bits* quando a contagem do primeiro atingir 0xFFFF. Portanto, se configurarmos o módulo `RTC` de forma que a sua fonte de relógio seja 32768Hz, o conteúdo do `RTC_TSR` será incrementado em cada segundo [2]. Seção 5.7.3 em [1] mostra as três possíveis fontes do módulo `RTC`. A fonte `OSC32KCLK` requer que o oscilador de cristal que alimenta o micro-controlador seja de baixa frequência, entre 32 a 40kHz [9], ma a fresquência do oscilador do *kit* FRDMKL25Z é 8MHz [10]. A segunda alternativa seria usar um sinal externo `RTC_CLKIN`. Em [11] encontra-se uma forma de gerar com o próprio microcontrolador um sinal de 32kHz (`CLKOUT`) e alimentar o pino `RTC_CLKIN` com este sinal. O problema é que os pinos que servem os dois sinais são `PTC1` e `PTC3` que já se encontram ocupados pelo LCD [7]. Resta-se a alternativa da fonte `LPO` de 1kHz [12], ou seja, ao invés de 32768Hz, os incrementos do `RTC_TPR` ocorrerão a 1000Hz. **Como você programaria o módulo `RTC` usando `LPO` 1kHz na implementação de um relógio digital com uma resolução de segundos**? Caso você precise de alguma dica, dê uma analisada no projeto rtc.zip [13].

5 *Vamos ver se você entendeu? O relatório deste roteiro é dividido em 2 partes*

(a) (50%) Relatório 6a: Suba as respostas dos itens (2) e (3) junto com a versão de interrupção completa do projeto nvic.zip [6] contendo arquivos organizados em biblioteca (item 3) no sistema Moodle.

(b) (50%) Relatório 6b: Implemente um projeto que atende o diagrama de transições abaixo, em que a configuração do intervalo do temporizador é pela quantidade de clicks na botoeira PTA5 (quantidade de pares aperta-solta detectada) como no experimento 5 [14]. Quando a botoeira fica mais de 1.5s ociosa, o sistema entende que o usuário finalizou a entrada. O mesmo acontece com o término de cronometragem (estado 3). Utilize o temporizador `SysTick` para controlar o tempo de ociosidade. O temporizador PIT é utilizado na cronometragem dos tempos transcorridos após PTA4 ou PTA12 terem sido acionados. Além disso, é adicionado ao sistema um relógio digital que mostra o horário no formato HH:MM:SS no LCD quando não há nenhuma função ativada. Documente bem o seu código, exporte o projeto e a biblioteca em arquivos **zip**, depois de limpá-los com o comando "*Project* > *Clean*". Submeta-os no sistema Moodle.



REFERÊNCIAS

Todas as referências podem ser encontradas nos *links* abaixo ou ainda na página do curso.

[1] *Kinetis L Peripheral Module Quick Reference – Freescale Semiconductors*, Setembro 2012.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KLQRUG.pdf
[2] *KL25 Sub-Family Reference Manual – Freescale Semiconductors (doc. Number KL25P80M48SF0RM)*, Setembro 2012.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf
[3] *ARMv6-M Architecture Reference Manual – ARM Limited*.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/ARMv6-M.pdf
[4] Wu, S.T. Exceções
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/Excecoes.pdf
[5] Wu, Shin-Ting e D.S. Oliveira. Linguagem C: Operações sobre os Dados.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/Operadores.pdf
[6] nvic.zip
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/codes/nvic.zip
[7] lcdled.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2018/T/codes/lcdled.zip
[8] SATE. Esquemático do *shield* FEEC-EA871
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/complementos/Esquematico_EA871-Rev3.pdf
[9] NXP. Can I use OSC32KCLK (from system oscillator) to run RTC?
https://community.nxp.com/thread/327605
[10] *FRDM-KL25Z User's Manual – Freescale Semiconductors*, Setembro 2012.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/FRDMKL25Z.pdf
[11] Using RTC module on FRDM-KL25Z
https://community.nxp.com/docs/DOC-94734
[12] µTasker KL RTC Support
http://www.utasker.com/kinetis/KL_RTC.html
[13] rtc.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2018/T/codes/rtc.zip
[14] Wu, Shin-Ting. EA871 - Roteiro 5 – 1s2018.
http://www.dca.fee.unicamp.br/cursos/EA871/1s2018/T/roteiros/exp5.pdf

Agosto de 2016
Revisado em Fevereiro de 2017
Revisado em Julho de 2017
Revisado em Março de 2018.