EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 7 – 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.

Tratamento de interrupções não mascaráveis (*non-mascarable interrupt* NMI).

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

INTRODUÇÃO

Nos experimentos 5 e 6, para saber se as botoeiras PTA5 ou PTA12 foram pressionadas, tivemos que testar periodicamente os seus estados. Quando o nível lógico nos respectivos pinos for 0, consideramos que fechou-se o circuito. 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 disperdício 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 utilizada para um micro-controlador responder a um evento externo. E, o nosso MCU 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*) (Cap.3 de [1]) e retornando 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 inrterrupção. Figura 3-2 em [2] mostra a relação do NVIC com outros módulos do nosso MCU.

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 relocado. Esta relocação ocorre automaticamente antes da execução do nosso programa. O conteúdo da tabela de vetores é, de fato, os ponteiros a todas as rotinas de serviço correspondentes às exceções processáveis pelo nosso MCU. 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 já implementadas no nosso microcontrolador, o trabalho do programador se reduz em configurar o mecanismo de interrupção dos módulos do nosso MCU para que eles operem no modo de interrupção e em customizar a forma de tratamento de cada exceção através da reprogramação das rotinas de serviço. E, quando se trata de uma exceção externa, é necessário configurar o NVIC para ele poder arbitrar a prioridade de atendimento de múltiplas e aninhadas interrupções.

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* [5] que utiliza uma biblioteca de rotinas [6]? Este programa possui as mesmas funcionalidades do programa do experimento 6 [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 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) e habilitamos a capacidade de gerar sinais de interrupção dos pinos PTA5 e PTA12 (Seção 11.5.1 em [2], p. 183). Em termos de códigos, adicionamos três novos arquivos na biblioteca, `nvic.*`, `pit.*` e `systick.*`, e atualizamos o arquivo `pushbutton.*`. Além disso, inserimos um arquivo de rotinas de serviço `handler.*`.

   Os módulos de temporização, SysTick e PIT, são de fato contadores. Quando habilitamos a sua capacidade de gerar sinais de interrupção (Seção B3.3.3 em [3], p. 277, e Seção 32.3.6 em [2], p.579), eles os geram toda vez que os seus contadores internos atingem um valor previamente programado. As rotinas de configuração destes dois módulos foram organizados em dois arquivos, `systick.c` e `pit.c`, respectivamente.

   Como SysTick é considerado um módulo do núcleo (Seção 3.3.2.3 em [2], p. 52), as suas interupções 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`. E onde estão as instruções desta rotina? Elas estão no arquivo `handler.c`. Veja no programa [5] que SysTick é utilizado para medir o intervalo de tempo em que uma botoeria ficou pressionada; portanto, somente a variável global `tempo` é incrementado na rotina.

   O PIT, por sua vez, é um módulo considerado externo ao núcleo (Seção 3.3.2.3 em [2], p. 52). Isso significa que suas interrupções só serão atendidas se o NVIC estiver configurado para atendê-las (Seção B3.4.2 em [3], p. 283). 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 controlar o período das piscadas dos *leds* no nosso programa, a rotina somente liga e desliga os *leds* em cada interrupção. É

importante observar que no nosso micro-controlador a *flag* de interrupção de um módulo não é limpada automaticamente após o seu atendimento. É preciso limpá-la expicitamente escrevendo 1 (*write-1-to-clear*) no campo correspondente (Seção 32.3.7 em [2], p.580).

Os pinos da porta A interagem com o mundo externo ao micro-controlador. As interrupções que eles geram dependem dos fatores externos. Adicionamos a rotina `enablePushbuttonIRQA` no arquivo `pushbutton.c` para habilitar o mecanismo de interrupção dos pinos 5 e 12 da porta A (Seção 11.5.1 em [2], p.183) . O módulo PORTA é também um módulo externo ao núcleo (Seção 3.3.2.3 em [2], p. 52), portanto, o NVIC deve ser configurado, como mostra a rotina `enableNVICPTA` em `nvic.c` que setou o nível de prioridade da PORTA em 3. Quando atendida uma interrupção deste módulo, o fluxo é desviado para a rotina de serviço `PORTA_IRQHandler`. Observe que a rotina trata das interrupções da PTA5 e da PTA12, pois a nível do NVIC elas não são diferenciadas. Se quisermos que as respostas às duas botoeiras sejam diferentes, cabe a nós programarmos o tratamento diferenciado na rotina de serviço. No nosso caso, ele habilita `SysTick` para contar o tempo em que a botoeira ficou pressionada – intervalo de tempo entre uma borda de descida, pressionou, e uma borda de subida, soltou. Assim que entrarmos na rotina de serviço `PORTA_IRQHandler` lemos o valor de contagem do temporizador `tempo1`, e ficamos monitorando o número de vezes, `tempo`, que o SysTick atingiu contagem mínima. Quando se solta a botoeira lemos novamente o valor de contagem do temporizador `tempo2`. Podemos determinar a partir destes valores o intervalo de tempo em que uma botoeira ficou pressionada em termos de múltiplos de 0.25s e atribuí-lo à variável `indice`. Note que precisamos também limpar explicitamente a *flag* de interrupção deste módulo após o seu atendimento.

Finalmente, vale comentar que há ainda uma outra rotina de serviço `NMI_Handler` vazia no arquivo `handler.c`. Esta rotina é para atender uma interrupção não-mascarável (*Non-Maskable Interrupt*) . Por quê precisamos inserí-lo? Porque o pino 4 da porta A em que a botoeira NMI está conectada é configurado, *por default*, com a função NMI (Seção 10.3.1em [2], p. 162). Ao habilitarmos a interrupção da porta A, variações no sinal deste pino podem gerar interrupções não mascaráveis. Uma solução é desviar o fluxo para uma rotina que não faz absolutamente nada. Há, porém, outras soluções mais elegantes.

3. *Vamos praticar o que aprendemos?* Substitua “xxxx” em [5] e [6] com os valores corretos conforme o que você aprendeu para que o projeto apresente funcionamento similar ao [7]. Determine os valores de contagem necessários no contado do PIT para cada frequência de piscadas do *led* RGB que você considerou no experimento 6 e teste-os.

4. *Vamos ver se você entendeu?* Crie um novo projeto com o seu programa do experimento 6 (a versão com biblioteca) e adapte-o para operar com uso do mecanismo de interrupção. Exporte o projeto e a biblioteca em arquivos zip, depois do comando “*Project* > *Clean*”. Submeta-os no sistema Moodle. Suba ainda um pequeno relatório contendo (a) uma explicação detalhada como você determinou o valor `indice` com uso dos tempos, (b) como você determinou os valores de contagem máxima do PIT para cada frequência de piscadas, (c) a sua resposta à seguinte perguntas: Por quê o arquivo `handler.c` não foi colocado na biblioteca?, e (d) uma sua proposta para evitar que se gera interrupções indesejadas pelo pino 4 da porta A quando o mecanismo de interrupção estiver habilitado.

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] biblioteca.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2017/ST/codes/biblioteca.zip
[6] nvic.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2017/ST/codes/nvic.zip
[7] lcdled.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2017/ST/codes/lcdled.zip

Agosto de 2016
Revisado em Fevereiro de 2017