EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 9 – *TPM – Input Capture e Output Compare*

Profa. Wu Shin-Ting

OBJETIVO: Apresentação das funcionalidades dos módulos TPMx (*Timer/PWM*).

ASSUNTOS: Controle com maior precisão os instantes de captura dos sinais de entrada e de mudança dos estados dos sinais (digitais) de saída, programação do MKL25Z128 para processamento destes sinais via módulos TPMx.

O que você deve ser capaz ao final deste experimento?

Programar MKL25Z128 para capturar o instante em que um evento externo causou uma interrupção (*Input Capture*).

Programar MKL25Z128 para mudar o estado de um pino digital de saída num instante pré-programado (*Output Compare*).

Gerar sinais audíveis com uso de microcontroladores.

INTRODUÇÃO

Vimos no experimento 8 [11] que o módulo TPM é um termporizador cuja fonte de relógio é configurável e suporta três funções: PWM (*Pulse With Modulation*), *Input Capture* e *Output Compare*. Vimos ainda que, com a técnica de modulação por largura de pulso, podemos controlar a potência média transmitida a uma carga. Na prática, esta técnica é muito utilizada no controle de cargas com inércia como motores. Além da capacidade de gerar sinais modulados pela largura de pulso, o módulo TPM consegue "**rotular**" **as condições de exceção assíncronas com os instantes em que ocorreram** (*InputCapture*) e **gerar sinais digitais em pinos de saída nos instantes pré-programados** (*Output Compare*).

Neste experimento vamos aprender como configurar um módulo TPM para operar nos modos *Input Capture* e *Output Compare* em cima das funções do arquivo `tmp.c` que vocês completaram no experimento 8 [11]. Vamos ainda aplicar estas novas funcionalidades para solucionar os problemas que temos identificados ao longo dos exeprimentos anteriores:

1. Identificação mais precisa do intervalo de tempo em que uma botoeira fica pressionada e
2. Identificação mais precisa do instante de tempo em que ocorreu *timeout* do intervalo de tempo em que uma botoeira ficou pressionada e sinalização da ocorrência deste evento através de um *buzzer*.

EXPERIMENTO

1. ***Vamos entender como programar um módulo TPMx no nosso microcontrolador para que opere no modo InputCapture e no modo OutputCompare?*** Leia atentamente a Seção 12.4.1 em [2] procurando fazer um paralelo com o diagrama de bloco mostrado na Figura 31-1, p. 549, em [1] para entender o princípio de funcionamento de um módulo TPM no modo *Input Capture*. Quanto ao modo de operação *Output Compare*, sugiro a leitura da Seção 31.4.5 em [1] e que procure entender

as relações das formas de onda nas Figuras 31-82, 31-83 e 31-84 em [1]. Essencialmente, no modo *Output Compare* o canal coloca um nível do sinal (0, 1 ou *toggle*) pré-configurado no pino, associado ao canal, quando certas condições forem satisfeitas. Em termos de configuração por *software*, os registradores utilizados são os mesmos que utilizamos no experimento 8 [11].

2. ***Vamos ver como os conceitos são traduzidos na prática através do programa tpm_icoc.zip [3]?*** Este programa consegue detectar o intervalo de tempo em que um usuário pressionou as botoeiras PTA5 e PTA12, como nos programas anteriores (por exemplo, `ledRGBPB.zip` [5]), e acende os *leds* vermelhos do *shield* FEEC quando um usuário pressiona uma botoeira. Também como os projetos anteriores, o usuário seleciona o intervalo de tempo do temporizador através da seleção da cor exibida pelo *led* RGB. As cores exibidas variam ciclicamente na frequência de 0.5s quando a botoeira PTA5 é pressionada. A cada cor é associada um intervalo de tempo que corresponde a `3*0.5s*indice5`. Diferentemente dos projetos anteriores (por exemplo, `lcdled.zip` [8]), trocamos a função da botoeira PTA4 com a função da botoeira PTA12: PTA4 é usada para disparar o temporizador e PTA12 para manipular o cronômetro. Esta troca foi necessária porque foi percebido que os pinos PTA4 (botoeira NMI) e PTD1 (*led* azul) servem o mesmo canal TPM0_CH1. E a decisão do projeto foi priorizar a precisão das medições dos tempos do cronômetro. **Esta troca poderia ser evitada se o projeto tivesse sido pensado como todo na fase do seu desenho!**

   Nos projetos citados detectamos o evento "botoeira pressionada" por *polling*, monitorando o estado das botoeiras num laço, e a partir do projeto `nvic.zip` [9] tal evento é detectado por interrupção, configurando os pinos para levantarem bandeiras (*flags*) ao detectarem bordas de subida e de descida. Percebemos que a primeira detecção é pouco precisa, principalmente quando a frequência das piscadas for muito baixa, pois o intervalo de tempo entre dois testes do estado de uma botoeira é grande e o sistema pode perder uma "pressionada rápida" de um usuário. E na segunda alternativa, embora bem melhor, há ainda um pequeno atraso entre o momento da ocorrência do evento e o início de cômputo do intervalo nas rotinas de serviço `SysTick_Handler` e `PIT_IRQHandler`. No programa `tpm_icoc.zip` [3] aplicamos a função *Input Capture* do módulo TPMx para capturar o instante exato em que um usuário pressionou uma botoeira. Para isso transferimos as duas botoeiras PTA5 e PTA12 do módulo GPIO para um canal n do módulo TPMx através da rotina `setaMux`. Veja na Seção 10.3.1 em [1] que PTA12 somente serve o canal 0 do módulo TPM1, enquanto a PTA5 o canal 2 do TPM0. Portanto, além dos módulos TPM0 e TPM2 que usamos para controlar as cores do *led* RGB e PTA5, é também usado o módulo TPM1 neste projeto. Configuramos o canal 0 do módulo TPM1 com modo *Input Capture*, capaz de capturar bordas de descida (no momento de apertar) e bordas de subida (no momento de soltar), com o mecanismo de interrupção habilitada. Desta forma, toda vez que uma borda é capturada, dispara-se uma interrupção que é tratada pela rotina `FTM1_IRQHandler`. Da mesma forma, ao configurarmos o canal 2 do TPM0 (PTA5) com modo *Input Capture* por interrupção, precisamos incluir na rotina de serviço `FTM0_IRQHandler` o comportamento esperado da PTA5. Reusamos os códigos do projeto tpm_pwm.zip [6] para gerar uma variedade maior de cores. Vale lembrar que, sendo os módulos TPMx considerados externos ao núcleo do processador, é preciso ativar o seu processamento pelo NVIC.

   O mais interessante a ser observado é que se compararmos o projeto `tpm_icoc.zip` **[3]** e o projeto `nvic.zip` [9] é que o módulo TPMx nos permite não só capturar os eventos externos ou gerar um sinal externo como também controlar com precisão o instante em que eles ocorreram. Além da eliminação dos gastos de tempo entre um evento e um "disparo" de um contador periódio

(SysTick ou PIT), ele integra num circuito as funcionalidades de um módulo PORTx com as funcionalidades de um contador periódico. No código `tpm_icoc.zip` **[3]** capturamos as bordas da botoeira PTA5 pelo canal TPM0_CH2 e configuramos o canal TPM0_CH0 para gerar interrupções periódicas a partir do momento em que tal evento foi capturado. Como sabemos a frequência de operação e o período de operação (ciclo completo de contagem) do módulo, podemos computar com maior precisão os intervalos de tempo `tempo` que configuramos para o temporizador quando se solta a botoeira no canal TPM0_CH2. Propomos aplicar também a função de *Output Compare* para computar os intervalos de tempo `tempo` transcorridos quando o cronômetro for habilitado através do canal TPM1_CH1. Vale ainda observar que foi associado a este canal o pino PTE21. Caso liguemos neste pino um *buzzer* (pinos 3 (PTE21) e 5 (Terra) do *header* H5 [4]), configuremos o canal para "*toogle on match*" e setemos a frequência do sinal gerado numa frequência audível (na faixa de 20 a 20.000Hz), podemos escutar um som quando o cronômetro começa a correr.

Finalmente, cabe fazer uma pequena observação em relação à botoeira PTA4 que tem a função de disparo do temporizador. Note que, como no projeto `nvic.zip` [9], o temporizador compartilha com o cronômetro mesmo circuito de contagem. Ao invés do módulo PIT, é usado neste projeto o módulo TPM1, mais precisamente o canal TPM1_CH1. Se você habilitar este canal dentro da rotina de serviço `PORTA_IRQHandler` com modo "*Output Compare – toogle on match*", escutará também um som audível.

3. ***Vamos praticar o que aprendemos com tpm_icoc.zip [3]?*** Substitua “xxxx” pelos dados corretos na configuração dos módulos utilizados no projeto `tpm_icoc.zip` **[3]**. Crie um novo projeto com as rotinas `main.c` e `handler.c` completadas. Execute o projeto com um buzzer ligado.

4. ***Vamos ver se você entendeu?*** Aprimore a funcionalidade do seu projeto de relógio digital [7], incrementando a quantidade de intervalos de tempo configuráveis para o temporizador como em [11] e substituindo o controle das botoeiras PTA5 e PTA12 totalmente pelos módulos TPMx e o controlo da botoeira PTA4 pelo módulo TPM1, ficando o módulo PIT responsável somente para o relógio digital. Além disso, adicione um alarme sonoro (*buzzer*), de forma que quando o temporizador atinja zero, este alarme é ativado automaticamente por 5 segundos. . Suba-o junto com o projeto exportado no formato zip no Moodle. Não se esqueça de limpar o projeto (*Project > Clean...*) antes de exportá-lo.

REFERÊNCIAS

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

[1] KL25 Sub-Family Reference Manual – Freescale Semiconductors (doc. Number KL25P80M48SF0RM), Setembro 2012.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KL25P80M48SF0RM.pdf

[2] Kinetis L Peripheral Module Quick Reference – Freescale Semiconductors, Setembro 2012.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KLQRUG.pdf

[3] Wu, S.-T. tpm_icoc.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2017/ST/codes/tpm_icoc.zip

[4] Wu, S.-T. Ambiente de Desenvolvimento – *Hardware*
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/AmbienteDesenvolvimentoHardware.pdf

[5] Wu, S.-T.. ledRGBPB.zip
http://www.dca.fee.unicamp.br/cursos/EA871/1s2017/ST/codes/ledRGBPB.zip
[6] Wu, S.-T. tpm_pwm.zip
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/codes/tpm_pwm.zip
[7] Wu, S.-T. Roteiro 7
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/roteiros/exp7.pdf
[8] Wu, S.-T. lcdled.zip
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/codes/lcdled.zip
[9] Wu, S.-T. nvic.zip
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/codes/nvic.zip
[10] Wu, S.-T. uart_interrupcao.zip
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/codes/uart_interrupcao.zip
[11] Wu, S.-T. Roteiro 8
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/roteiros/exp8.pdf

Agosto de 2016
Revisado em Fevereiro de 2017
Revisado em Agosto de 2017