EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 10 – *TPM - PWM*

Profa. Wu Shin-Ting

OBJETIVO: Apresentação das funcionalidades PWM do módulo TPMx (*Timer/PWM*).

ASSUNTOS: Geração de sinais PWM (*Pulse Width Modulation*), programação do MKL25Z128 para processamento destes sinais via módulos TPMx.

O que você deve ser capaz ao final deste experimento?

Entender o princípio de funcionamento de TPMx.

Saber programar a base de tempo do contador de um módulo TPMx.

Saber configurá-lo para a função PWM.

Programar MKL25Z128 para gerar sinais PWM de ciclos de trabalho de interesse.

INTRODUÇÃO

Além do *timer* integrado ao núcleo, *SysTick* (Seção B.3.3 em [1]), e dos módulos PIT (*Periodic Interrupt Timer*) (Cap. 32 de [2]), LPTMR (*Low-Power Timer*) (Cap. 33 de [2]), RTC (*Real Time Clock*) (Cap. 34 de [2]), o nosso microcontrolador dispões ainda de três módulos `TPM` (Timer/PWM), 1 com 6 canais e 2 com 2 canais.

Todos os canais de um mesmo módulo TPMx compartilham um mesmo contador de 16 *bits*, `TPMx_CNT`, que pode contar de forma crescente (*up*) ou crecente-decrescente (*up-down*) (Figura 31-1, p. 549, de [2]). São alimentados por um sinal de relógio assíncrono (independente) do núcleo. A fonte de relógio é selecionável pelos campos `SIM_SOPT2[TPMSRC]` e `SIM_SOPT2[PLLFLLSEL]` ou pelos *bits* `TPMx_SC[CMOD]` quando se trata de uma fonte externa. De acordo com as Seções 31.4.2, p. 562, e 31.4.3, p. 562, de [1], o período de contagem máxima deste contador depende, além da frequência do relógio selecionado, dos valores setados em `TPMx_MOD[MOD]` (valor de referência para contagem máxima) e em `TPMx_SC[PS]` (*prescaler* ou divisor de frequência). O estouro na contagem é uma condição de exceção e a bandeira `TPMx_SC[TOF]` é setada. Se o *bit* `TPMx_SC[TOIE]=1` e o controlador NVIC é adequadamente configurado (Seção B3.4, p. 281, em [1]), o fluxo de controle é, então, automaticamente desviado para a sua rotina de serviço cujo endereço está pré-definido no arquivo `Project_Settings/Startup_Code/kinetis_sysinit.c` gerado pelo IDE CodeWarrior. Através do registrador de controle `TPMx_CONF` pode-se configurar o instante e o valor com que o contador deve ser recarregado (Seções 3.8.1.3, p. 84, e 31.3.7, p.559, em [2]), como também qual fonte de *clock* é compartilhada entre os 3 módulos.

A principal característica do módulo TPM é que cada um dos seus canais pode ser configurado para operar num dos três modos: *Output Compare* (Seções 31.4.4 em [2]), *Input Capture* (Seções 31.4.5 em [2]) e *PWM* (Seções 31.4.6 e 31.4.7 em [2]). É associado a cada canal um registrador de controle `TPMx_CnSC` para configurar individualmente o seu modo de operação e um registrador de dados `TPMx_CnV[VAL]` cujo uso depende do modo de operação configurado. Para a função ***PWM (Pulse Width Modulation)*** [3], a largura do pulso dentro de um período de contagem máxima programado depende do valor setado no registrador `TPMx_CnV[VAL]` do canal `n`. Porisso, é possível controlar, por *software*, a relação da largura do nível alto em relação ao período do sinal é conhecido como ciclo

de trabalho (*duty cycle*) (Figura 1). Dentre o modo de PWM distingue-se ainda o modo PWM alinhado com a borda (EPWM, alterna quando `TPMx_CNT` = 0 como mostra Figura 31-87, p.568, em [2]) e o modo PWM alinhado com o centro do pulso (CPWM, atinge o meio do pulso quando `TPMx_CNT`=0 como ilustra Figura 31-88, p.569, em [2]). Como a contagem em `m` ocorre na transição de `m-1` para `m`, para que tenhamos um ciclo de trabalho 100% é necessário que o valor setado em `TPMx_CnV[VAL]` seja maior do que o valor setado em `TPMx_MOD`. É ainda configurável pelos *bits* `TPMx_CnSC[ELSnB:ELSnA]` a polaridade dos sinais, tal que a largura do pulso possa ser de nível lógico 0 (*low*) ou de nível lógico 1 (*high*).

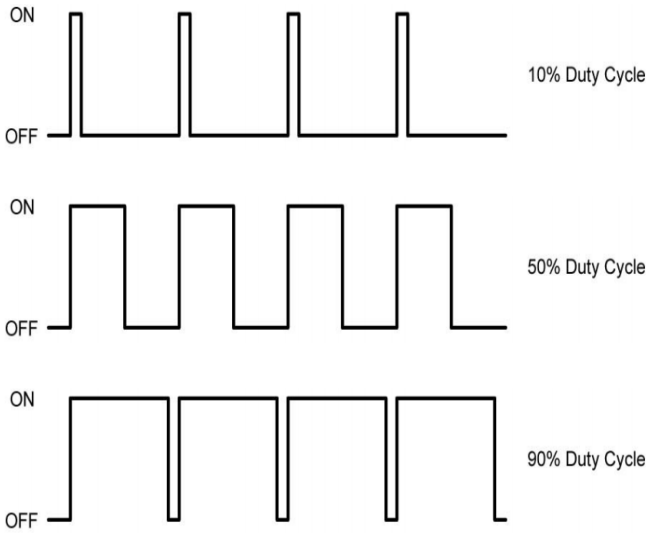

Figura 1: Modulação por largura de pulso [3]

EXPERIMENTO

Neste experimento vamos ampliar a paleta de cores dos *leds* no nosso sistema usando o conceito de formação aditiva (da energia) de cores primárias (de frequências correspondentes a) *R(ed)*, *G(reen)* e *B(lue)*. Baseado na tabela de cores [4], são somadas as contribuições ponderadas das componentes primárias para obter uma variedade maior de cores. Por exemplo, a cor verde floresta (*Forest Green*) é formada por (34,139,34) em relação ao valor máximo 255, ou seja, (0.13,0.55,0.13) da intensidade máxima de cada cor primária (R,G,B). Vamos criar um novo projeto **led_multicores** com base no projeto **escolha_cor_interrupt** (experimento 7). Neste novo projeto é estendida a paleta de cores para 14 cores, que inclui as 8 cores já existentes. Além disso, é mostrado na primeira linha do visor do LCD a percentagem com duas casas decimais de cada cor primária na formação da cor selecionada.

1. Com uso de um analisador lógico, analise comparativamente os sinais de alimentação do *led* azul no projeto **timer_pwm**, mais especificamente a diferença entre os períodos com prescaler (ps) 32 e 128, a diferença entre os períodos dos sinais configurados com CPWMS=0 e CPWMS=1, a diferença entre os níveis de sinais para a polaridade *high* e *low*, e a diferença nos ciclos de trabalho. O que você pode concluir sobre a relação entre o ciclo de trabalho e os parâmetros ps, CPWMS e polaridade?
2. Quais canais de quais TPMx servem os pinos PTE21, PTE22 e PTE23 em que os leds R, G e B estão conectados, respectivamente? Setando TPMx_MOD em 49152. Qual é o período (de um ciclo de contagem completa) do módulo para a configuração de frequência habilitada no nosso

microcontrolador? Qual valor de prescaler devemos setar para que o período seja 0.15s? Justifique.

3. Especifique as 6 cores adicionais incluídas no seu projeto **led_multicores** em termos da percentagem da intensidade máxima de cada *led*. Detemine, para cada cor, os valores a serem carregados no registrador `TPMx_CnV[VAL]` de cada canal do *led*.
4. Escreva o pseudocódigo do fluxo de controle principal do seu projeto, onde se encontram tanto as instruções para atualizar o valor do TPMx_CnV quanto o processamento do LCD. Destaque as variáveis definidas para coordenar as ações na rotina de serviço PORTA_IRQHandler e no fluxo de controle prncipal.
5. Implemente a função ftoa_int (unsigned int valor, uint8_t casas_decimais, char *string) que converte um valor inteiro numa string com a vírgula deslocada de duas casas ára esquerda. Não se esqueçam de tratar os zeros que precisam ser inseridos quando o valor inteiro é menor que 100.
6. Implemente o aplicativo **led_multicores** em C.
7. Documente todas as funções que não foram geradas pelo IDE CodeWarrior.

RELATÓRIO

Para este experimento, responda as questões 1 a 4 num arquivo em **pdf**, implemente e documente o projeto **led_multicores.** Exporte o projeto no ambiente IDE CodeWarrior para um arquivo em formato zip. Suba **os dois arquivos, em separado**, no sistema *Moodle*. Não se esqueça de identificar todos os seus arquivos de códigos com a palavra reservada "@author" de Doxygen.

REFERÊNCIAS

[1] ARMv6-M Architecture Reference Manual – ARM Limited.
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/ARMv6-M.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] PWM – Modulação por Largura do Pulso
http://www.mecaweb.com.br/eletronica/content/e_pwm

[4] RGB to Color Name Mapping (Triplet and Hex)
https://web.njit.edu/~kevin/rgb.txt.html

Revisado em Novembro de 2020
Revisado em Fevereiro de 2017
Agosto de 2016