

EA871 – LAB. DE PROGRAMAÇÃO BÁSICA DE SISTEMAS DIGITAIS

EXPERIMENTO 8 – 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 configurar diferentes funcionalidades para cada um dos (até oito) canais do TPMx.

Programar MKL25Z128 para gerar sinais PWM.

Saber declarar e usar variáveis de endereços dos registradores.

INTRODUÇÃO

Além dos *timers* integrados ao núcleo, Watchdog (Seção 3.4.10 em [1]) e SysTick (Seção B3.3 em [2]), o nosso microcontrolador dispõe ainda outros periféricos de temporização, como os módulos LPTMR (*Low-Power Timer*) (Cap. 33 de [1]), TPM (*Timer/PWM*) (Cap. 31 de [1]) e PIT (*Periodic Interrupt Timer*) (Cap. 32 de [1]). O primeiro é um temporizador de 16 *bits* de baixo consumo de potência, o segundo é constituído de um contador LPTPM (*Low-Power TPM*) de 16 *bits*, também de baixo consumo, enquanto o terceiro é um temporizador configurável entre 32 e 64 *bits*, com a função específica de disparar interrupções condicionadas a condições pré-programadas. Trabalhamos com os módulos SysTick e PIT no roteiro 7 [8].

O módulo TPM (*Timer/PWM*) contém 2 a 8 canais compartilhando um mesmo contador de 16 *bits*, TPMx_CNT, que pode contar de forma crescente (*up*) ou crescente-decrescente (*up-down*) (Figura 31-1 em [1]). Como UARTx, TPM dispõe de um sinal de relógio assíncrono (independente) para este contador. A fonte de relógio é selecionável pelos campos SIM_SOPT2[TPMSRC] e SIM_SOPT2[PLLFLSEL] ou pelos *bits* TPMx_SC[CMOD] quando se trata de uma fonte externa. O período de contagem deste contador depende, além da frequência do relógio selecionado, dos valores setados em TPMx_MOD[MOD] (módulo do valor máximo de contagem, Seção 31.4.3 em [1]) e em TPMx_SC[PS] (pré-escala ou divisor de frequência, Seção 31.4.2 em [1]). O estouro na contagem é uma condição de exceção e a bandeira TPMx_SC[TOF] é setada nesta condição. Se o *bit* TPMx_SC[TOIE]=1 e o controlador NVIC é adequadamente configurado (Seção B3.4 em [2]), o fluxo de controle é, então, automaticamente desviado para a rotina de serviço correspondente cujo endereço está registrado na Tabela de Vetores de Interrupção definida no arquivo Project_Settings/Startup_Code/kinetis_sysinit.c. Através do registrador de controle TPMx_CONF pode-se configurar o instante e o valor com que o contador deve ser recarregado no modo de depuração ou de espera (Seção 31.3.7 em [1]). A principal característica do módulo TPM é que cada um dos seus canais pode ser configurado para operar num dos três modos: *Input Capture* (Seções 31.4.4 em [1]), *Output Compare* (Seções 31.4.5 em [1]) e *PWM* (Seções 31.4.6 e 31.4.7 em [1]).

[1]). É associado a cada canal um registrador de controle `TPMx_CnSC` para configurar individualmente o seu modo de operação (Seção 31.3.4 em [1]) e um registrador de dados `TPMx_CnV [VAL]` cujo uso depende do modo de operação configurado (Seção 31.3.5 em [1]).

- a. **Input Capture:** Quando o tipo de borda ou de nível pré-configurado for detectado no sinal do pino de entrada alocado a um canal n , o valor do contador `TPMx_CNT [COUNT]` é capturado no registrador `TPMx_CnV [VAL]`. Esta captura é a condição de interrupção deste modo de operação (Seção 31.4.4 em [1]). Para que o sinal de entrada seja corretamente amostrado, sua frequência deve ser 2 vezes menor que a frequência de contagem do relógio do módulo `TPMx`.
- b. **Output Compare:** Quando o valor do contador `TPMx_CNT [COUNT]` fique igual ao valor setado no registrador `TPMx_CnV [VAL]` do canal n , é colocado o valor pré-definido (nível lógico 0 ou 1) no pino de saída alocado ao canal. A transição em que os dois valores fiquem iguais é a condição de interrupção deste modo de operação (Seção 31.4.5 em [1]).
- c. **PWM (Pulse Width Modulation)**[4]: Quando o valor do contador `TPMx_CNT [COUNT]` fique igual ao valor setado no registrador `TPMx_CnV [VAL]` do canal n , o sinal no pino de saída do canal é alternado. A transição em que os dois valores fiquem iguais é a condição de interrupção deste modo de operação (Seções 31.4.6 e 31.4.7 em [1]). O nível inicial do sinal é pré-configurado pelos campos do registrador de controle `TPMx_CnSC [ELSnB:ELSnA]`. A relação da largura do nível alto em relação ao período do sinal é conhecido como ciclo de trabalho. 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 em [1]) e o modo PWM alinhado com o centro do pulso (CPWM, o sinal alterna quando o valor do `TPMx_CNT` fique igual ao valor setado no `TPMx_CnV` tanto na contagem decrescente quanto na contagem crescente; portanto, o meio do pulso coincide com `TPMx_CNT=0` como ilustra Figura 31-88 em [1]). Como as transições dos níveis dos sinais ocorrem nas transições dos estados do contador, 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` no modo de operação EPWM.

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 [3], vamos somar contribuições parciais das componentes primárias para obter mais variações 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 das cores primárias (R,G,B). Como podemos gerar luzes de intensidades diferentes com um sistema digital? O que acham de aplicar modulação por largura de pulso (PWM) para variar a potência média de energia entregue a cada *led*? Quanto maior é o ciclo de trabalho, maior será a potência aplicada no *led* e, portanto, maior será a intensidade luminosa do *led*.

O sinal PWM é muito utilizado no controle da potência média aplicada a uma carga [4]. Localize no esquemático do *shield* FEEC (Anexo 1 em [7]) um transistor Darlingon TIP31 que opera como chave eletrônica. Quando `PTB0` estiver em 1, o transistor entra no estado de saturação e conduz; caso contrário, ele fica no estado de corte. Se ligarmos uma carga, como um *cooler*, entre os pinos 2 e 4 do *header* H6 e uma fonte de alimentação de 12V DC, podemos controlar a velocidade do *cooler* por *software* programando o ciclo de trabalho do sinal PWM.

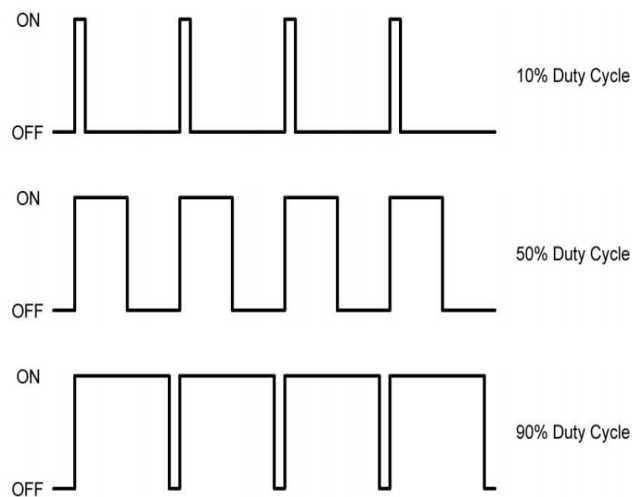


Figura 1: Modulação por largura de pulso [4]

EXPERIMENTO

1. **Vamos entender como programar os sinais PWM com uso do módulo TPMx no nosso microcontrolador?** Leia atentamente a Seção 12.3 em [5], procurando fazer um paralelo com o diagrama de bloco mostrado na Figura 31-1 em [1] e as formas de onda geradas no pino de saída em cada canal em função dos valores setados nos registradores `TPMx_MOD`, `TPMx_CnV` e o modo de operação nas Figuras 31-82 a 31-90 em [1]. Veja que os canais de um mesmo módulo `TPMx` compartilham o mesmo relógio mas tem registradores de controle e de estado separados.
2. **Vamos ver como os conceitos são traduzidos na prática através do programa `tpm_pwm.zip` [6]?** Este programa é um variante do programa `hello_world.zip` [9]: uma sequência cíclica de 2 piscadas de cada cor. Ao invés de conectar os três *leds* RGB nos pinos GPIO, reconfiguramos-os para que sejam saídas dos módulos `TPMx`. Cada pino só pode servir um canal `n` de um módulo `TPMx`, como mostra a tabela da Seção 10.3.1 em [1], de forma que possamos aplicar sinais PWM para gerar uma gama maior de cores. Como os pinos dos *leds* R, G e B estão ligados nos pinos PTB18, PTB19 e PTD1 [7], eles só podem servir os canais `TPM2_CH0`, `TPM2_CH1` e `TPM0_CH1` quando o campo `MUX` dos respectivos registradores `PORTx_PCRn` estiver apropriadamente setado. Para flexibilizar a configuração das funções dos pinos, adicionamos neste experimento os arquivos `port.*` na nossa biblioteca com a função `setaMux`, `resetMux` e `enableIRQ`. Inserimos o arquivo `tpm.*` contendo as funções de configuração dos três módulos `TPM0`, `TPM1` e `TPM2` do nosso micro-controlador. E adicionamos novas funções no arquivo `ledRGB.c`.

Pelo diagrama de blocos da Figura 31-1 em [1], podemos dividir a configuração de cada canal de um módulo `TPMx` em duas partes: a configuração do contador do módulo e a configuração dos canais do módulo que compartilham este mesmo contador. A fonte dos sinais de relógio como também o divisor de frequência destes sinais são setados na rotina `initTPM` do nosso programa. Embora o contador dos módulos `TPMx` tenha 16 *bits*, é configurável o valor máximo que o contador pode atingir antes de retornar para zero através do registrador `TPM_MOD`. Mesmo mantendo `MCGFLLCLK` (20.971520MHz) como a fonte de relógio dos módulos `TPMx`, podemos configurar com `initTPM` uma grande variedade de intervalos de tempo para cada ciclo de

contagem (período) de um módulo TPMx. Equações na Seção 12.3.6 em [5] mostram como se obtém estes valores a partir dos valores `TPMx_MOD`, `TPMx_SC_PS` e `TPMx_SC_CPWMS` setados.

Definida a base de tempo de um módulo TPMx, podemos configurar os canais para operarem no modo PWM. Módulos TPMx conseguem gerar dois tipos de sinais PWM, o sinal alinhado pela borda (EPWM) e o sinal alinhado pelo centro (CPWM) em dois sub-modos, *high-true* e *low-true pulses* (Seção 31.3.4 em [1]). Essencialmente, no modo PWM o circuito do módulo alterna o sinal de saída quando o contador atinge o valor setado no registrador `TPMx_CnV` do canal e coloca o sinal de saída no estado configurado, *high* ou *low*, no final de cada ciclo de contagem,. Assim, dependendo do estado inicial do sinal de saída e do valor setado no registrador `TPMx_CnV`, podemos ter, como mostra a Figura 1, pulsos de diferentes larguras (em nível alto) dentro de um período. Alimentando os *leds* R, G e B com estes sinais podemos ter uma gama maior de cores. Para contemplar esta variabilidade dentro de um período, adicionamos uma nova estrutura de dados `cor_pf` e novas rotinas `*PWM` no arquivo `ledRGB.h` para representar as componentes de uma cor em valores fracionários em relação à contagem máxima, no lugar de valores binários ACESO e APAGADO como no projeto `hello_world.zip` [9].

Observe que, ao invés de "ligar" e "desligar" o modo PWM dos canais, mantemos desligado o modo quando a largura do pulso é menor que 0.1% do período do contador do módulo na rotina `ligaLedRGBPWM`. Esta decisão evita chaveamentos muito rápidos que o nosso micro-contrlador pode não responder a contento sem comprometer a qualidade visual dos efeitos de piscada. Outro detalhes de operação do módulo TPMx é o instante de alternância do sinal de saída quando o contador atinge o valor setado no registrador `TPMx_CnV`: no início de um pulso. Portanto, quando queremos que o ciclo de trabalho seja 100%, precisamos setar um valor maior que a contagem máxima no modo de operação EPWM.

Módulos TPMx são providos de mecanismo de interrupção a nível do módulo e a nível do canal. A nível do módulo, pode-se programar a geração de um evento de interrupção quando o contador atinge a contagem máxima pré-estabelecida; e a nível do canal, a ocorrência de um evento do modo para o qual o canal foi configurado pode disparar uma interrupção. Embora não seja usado neste experimento, é incluído no `tpm_pwm.zip` os arquivos `nvic.*` contendo as funções de ativação de interrupção dos módulos TPMx no NVIC.

Vale comentar que foi aplicada nos códigos `tpm_pwm.zip` [6] a técnica de variável de endereços de registradores para processar diferentes módulos e diferentes canais numa mesma rotina uma vez que o procedimento para configurá-los é o mesmo. No arquivo `tpm.c` foram definidos as variáveis globais de endereços dos registradores com qualificador `volatile` para viablizar o uso de um mesmo conjunto de instruções sobre registradores de mesma natureza:

```
volatile uint32_t *tpm_sc;

volatile uint32_t *tpm_cnt;

volatile uint32_t *tpm_mod;

volatile uint32_t *tpm_cnsc;

volatile uint32_t *tpm_cnv;

volatile uint32_t *tpm_status;
```

```
volatile uint32_t *tpm_conf;
```

No arquivo `derivative.h/MKL25Z4.h` encontram-se definidas as macros dos endereços de todos os registradores do nosso micro-controlador nos blocos de códigos denominados "Register accessors". Estes endereços seguem a mesma convenção de nomes utilizados em [1] terminados com `REG`. Por exemplo, o endereço do registrador `TPM0_SC` é `&TPM0_SC_REG(TPM0_BASE_PTR)` a partir do endereço-base `TPM0_BASE_PTR (0x40038000u)`

```
#define TPM0_BASE_PTR ((TPM_MemMapPtr) 0x40038000u)
```

Aplicamos esta mesma técnica para os registradores do módulo `PORT` no arquivo `port.c`. As instruções nas rotinas `setaMux`, `resetaMux` e `enableIRQ` são aplicáveis para os registradores de qualquer pino. Basta passar para estas rotinas o endereço do registrador `PORTx_PCRn` como foi feito na rotina `initLedRGBPWM` da versão atualizada do arquivo `ledRGB.c`.

3. *Vamos praticar o que aprendemos com `tpm_pwm.zip` [6]?*

- Substitua "xxxx" pelos dados corretos e complete as rotinas definidas nos arquivos `port.c`, `tpm.c`, `nvic.c` e `ledRGB.c`. **Adicione os arquivos novos na biblioteca e as novas rotinas nos arquivos já existentes da biblioteca**, crie um novo projeto com a rotina `main.c`. Execute o projeto.
- Calcule o período (ciclo completo de uma contagem até o valor `TPMx_MOD`) do sinal PWM que alimenta os três *leds*.
- Compare no arquivo `MKL25Z4.h` a definição das macros dos registradores do micro-controlador no bloco de códigos "Register accessors" e a das macros no bloco de códigos "Register instance definitions" que utilizamos até o experimento 7 [8]. Podemos substituir, por exemplo, no nosso código `&TPM0_SC_REG(TPM0_BASE_PTR)` por `&TPM0_SC`? Justifique.
- Em relação à técnica de programação chamo novamente atenção ao uso de desigualdade "`<`" nas comparações com "`0.0`" na rotina `ligaLedRGBPWM` do arquivo `ledRGB.c`. O que acontecerá se substituirmos pela igualdade "`== 0.0`"?

4. *Vamos ver se você entendeu?* Crie um novo projeto com o seu programa do experimento 7 [8] e adapte-o para ter 16 alternativas de cores, incluindo preta (apagado). Utilize 16 como fator de pré-escala, PS, e o modo centralizado do sinal PWM (CPWM), mantendo porém o mesmo período do contador `TPMx_CNT` do `tpm_pwm.zip` [6]. Submeta um relatório contendo as respostas das questões b—d do item 3 e o novo projeto junto com a biblioteca com os arquivos atualizados no sistema Moodle. Limpe os dois projetos (*Project > Clean ...*) antes de exportá-los.

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] *ARMv6-M Architecture Reference Manual – ARM Limited*.
<ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/ARMv6-M.pdf>
- [3] *RGB to Color Name Mapping (Triplet and Hex)*
<https://web.njit.edu/~kevin/rgb.txt.html>
- [4] PWM – Modulação por Largura do Pulso
http://www.mecaweb.com.br/eletronica/content/e_pwm
- [5] *Kinetis L Peripheral Module Quick Reference – Freescale Semiconductors*, Setembro 2012.
<ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/ARM/KLQRUG.pdf>
- [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. Ambiente de Desenvolvimento – *Hardware*
ftp://ftp.dca.fee.unicamp.br/pub/docs/ea871/apostila_C/AmbienteDesenvolvimentoHardware.pdf
- [8] Wu, S.-T. EA871 - Roteiro 7 - 2s2017
<http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/roteiros/exp7.pdf>
- [9] Wu, S.-T. hello_world.zip
http://www.dca.fee.unicamp.br/cursos/EA871/2s2017/EM/codes/hello_world.zip

Agosto de 2016

Revisado em Fevereiro de 2017

Revisado em Agosto de 2017