Freescale Semiconductor
应用笔记

Document Number: AN5121
Rev 0, 04/2015

利用 TPM 和 DMA 生成固定的 PWM 脉冲数

内容

1 简介 1
2 TPM 异步特性 2
3 所需硬件和软件 2
4 TPM / DMA 配置 2
4.1 TPM/DMA 软件流程 3
4.2 TPM 配置 3
4.3 DMA 配置 4
5 代码列表 4
6 参考资料 8
7 修订历史记录 8

1 简介

定时器/PWM 模块（TPM）是基于多年来广泛用于飞思卡尔 8 位微处理器（MCU）的简单定时器。Kinetis MCU 中使用的 TPM 扩展了功能以支持在低功耗模式下运行，通过设定可以在低功耗模式下保持正常功能的异步时钟计数器和比较与捕捉寄存器实现。

该用例在 DMA 模式下使用 TPM 以生成固定数量的 PWM 脉冲。还可以通过 DMA 控制器来改变这些脉冲的占空比。该用例的主要意图是利用 DMA 传输字节计数寄存器 DMA_DSR_BCRn [BCR]进行：

1. 计数所生成的脉冲。
2. 在 DMA 传输完成中断发生时停止 TPM 模块的 DMA 请求。
3. 使能 TPM 通道中断以完成最后一个 PWM 脉冲生成。

在 TPM 通道中断例程处，必须重新配置 TPM / DMA 以产生更多脉冲。由于 TPM 具有异步 DMA 功能，这可以使 MCU 进入 VLPS 模式，异步 TPM / DMA 中断可以将 MCU 从 VLPS 模式中唤醒。

该用例还可作为 FTM 用作传统 TPM 模式时的参考不过，当前在 Kinetis MCU 中使用的 FTM 不支持本文档中提及的异步 DMA 特性。

2 TPM 异步特性

TPM 模块使用单个的时钟源，该时钟可从 OSCERCLK、MCGIRCLK 或 MCGPCLK 中选择。选定的时钟源由 SIM_SOPT2 [TPMSRC]控制。每个 TPM 还支持外部时钟模式（TPM_SC [CMOD]=1x），该模式下计数器在检测到外部时钟输入的同步（与选定的 TPM 时钟源同步）上升沿之后递增。可用的外部时钟（为 TPM_CLKIN0 或 TPM_CLKIN1）由 SIM_SOPT4 [TPMxCLKSEL]控制寄存器进行选择。这些时钟源均为异步，并且可选择用于 VLPS 模式。

3 所需硬件和软件

本文档说明了基于飞思卡尔 Freedom 系统的示例应用。该基本概念也可以方便地在定制硬件上实现。

可使用以下 FRDM 系统板方便地构建本应用：

- FRDM-KL43Z

该用例可以非常方便地集成到 SDK 1.2 GA 软件包中，详情请参见第 5 章。您可以从 freescale.com/ksdk 下载完整的软件包。

4 TPM / DMA 配置

该用例使用 TPM 通道标志以触发 DMA 进入周期数据搬移模式，从而将数据从 SRAM 或 Flash 转移到 TPM 通道值寄存器。在 TPM 定时器溢出后，将重载 TPM 通道值，然后 PWM 脉冲的占空比将根据输入到 SRAM 或 Flash 中的数值按顺序进行更改。DMA 传输字节计数器寄存器 DMA_DSR_BCRn [BCR]将持续计数所生成的 PWM 脉冲数。生成一定数量的 PWM 波形之后，会发生 DMA 传输完成中断，随后可以停止 TPM 模块或使能 TPM 通道中断来完成最后一个 PWM 脉冲生成。在 TPM 通道中断例程处，必须重新配置 TPM / DMA 以生成更多脉冲。由于 TPM 具有异步 DMA 功能，MCU 可以进入 VLPS 模式，异步 TPM / DMA 中断可以将 MCU 从 VLPS 模式中唤醒。

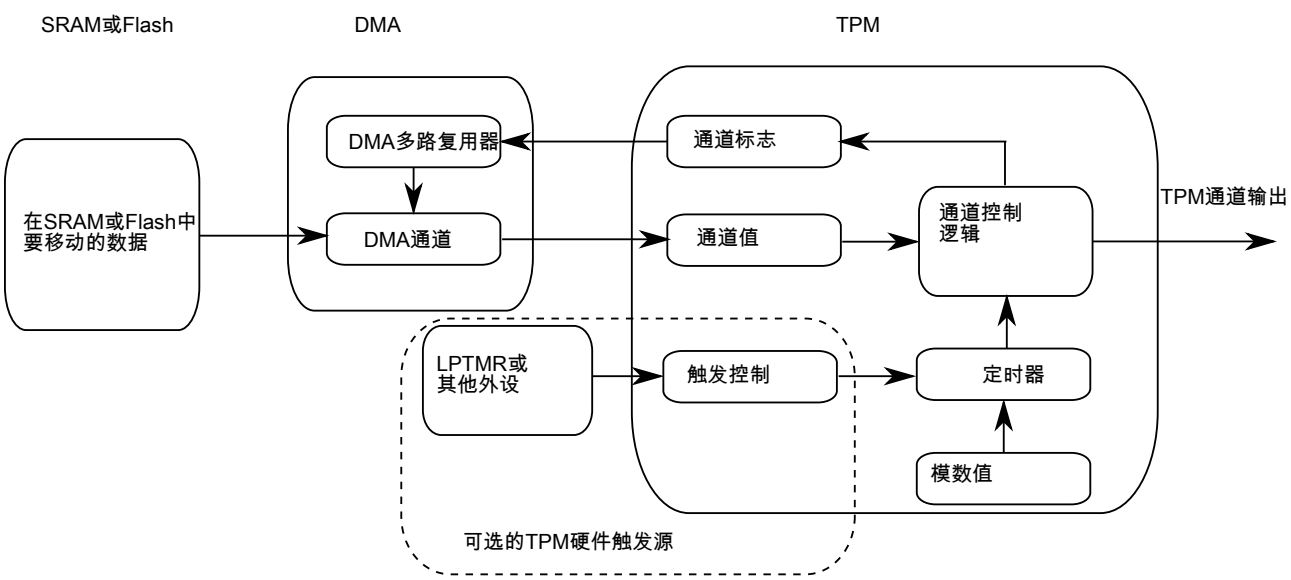

图 1. TPM/DMA 流程

具有 4 个脉冲的 TPM 输出波形如下所示：

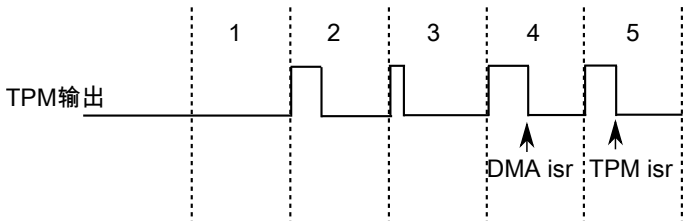

图 2. TPM 输出波形

4.1 TPM/DMA 软件流程

由于 TPM 通道值只能在 EPWM 模式下 TPM 计数器溢出后才能重载，这会在这些脉冲之前存在一个 TPM 通道值为 0 的额外 TPM 定时器周期。以生成 4 个脉冲为例，详细说明如下：

阶段 1：TPM 以初始通道值 0 开始运行。这可由硬件或软件触发，DMA BCR 减 1，由于通道值为 0，因此该阶段无脉冲输出。

阶段 2：DMA 在 PWM 脉冲的下降沿将输出移入 TPM 通道值寄存器，DMA BCR 减 1。

阶段 3：与阶段 2 一致。

阶段 4：与阶段 2 一致，DMA BCR 递减为 0，进入 DMA 中断。使能 TPM 通道中断以完成最后一个 PWM 波形生成。

阶段 5：在 PWM 下降沿发生 TPM 通道中断，设置 TPM 时钟为无时钟，复位 TPM 计数器，禁止 TPM 通道中断。重新初始化 DMA 和 TPM，使能 TPM 时钟运行。

注

如果您希望仅使用 DMA 中断而不使用 TPM 中断，则必须通过软件的阶段 1 和阶段 5 输入第一个脉冲 TPM 通道值。同时，在阶段 4，DMA 中断必须重新初始化 DMA 和 TPM 模块为运行模式。必须确保最后一个 PWM 周期能够正确地生成脉冲。TPM 硬件触发模式不支持该方法。

异步 DMA 和 TPM 中断均能将 MCU 从 VLPS 模式中唤醒。

如果不希望改变 PWM 占空比，必须将 DMA 传输目的地设为 SRAM 中的一个空值区域。

4.2 TPM 配置

TPM 在边沿对齐 PWM 模式中设置，通道标志触发 DMA 传输。TPM 配置的详细信息如下所示：

- 状态和控制（TPMx_SC）。
  - CPWMS：选择 EPWM，设为 0。
  - CMOD：选择 TPM 计数器时钟 – 1。
- 模数（TPMx_MOD）
  - MOD：选择定时器溢出周期。
- 通道（n）状态和控制（TPMx_CnSC）
  - MSB MSA ELSB ELSA：EPWM，高真脉冲模式 – 1010。
  - DMA：使能通道标志 DMA 请求 – 1。
- 通道（n）值（TPMx_CnV）
  - VAL：PWM 占空比，初始值为 0。
- 配置（TPMx_CONF）

- TRGSEL：选择 TPM 的触发源（如果使用其他外设触发 TPM）。
- TRGSRC：选择触发源类型（由其他外设外部触发）– 0。
- TRGPOL：选择触发极性（高）– 0。
- CPOT：选择计数器暂停或不暂停（触发时不暂停）– 0。
- CROT：选择计数器重载或不重载（触发时不重载）– 0。
- CSOO：选择溢出后计数器停止或不停止（溢出时不停止）– 0。
- CSOT：选择计数器何时开始计数（如果使用硬件触发模式，则在发生触发事件时开始运行）– 1。

4.3 DMA 配置

DMA 通道多路复用器配置为 TPM 通道请求源。DMA 控制器设置为周期数据搬移模式，在每个 TPM 请求时移动仅有的一个 16 位数据。DMA / DMA 多路复用器配置的详细信息如下所示：

- 通道配置寄存器（DMAMUXx_CHCFGn）
  - SOURCE：选择 TPM 通道标志作为触发源。
  - ENBL：使能该 DMA 通道复用 – 1。
- 源地址寄存器（DMA_SARn）
  - SAR：在 SRAM 或 Flash 中选择源数据地址。
- 目标地址寄存器（DMA_DARn）
  - DAR：选择 TPM 通道值寄存器的目标数据地址，在不需要改变 PWM 占空比的情况下选择 SRAM 中的空数据。
- DMA 状态寄存器/字节计数寄存器（DMA_DSR_BCRn）
  - BCR：要移动的数据，意味着需要生成 PWM 脉冲。由于数据为 16 位长，BCR 值为脉冲数 x2。
- DMA 控制寄存器（DMA_DCRn）
  - EINT：使能 DMA 完成中断 – 1。
  - ERQ：使能外设 DMA 请求 – 1。
  - CS：使能周期数据搬移模式 – 1。
  - EADREQ：使能异步 DMA – 1。
  - SINC：选择源地址递增 – 1。
  - SSIZE：选择源数据大小为 16 位 – 10。
  - DINC：选择目标地址不递增 – 0。
  - DSIZE：选择目标数据大小为 16 位 – 10。
  - D_REQ：在 DMA 完成后禁止外设 DMA 请求 – 1。

5 代码列表

该用例主要包含两个文件。您可以在任何 SDK 1.2 KL43 示例中使用这些文件来取代现有文件。例如，您可以打开 KL43 IAR hello world 演示程序，在“source”文件夹中删除文件（*main.c* 和 *fsl_lptmr_irq.c*），然后在该文件夹中插入这两个文件。您必须编译并调试该用例。这两个文件的内容显示如下：

Main.c

```
////////////////////////////////////////////////////////////////////////////////
// Includes
////////////////////////////////////////////////////////////////////////////////
// Standard C Included Files
#include <stdio.h>
#include <string.h>

// SDK Included Files
#include "fsl_tpm_driver.h"
#include "board.h"
#include "fsl_debug_console.h"
#include "fsl_dma_driver.h"
```

```
#include "fsl_dma_hal.h"
#include "fsl_clock_manager.h"
#include "fsl_smc_hal.h"
#include "fsl_lptmr_hal.h"

////////////////////////////////////////////////////////////////////////////////
// Definitions
////////////////////////////////////////////////////////////////////////////////
#define TPM_INSTANCE   1      /* use TPM1 CH1 for this case */
#define TPM_CHANNEL    1      /**/
#define TPM_MODULE     (TPM1)   /**/
#define TPM_CnV        TPM1_C1V /**/
#define TPM_PULSE_CNT         8u  /*! The size of pulse to send */
#define TPM_DUTY_CHANGE       1    /*! Whether change the duty cycle for those pulses */
#define TPM_PWM               0x28 /*! High true PWM mode */
#define TPM_CPWM              0    /*! edge or central PWM mode */
#define DMA_CHANNEL           0    /*! DMA Channel to use */
#define DMAMUX_CHANNEL        0    /*! DMA Channel to use */

#define LPTMR_TRIGGER         1    /* Use LPTMR trigger TPM or not, 1 to enable*/
#define LPTMR_INSTANCE        1    /* use LPTMR to trigger TPM */
#define LPTMR_TIMERVALUE   1000    /* LPTMR trigger TPM every 1 second*/
////////////////////////////////////////////////////////////////////////////////
// Variables
////////////////////////////////////////////////////////////////////////////////

// Source data can change TPM pulse duty, move to TPM CnV by DMA if TPM_DUTY_CHANGE is 1
uint16_t  srcAddr[16] =
{100,200,300,400,500,600,700,800,900,1000,1100,1200,1300,1400,1500,1600};
// Destination address in ram, dummy for non duty-change mode
uint16_t destAddr[TPM_PULSE_CNT] = {0};

volatile uint32_t isrFlag = 1;




////////////////////////////////////////////////////////////////////////////////
// Code
////////////////////////////////////////////////////////////////////////////////

/*!
 * @brief DMA callback
 */
void DMA_user_isr(void)
{
    uint32_t status;
    status = DMA_RD_DSR_BCR(DMA0, DMA_CHANNEL);

    if ((status&DMA_DSR_BCR_BED_MASK) || (status&DMA_DSR_BCR_BES_MASK) ||
(status&DMA_DSR_BCR_CE_MASK))
    {
        PRINTF("DMA transfer ERROR, status = 0x%04x\r\n", status);
        DMA_HAL_ClearStatus(DMA0, DMA_CHANNEL);
    }
    if (status&DMA_DSR_BCR_DONE_MASK)
    {
        DMA_HAL_ClearStatus(DMA0, DMA_CHANNEL);
    }

    //enable TPM channel interrupt to finish last whole PWM cycle
    TPM_HAL_EnableChnInt(TPM_MODULE, TPM_CHANNEL);
    TPM_WR_CnSC_DMA(TPM_MODULE, TPM_CHANNEL, false);

}

void TPM1_IRQHandler()
{
    TPM_HAL_ClearChnInt(TPM_MODULE, TPM_CHANNEL);
```

```
    //set TPM clock to none to disable TPM
    TPM_HAL_SetClockMode(TPM_MODULE, kTpmClockSourceNoneClk);
    TPM_WR_CNT(TPM_MODULE, 0);
    TPM_HAL_DisableChnInt(TPM_MODULE, TPM_CHANNEL);
    TPM_WR_CnSC_DMA(TPM_MODULE, TPM_CHANNEL, true);
    TPM_HAL_SetChnCountVal(TPM_MODULE, TPM_CHANNEL, 0);
    PRINTF("Wakeup from VLPS mode by TPM1\r\n");
    //Reset DMA settings
    DMA_HAL_SetDmaRequestCmd(DMA0, DMA_CHANNEL, true);
    DMA_HAL_SetSourceAddr(DMA0, DMA_CHANNEL, (uint32_t)&srcAddr[0]);
    DMA_HAL_SetTransferCount(DMA0, DMA_CHANNEL, TPM_PULSE_CNT*(sizeof(srcAddr[0])));
    // Set the TPM clock againg
    TPM_HAL_SetClockMode(TPM_MODULE, kTpmClockSourceModuleClk);
}
/*!
 * @brief Use TPM in PWM mode
 *
 */
int main(void)
{
    uint32_t i;
    smc_power_mode_config_t powerModeConfig;



    // Init hardware
    hardware_init();

    // Enable DMA clock

 CLOCK_SYS_EnableDmaClock(DMA_IDX);

 // Enable DMAMUX clock
    CLOCK_SYS_EnableDmamuxClock(DMAMUX0_IDX);
    // Enable TPM clock
    CLOCK_SYS_EnableTpmClock(TPM_INSTANCE);

    // Fill zero to dest buffer
    for (i = 0; i < TPM_PULSE_CNT ; i ++)
    {
        destAddr[i] = 0;
    }

    /* PTA13 TPM1 channel 1 */
    PORT_HAL_SetMuxMode(PORTA, 13u,kPortMuxAlt3);

    // DMAMUX configuration
    DMAMUX_HAL_SetTriggerSource(DMAMUX0, DMAMUX_CHANNEL, kDmaRequestMux0TPM1Channel1);
    DMAMUX_HAL_SetChannelCmd(DMAMUX0, DMAMUX_CHANNEL, true);
    // DMA configuration
    DMA_HAL_SetAutoAlignCmd(DMA0, DMA_CHANNEL, false);
    DMA_HAL_SetCycleStealCmd(DMA0, DMA_CHANNEL, true);
    DMA_HAL_SetAsyncDmaRequestCmd(DMA0, DMA_CHANNEL, true);
    DMA_HAL_SetDisableRequestAfterDoneCmd(DMA0, DMA_CHANNEL, true);
    DMA_HAL_SetIntCmd(DMA0, DMA_CHANNEL, true);
    DMA_HAL_SetSourceAddr(DMA0, DMA_CHANNEL, (uint32_t)&srcAddr[0]);
#if (TPM_DUTY_CHANGE)
    DMA_HAL_SetDestAddr(DMA0, DMA_CHANNEL, (uint32_t)&TPM_CnV);
#else
    DMA_HAL_SetDestAddr(DMA0, DMA_CHANNEL, (uint32_t)&destAddr[0]);
#endif
    DMA_HAL_SetSourceModulo(DMA0, DMA_CHANNEL, kDmaModuloDisable);
    DMA_HAL_SetDestModulo(DMA0, DMA_CHANNEL, kDmaModuloDisable);
    DMA_HAL_SetSourceTransferSize(DMA0, DMA_CHANNEL, kDmaTransfersize16bits);
    DMA_HAL_SetDestTransferSize(DMA0, DMA_CHANNEL, kDmaTransfersize16bits);
    DMA_HAL_SetTransferCount(DMA0, DMA_CHANNEL, TPM_PULSE_CNT*(sizeof(srcAddr[0])));
    DMA_HAL_SetSourceIncrementCmd(DMA0, DMA_CHANNEL, true);
#if (TPM_DUTY_CHANGE)
    DMA_HAL_SetDestIncrementCmd(DMA0, DMA_CHANNEL, false);
#else
```

```
    DMA_HAL_SetDestIncrementCmd(DMA0, DMA_CHANNEL, true);
#endif
    //Enable external peripherial request
    DMA_HAL_SetDmaRequestCmd(DMA0, DMA_CHANNEL, true);

    INT_SYS_EnableIRQ(DMA0_IRQn);


INT_SYS_EnableIRQ(TPM1_IRQn);

    // Init PWM configuration.
    // Set clock for TPM.
    CLOCK_SYS_SetTpmSrc(TPM_INSTANCE, kClockTpmSrcMcgIrClk);
    //  CLOCK_SYS_SetTpmSrc(TPM_INSTANCE, kClockTpmSrcIrc48M);
    CLOCK_HAL_SetLircSelMode(MCG, kMcgliteLircSel2M);
    CLOCK_HAL_SetLircCmd(MCG, true);
    CLOCK_HAL_SetLircStopCmd(MCG, true);
 TPM_HAL_SetChnMsnbaElsnbaVal(TPM_MODULE, TPM_CHANNEL, TPM_PWM);
    TPM_HAL_SetCpwms(TPM_MODULE, 0);
    TPM_HAL_SetMod(TPM_MODULE, 2000);
    TPM_HAL_SetChnCountVal(TPM_MODULE, TPM_CHANNEL, 0);
    TPM_WR_CnSC_DMA(TPM_MODULE, TPM_CHANNEL, true);
#if LPTMR_TRIGGER
    // Enable LPTMR clock
    CLOCK_SYS_EnableLptmrClock(LPTMR_INSTANCE);
    //Configure TPM in trigger mode
    TPM_HAL_SetTriggerSrc(TPM_MODULE, kTpmTrigSel14);
    TPM_HAL_SetTriggerMode(TPM_MODULE, true);
    //init LPTMR in timer mode with LPO
    LPTMR_HAL_SetCompareValue(LPTMR0, LPTMR_TIMERVALUE);
    LPTMR_WR_PSR(LPTMR0, kLptmrPrescalerClock1|LPTMR_PSR_PBYP_MASK);
    //enable LTPMR to trigger TPM
    LPTMR_HAL_Enable(LPTMR0);
#endif
    // Set the TPM clock
    TPM_HAL_SetClockMode(TPM_MODULE, kTpmClockSourceModuleClk);
    // Set VLPS mode to enter
    powerModeConfig.powerModeName = kPowerModeVlps;
// Starting channel
    PRINTF("\r\nStarting Pulse generation ...\r\n");
    while(1)
    {
        SMC_HAL_SetMode(SMC, &powerModeConfig);
    }
}
////////////////////////////////////////////////////////////////////////////////
// EOF
////////////////////////////////////////////////////////////////////////////////
```

Fsl_dma_irq.c

```
#include "fsl_dma_driver.h"

extern void DMA_user_isr(void);
/*******************************************************************************
 * Code
 ******************************************************************************/

/* DMA IRQ handler with the same name in startup code*/
void DMA0_IRQHandler(void)
{



    DMA_user_isr();
}

/* DMA IRQ handler with the same name in startup code*/
void DMA1_IRQHandler(void)
{
```

```
    DMA_DRV_IRQhandler(1);
}

/* DMA IRQ handler with the same name in startup code*/

void DMA2_IRQHandler(void)
{
   DMA_DRV_IRQhandler(2);
}

/* DMA IRQ handler with the same name in startup code*/
void DMA3_IRQHandler(void)
{
    DMA_DRV_IRQhandler(3);
}

/*******************************************************************************
 * EOF
```

6 参考资料

下列参考资料包含 Kinetis MCU 中 TPM 和异步 DMA 特性的更多信息。您可以在 www.freescale.com/Kinetis 上选择一个器件来查看具体的参考手册、数据手册或勘误表，然后选择感兴趣的系列以了解更多信息。可以在 freescale.com/ksdk 上找到最新的 SDK 安装程序。

- ***MCU*** 参考手册：参考手册的“芯片配置”章节包含特定 MCU 的设置详情，并对每个 MCU 的复位和功耗管理特性进行了详细说明。
- ***MCU*** 数据手册：数据手册包含所有 MCU 规格，包括时钟频率、低功耗模式的功耗预期值和更多内容。
- ***MCU*** 勘误表：器件勘误表标识因 MCU 问题而未能实现的功能和/或规格。大多数问题都有解决办法。
- 《在 ***Kinetis L*** 系列上使用异步 ***DMA*** 特性》(***AN4631***)：在 Kinetis L 系列上使用 DMA 特性，最新版本的 Kinetis K 系列现在也支持该特性。
- 《***Kinetis L*** 系列的功耗管理》(***AN5088***)：如何使用 Kinetis L 系列的低功耗模式。

7 修订历史记录

下表汇总了对本文档所做的修改。

表 1. 修订历史记录

修订版本号	日期	重要改动
0	2015 年 4 月	初始版本

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

本文档中的信息仅供系统和软件实施方使用 Freescale 产品。 本文并未明示或者暗示授予利用本文档信息进行设计或者加工集成电路的版权许可。 Freescale 保留对此处任何产品进行更改的权利，恕不另行通知。

Freescale 对其产品在任何特定用途方面的适用性不做任何担保、表示或保证，也不承担因为应用程序或者使用产品或电路所产生的任何责任，明确拒绝承担包括但不局限于后果性的或附带性的损害在内的所有责任。 Freescale 的数据表和/或规格中所提供的“典型”参数在不同应用中可能并且确实不同，实际性能会随时间而有所变化。 所有运行参数，包括“经典值”在内，必须经由客户的技术专家对每个客户的应用程序进行验证。 Freescale 未转让与其专利权及其他权利相关的许可。 Freescale 销售产品时遵循以下网址中包含的标准销售条款和条件：freescale.com/SalesTermsandConditions.

Document Number AN5121
Revision 0, 04/2015