Freescale Semiconductor, Inc.
Application Note

Document Number: AN5121
Rev. 0, 04/2015

Generating a Fixed Number of PWM Pulses Using TPM and DMA

1 Introduction

The Timer / PWM Module (TPM) is based on a simple timer used for many years on Freescale's 8-bit microcontrollers (MCUs). The TPM used in Kinetis MCUs extends the functionality to support operation in low power modes by setting the counter and comparing and capturing registers from an asynchronous clock that can remain functional in low power modes.

This use case utilizes the TPM in DMA mode to generate a fixed number of PWM pulses. It is also possible to change the duty cycle of these pulses using the DMA controller. The main idea for this case is to use DMA transfer Byte Count Register DMA_DSR_BCRn [BCR] to:

1. Count the pulses that are generated.
2. Stop the TPM module DMA request at DMA transfer done interrupt.
3. Enable the TPM channel interrupt to complete the last PWM pulse generation.

At the TPM channel interrupt routine, you must re-configure the TPM / DMA to generate more pulses. Because TPM has asynchronous DMA capability, this can place the MCU into VLPS mode, and asynchronous TPM / DMA interrupt can wake up the MCU from VLPS mode.

Contents

This use case can also act as a reference for FTM used in traditional TPM mode. However, the current FTM used in Kinetis MCUs does not support the asynchronous DMA feature mentioned in this document.

2 TPM asynchronous features

The TPM blocks are clocked from a single TPM clock that can be selected from OSCERCLK, MCGIRCLK or MCGPCLK. The selected source is controlled by SIM_SOPT2 [TPMSRC]. Each TPM also supports an external clock mode (TPM_SC [CMOD]=1x) in which the counter increments after a synchronized (to the selected TPM clock source) rising edge detect of an external clock input. The available external clock (either TPM_CLKIN0 or TPM_CLKIN1) is selected by SIM_SOPT4 [TPMxCLKSEL] control register. These clock sources are all asynchronous and can be optional available down to VLPS mode.

3 Required hardware and software

This document describes the example application based on the Freescale Freedom system. The basic concept can be easily implemented on the customized hardware as well.
The application can be easily set up using the following FRDM system boards:

- FRDM-KL43Z

This use case is very easy to integrate into the SDK 1.2 GA package, please see chapter 5 for details. You can download the full package from freescale.com/ksdk.

4 TPM / DMA configuration

This case uses the TPM channel flag to trigger DMA in cycle steal mode to move data from SRAM or Flash to the TPM channel value register. The TPM channel value will be reloaded after TPM timer overflow, then the duty cycle of PWM pulses are changed in line with the value you have entered into the SRAM or Flash. The DMA transfer Byte Count Register DMA_DSR_BCRn [BCR] will keep counting the PWM pulses which are generated. After a certain number of PWM waveforms are generated, DMA transfer done interrupt occurs, you can then stop the TPM module or enable TPM channel interrupt to complete the last PWM pulse generation. At TPM channel interrupt routine, you must re-configure the TPM / DMA to generate more pulses. Because TPM has asynchronous DMA capability, the MCU can be placed into VLPS mode, and asynchronous TPM / DMA interrupt can wake up the MCU from VLPS mode.

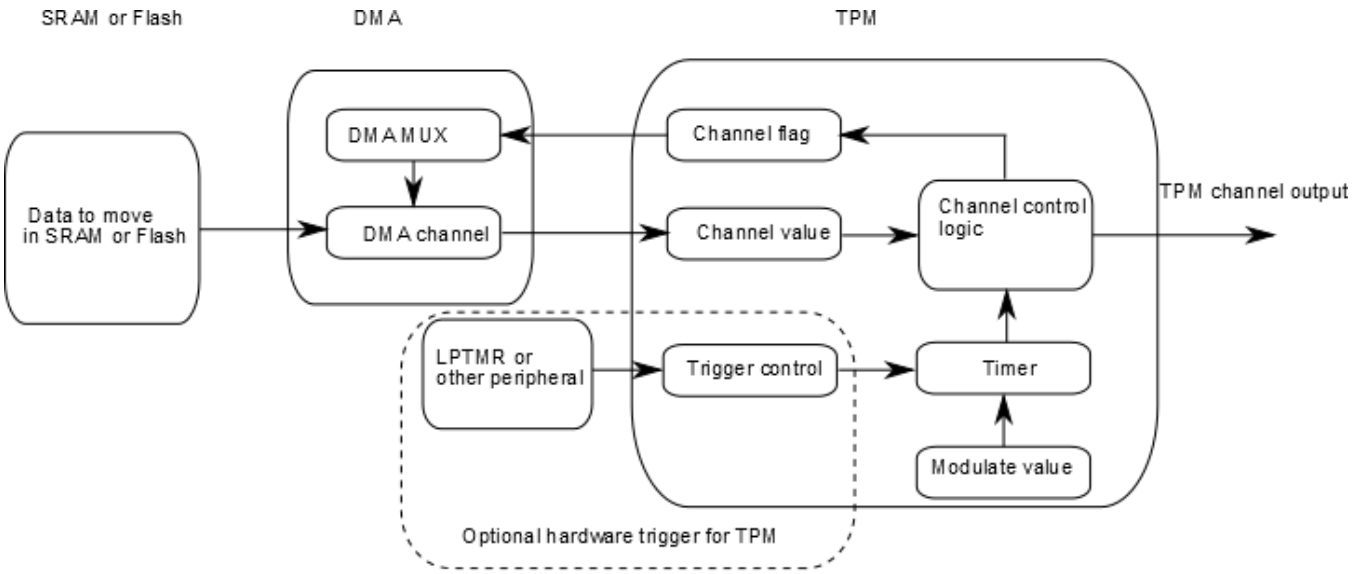

Figure 1 TPM/DMA workflow

The TPM output waveform with 4 pulses is as follows:

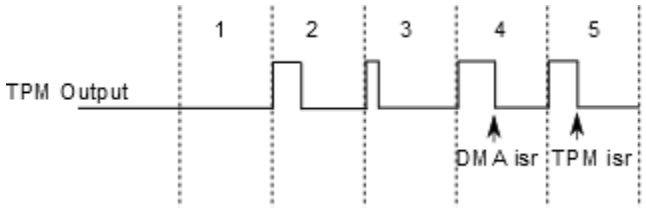

Figure 2 TPM output waveform

4.1 TPM/DMA software workflow

Because the TPM channel value can only reload after TPM timer overflow during EPWM mode, there will be an extra TPM timer period prior to these pulses with the TPM channel value set to 0. Taking 4-pulse generation as an example, the detailed description is as follows:

Stage 1: TPM starts running with initial channel value 0. It can be triggered by hardware or software, DMA BCR minus 1, no pulse output in this stage due to the channel value being 0.

Stage 2: DMA moves data to TPM channel value register at falling edge of PWM pulse, DMA BCR minus 1.

Stage 3: As per stage 2.

Stage 4: Like stage 3 and DMA BCR reach 0, DMA interrupt enter. Enable TPM channel interrupt to complete last TPM waveform generation.

Stage 5: TPM channel interrupt at PWM falling edge, set TPM clock to none, reset TPM counter, disable TPM channel interrupt. Re-initialize DMA and TPM, enable TPM clock to run.

Note:

If you want to only use DMA interrupt, and not TPM interrupt, then the first pulse TPM channel value must be entered via the software at stage 1 and 5. Meanwhile, in stage 4, DMA interrupt must re-initialize the DMA and TPM modules to run. You must ensure that the last PWM cycle correctly generates the pulse. This method is not supported in TPM hardware trigger mode.

Both asynchronous DMA and TPM interrupt can wake up the MCU from VLPS mode.

If you do not want to change the PWM duty cycle, you must set the DMA transfer destination to a dummy value space in SRAM.

4.2 TPM configuration

TPM is configured in edge-align PWM mode, channel flag trigger DMA transfer. The detail of the TPM configuration is as follows:

- Status and Control (TPMx_SC).
  - CPWMS: Select EPWM, set to 0.
  - CMOD: Select TPM counter clock – 1.
- Modulo (TPMx_MOD)
  - MOD: Select timer overflow period.
- Channel (n) Status and Control (TPMx_CnSC)
  - MSB MSA ELSB ELSA: EPWM, high true pulse mode – 1010.
  - DMA: Enable channel flag DMA request - 1.
- Channel (n) Value (TPMx_CnV)
  - VAL: PWM duty cycle, initial value 0.
- Configuration (TPMx_CONF)
  - TRGSEL: Select the trigger source for TPM if using other peripheral to trigger TPM.
  - TRGSRC: Select the trigger source type, externally by other peripheral – 0.
  - TRGPOL: Select trigger polarity, high – 0.
  - CPOT: Select counter pause or not, never pause at trigger – 0.
  - CROT: Select counter reload or not, never reload at trigger – 0.
  - CSOO: Select counter stop on overflow or not, never stop at overflow – 0.

- CSOT: Select counter when to run, start to run at trigger event if using hardware trigger mode– 1.

4.3 DMA configuration

DMA channel mux is configured to the TPM channel request source. The DMA controller is set to cycle steal mode to move only one 16-bit data at each TPM request. The detail of the DMA / DMA mux configuration is as follows:

- Channel configuration register (DMAMUXx_CHCFGn)
  - SOURCE: Select the TPM channel flag as the trigger source.
  - ENBL: Enable this DMA channel mux – 1.
- Source Address Register (DMA_SARn)
  - SAR: Select the source data address in SRAM or Flash.
- Destination Address Register (DMA_DARn)
  - DAR: Select the destination data address for TPM channel value register or dummy data in SRAM in case of no changing of PWM duty cycle.
- DMA Status Register / Byte Count Register (DMA_DSR_BCRn)
  - BCR: data to move, means PWM pulses to generate. Because it is 16-bit long, the BCR value is number of pulses x 2.
- DMA Control Register (DMA_DCRn)
  - EINT: Enable DMA done interrupt – 1.
  - ERQ: Enable peripheral DMA request – 1.
  - CS: Enable cycle steal – 1.
  - EADREQ: Enable async DMA -1.
  - SINC: Select source address increased – 1.
  - SSIZE: Select source data size 16-bit – 10.
  - DINC: Select destination address is not increased ‑ 0.
  - DSIZE: Select destination data size 16-bit – 10.
  - D_REQ: Disable peripheral DMA request after DMA done – 1.

5 Code listing

This use case mainly has two files. You can use the files in any SDK 1.2 KL43 example, and replace the existing files. For example, you can open KL43 IAR hello world demo, remove the files via the “source” folder (*main.c* and *fsl_lptmr_irq.c*), and insert these two files there. You must then build and debug this use case. The content of these two files is as follows:

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 References

The references listed below contain additional information regarding TPM and asynchronous DMA features in Kinetis MCUs. You can find a particular reference manual, data sheet or errata report by choosing a device at freescale.com/Kinetis and then selecting the family that is of interest to you in order to find more information. The latest SDK installer can be found at freescale.com/ksdk.

- **MCU Reference Manuals:** The reference manuals contain MCU-specific implementation details in the Chip Configuration chapters and include a detailed description of the Reset and Power Management Features of each MCU.
- **MCU Data Sheet Specifications:** The data sheet includes all of the MCU specifications, including clock rates, low power mode power consumption expectations and so on.
- **Errata for MCUs:** Device errata identify what functionality and/or specification is not being met due to a problem with the MCU. Most of the issues have workarounds.
- **Using the Asynchronous DMA features of the Kinetis L Series (AN4631)**: Using Asynchronous DMA feature on Kinetis L family, additionally the latest version Kinetis K family also support this feature.
- **Power Management for Kinetis L Family (AN5088)**: How to use Kinetis L family low power mode.

7 Revision history

The following table summarizes the changes made to this document.

Table 1: Revision history

Revision number	Date	Substantive changes
0	04/2015	Initial release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN5121
Rev. 0
04/2015