For EE342: Getting PWM Working for the KL43Z

Overview:

This document is intended to help students implement Pulse Width Modulation waveforms using the TPM timers on the FRDM-KL43Z demo boards used in EE342. The TPM timers for this microcontroller are very similar to the TPM timers for the HCS08 family microcontrollers used in EE247 and EGR222. However, the nature of the interface as seen from the programmer's perspective is very different. For the HCS08 family TPM modules, the programmer typically refers to the microprocessor manual and then programs by directly manipulating the hardware registers directly associated with the TPM timers. For the KL43, the programmer does not interact with the hardware except through functions, in some cased well removed from hardware specifics by defined constants, layers of function calls, and other forms of abstraction. That is the nature of programming in the modern world. This document is intended to help students make this transition to the more abstract function based use of hardware features, the TPM timer in particular. The examples and code in this document were developed on or for the "MCUXpresso" Integrated Development Environment (IDE).

The Demo:

MCUXpresso has an associated "Software Development Kit (SDK) with numerous example implementations of different functional elements, including a whole set of driver examples. Each driver example focuses on one unit. The TPM timers are very versatile, capable of a number of different functions. This document focuses on Pulse Width Modulation (PWM). Dy downloading the simplest TPM PWM example, students can see how the TPM module can be used for this purpose with a simple application that can be seen to execute on the FRDM-KL43Z demo boards used for EE342. The demo "main program" is given in Appendix A.

The demo, once compiled, downloaded to the microcontroller and run, has TPM0 (the first of three TPM timers) Channel 5 (the last of six for that TPM) driving the green LED on the demo board at duty cycles ranging from 10% (dim) to 90% (bright). The demo program allows the duty cycle to be changed by typing in characters.

This "simple TPM" demo is very helpful in illustrating the various "#define" statements and function calls needed to define and initialize the TPM for use and to modify the duty cycle. In part, this document walks the reader through this demo program, explaining some of the features and why they are there. Second, these examples form a template for using TPM in another context. For us, that most immediately is the terminal based applications we are using for developing an understanding of the processor, and later we will be able to use TPM for speed controls.

Unlike some demos, and what is in the interest of good organization in a more complex application, the TPM demo program does most of its business inside the "main" file. Many defines and functions are there that would normally be in files such as "board.h" or "board.c" or "pinmux.h" or "pinmux.c". This document will not try to address those organizational issues.

The TPM Timer concept:

The TPM timer is described in the corresponding chapter of the KL43 manual. The diagram below (Figure 29.1 from the manual) is a good overview. Briefly, pulses from a clock (typically the 48MHz system clock are pre-scaled by dividing by some 2^n (1 to 128) and counted in the TPM module 16 bit counter. When the timer counter gets up to the value stored in a

"modulo" register (MOD) it "times out" and sets a flag, and perhaps an interrupt. Meanwhile, the counter is being compared to the "channel values" for each of the active channels. When the numbers match, the channel does something. At the very least, it sets a flag. It may trigger an interrupt. For PWM, it also controls a pin. It can either cause the pin to go low on compare or high on compare. That's what creates the PWM pulse train. (In a "center aligned" mode the counter doesn't return to 0 immediately at the MOD value, but starts counting down. At 0 it goes back to counting up.)

TPM Signal Descriptions

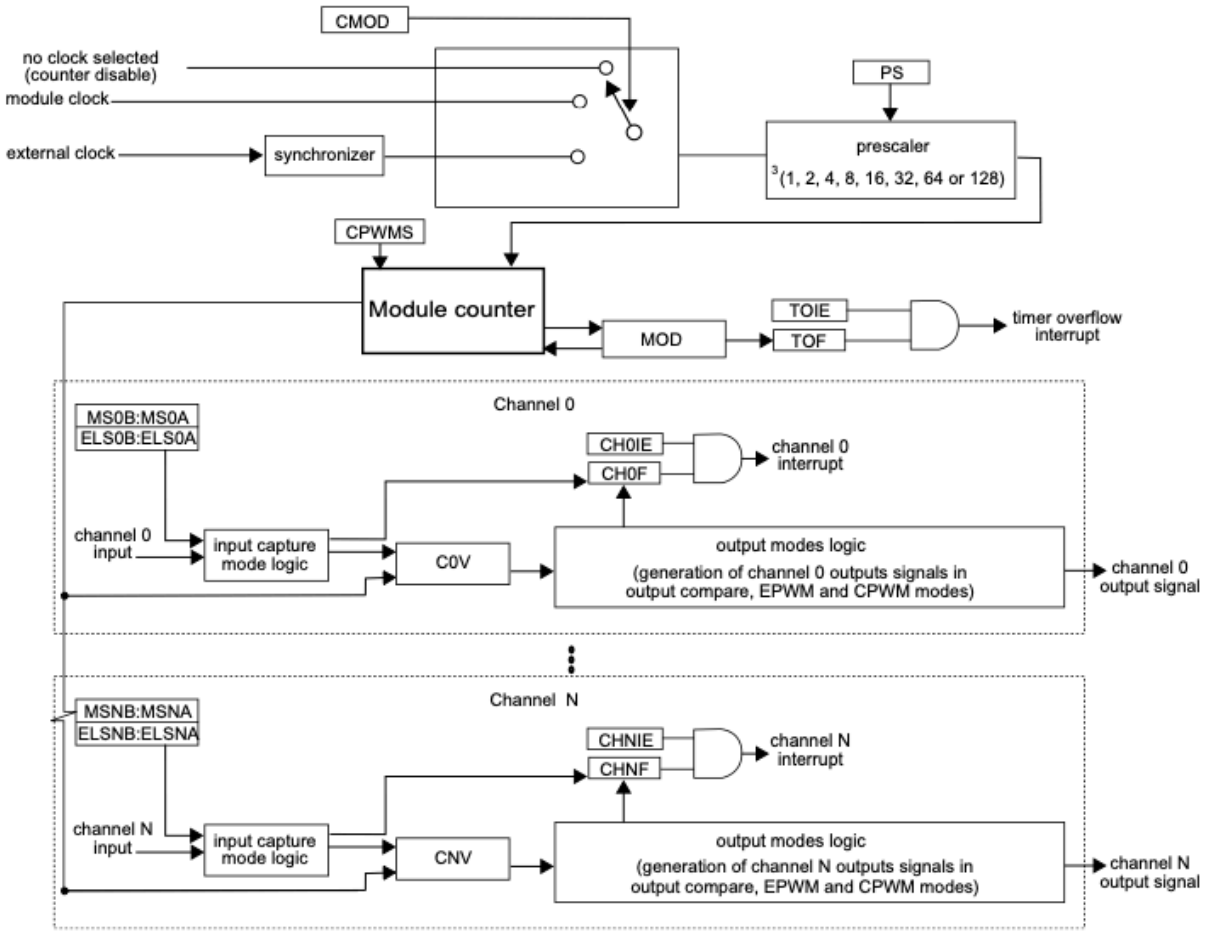


Figure 29-1. TPM block diagram

TPM Timer Definition:

It is assumed a particular application needs to use a TPM Timer to produce a PWM waveform for some practical purpose. One of the first issues that needs to be addressed is, "Which TPM?" And, secondly, "Which Channel(s)?" Put a different way, the question is "Which Pin?" The microcontroller has three TPM's, ten channels, and many pins. But, most of those pins have multiple purposes, and it's important not to use a pin that we might need for something else. For example, two of the pins (PTA1, PTA2) serve as UART0_RX and UART0_TX and so are needed for communication (via serial port) with the user. Were not that

the case, those pins could be used for TMP2_CH0 and TPM2_CH1 respectively. So, PTA1 and PTA2 are not available. Similarly, we can't use PTA4 or PTC3 (pushbuttons). Many pins are needed for the LCD display, some of which are potential TPM outputs.

The TPM and Channel used for the demo program were chosen to use the available hardware, in particular, the green LED which is attached to PTD5. That pin serves as a potential output (or input) for TPM0 Channel 5. (If we wanted to use a different pin, we'd need to find a pin that is available and has an associated TPM channel that we don't need for something else.)

So, before we can do much else, we need to include in our C file (project.c, for whatever our project is) an include for appropriate driver functions (# include "fsl_tpm.h") as well as #defines for which TPM and channel we intend to use, and other associated constants. The following can be copied from the "tpm_simple_pwm.c" project file:

```
#include "fsl_tpm.h"
```

(We should already have #includes for "fsl_debug_console.h", "board.h", and "pin_mux.h".)

```
/* The Flextimer instance */
#define BOARD_TPM_BASEADDR TPM0
/* Interrupt to enable and flag to read; depends on the TPM channel
used */
#define TPM_CHANNEL_INTERRUPT_ENABLE kTPM_Chnl5InterruptEnable
/* Interrupt number and interrupt handler for the TPM instance used */
#define TPM_INTERRUPT_NUMBER TPM0_IRQn
#define TPM_CHANNEL_FLAG kTPM_Chnl5Flag
#define TPM_LED_HANDLER TPM0_IRQHandler
/* Get source clock for TPM driver */
#define TPM_SOURCE_CLOCK CLOCK_GetFreq(kCLOCK_McgIrc48MClk)
```

The first #define above says that we will use TPM0. TPM0 is a pointer to the base address, that is, the starting address where this particular TPM timer is defined. That address is given in "MKL43Z4.h", line 13085, and preceding. The idea is that from here on the programmer doesn't need to know that address, or even that TPM0 is the timer being used. The symbol "BOARD_TPM_BASEADDR" is used to identify which TPM is to be used. Where is the channel to be used? For some reason, that is defined in "board.h" (lines 37 and 38) in this example instead. Notice the redundancy; the TPM is identified as TPM0 there as well. (Board.h also has some other TPM related defines related to the use of the accelerometer that are not applicable to the simple TPM demo application.)

```
#define BOARD_TPM_BASEADDR TPM0
#define BOARD_TPM_CHANNEL 5U
```

The #defines pertaining to the interrupt really are not needed for this example application, which makes no use of the interrupt. One can enable an interrupt for when the TPM counter times out or when the counter reaches the channel value. Those can be useful, but won't be addressed here. One thing of note: It looks like "Channel 15" Interrupt enable and flag are being defined. No! The abbreviation "Chnl" for "Channel" is being used. This is confusing, because the lower case letter "l" (L) looks like the numeral "1" (one). This is a confusion that should be avoided whenever possible. (The capital "O" is also easily confused with the numeral "0" in some fonts.)

The last #define provides a means to query the clock frequency for the TPM timer. When the "constant" TPM_SOURCE_CLOCK is used, what actually happens is a call to the function CLOCK_GetFreq(). The value passed says, "the 48MHz clock." (You'd think that would always be 48MHz, no?)

The demo includes some global variables, of which "updatedDutycycle" is the most important. The initial value of 10U (10 unsigned) means a 10% duty cycle.

```
volatile bool brightnessUp=true;/*Indicate LED is brighter or dimmer*/
volatile uint8_t updatedDutycycle = 10U;
volatile uint8_t getCharValue     = 0U;
```

In the example program, initialization of the TPM is done in the main() function. Two structures are defined, for the TPM timer and one (or more) of its channels. If more than one channel is to be used, each channel structure needs to be a member of an array, one per timer being used. The code immediately following loads up the pwm channel structure with initial data, most importantly identifying which channel, whether it drives its pin active high or active low, and the initial duty cycle. (The definition of TPM_LED_ON_LEVEL should logiacally have been done earlier with the other #define statements, or in "board.h".)

```
tpm_config_t tpmInfo;
tpm_chnl_pwm_signal_param_t tpmParam;
#define TPM_LED_ON_LEVEL kTPM_LowTrue
/* Configure tpm params with frequency 24kHZ */
tpmParam.chnlNumber       = (tpm_chnl_t)BOARD_TPM_CHANNEL;
tpmParam.level            = TPM_LED_ON_LEVEL;
tpmParam.dutyCyclePercent = updatedDutycycle;
```

Somewhere else (following) the clock for the TPM timer needs to be set:

```
/* Select clock source for TPM counter as kCLOCK_McgIrc48MClk */
CLOCK_SetTpmClock(1U);
```

Also, in "board.c" within the function "BOARD_InitPins()" we find an important statement that connects PTD5 (the green LED pin) to the TPM0 channel 5 signal instead of the general purpose I/O (GPIO) signal. (There's also a statement that connects the RED led at PTE31 to channel 4, but we don't want to do that if we want it to be GPIO. This statement supports the accelerometer demo.)

```
/* Port D Clock Gate Control: Clock enabled */
CLOCK_EnableClock(kCLOCK_PortD);
```

(and much later:)

```
/* PORTD5 (pin 62) is configured as TPM0_CH5 */
PORT_SetPinMux(PORTD, 5U, kPORT_MuxAlt4);
/* PORTE31 (pin 19) is configured as TPM0_CH4 */
//PORT_SetPinMux(PORTE, 31U, kPORT_MuxAlt3); /*leave out */
```

There's an issue here. The pin_mux options include 0 (disabled), 1 (gpio, what we've normally used for buttons and LED's, and other options. Those other options are identified as "kPORT_MuxAltn" where n is a small integer. How do we know what n to use? Here we see that the TPM signal is alternative 4 for PTD5 and 3 for PTE31. The description of pin PTD5 on the schematic is: "PTD5/LCD_P45/ADC0_SE6B/SPI1_SCK/UART 2_TX/TPM0_CH5". Counting "PTD5" as one, and counting through the alternatives given, makes TPM0_CH5 6, not 4. For PTE31 "PTE31/TPM0_CH4" is the description and the actual alternative is kPORT_MuxAlt3. So, I don't know where the pin by pin alternate assignments can be found. I did not find them in the manual.

The TPM timer is finally initialized and started up with the following four statements:

```
/* Initialize TPM module */
TPM_GetDefaultConfig(&tpmInfo);
TPM_Init(BOARD_TPM_BASEADDR, &tpmInfo);
TPM_SetupPwm(BOARD_TPM_BASEADDR, &tpmParam, 1U,
      kTPM_CenterAlignedPwm, 24000U, TPM_SOURCE_CLOCK);
TPM_StartTimer(BOARD_TPM_BASEADDR, kTPM_SystemClock);
```

These need some explaining. Recall tpmInfo is the structure used to configure the TPM. The "&" is a "pointer to." So, the call to the "TPM_GetDefaultConfig( )" function loads up the structure with default values (a "vanilla" configuration). We don't modify that, but immediately call "TPM_Init" to make it so for TPM0. The really specific configuration call, to TPM_SetupPwm(), uses the tpmParam structure already defined (channel 5, active low, 10% duty cycle). That the pointer passed (&tpmParam) points to only one channel structure. The PWM frequency desired is 24,000 Hz. The TPM counter clock frequency is TPM_SOURCE_CLOCK, which should have the value 48 MHz. (These two numbers are used to set the clock divisor and modulo register for the TPM timer. This function (located in "fsl_tpm.c") is quite complicated and lengthy. Finally, the TPM timer is turned on by starting the clock. From this point on, the TPM timer is operating. In the demo program, the green LED is dim as it is only on 10% of the time, although at 24KHz the discontinuous nature of its operation is not apparent to the eye.

Changing the PWM Duty Cycle:

In the example demo application, this is done as follows. Given all the initialization stuff, this should be fairly straightforward. It is noteworthy that the TPM is turned off while being updated. (That wasn't necessarily done with the HCS08 TPM's.)

```
/* Disable channel output before updating the dutycycle */
TPM_UpdateChnlEdgeLevelSelect(BOARD_TPM_BASEADDR,
         (tpm_chnl_t)BOARD_TPM_CHANNEL, 0U);
/* Update PWM duty cycle */
TPM_UpdatePwmDutycycle(BOARD_TPM_BASEADDR,
         (tpm_chnl_t)BOARD_TPM_CHANNEL, kTPM_CenterAlignedPwm,
                        updatedDutycycle);
/* Start channel output with updated dutycycle */
```

```
TPM_UpdateChnlEdgeLevelSelect(BOARD_TPM_BASEADDR,
          (tpm_chnl_t)BOARD_TPM_CHANNEL, TPM_LED_ON_LEVEL);
```

Putting it Into another Context:

Most of this code can be simply lifted and put in place inside your application. The "UpdateDutycycle" changes just above might be inside a command callback function "cmd_tpm()" which would set the duty cycle according to a given parameter. There could be changes to the frequency too, although that's not usually done; usually the frequency remains constant. The one remaining issue is how to choose a different pin (rather than PTD5) and how to find the proper "alternative" pin_mux setting other than by trial and error. I have been trying to add a blue LED at PTE29, which would be "TPM0_CH2 " if the correct pin_mux setting could be found. As of this writing I've not yet found the correct mux setting. I also have not seen in the TPM simple demo where the pin is configured. Perhaps simply setting the pinmux is enough to enable it with default settings. As with much else, we are discovering such things as we go.

MCUXpresso “Simple TPM” example, Main project file “tpm_simple_pwm.c”

```
/*
 * Copyright (c) 2015, Freescale Semiconductor, Inc.
 * Copyright 2016-2017 NXP
 * All rights reserved.
 *
 * SPDX-License-Identifier: BSD-3-Clause
 */

#include "fsl_debug_console.h"
#include "board.h"
#include "fsl_tpm.h"

#include "pin_mux.h"
/*******************************************************************************
 * Definitions
 ******************************************************************************/
/* The Flextimer instance */
#define BOARD_TPM_BASEADDR TPM0

/* Interrupt to enable and flag to read; depends on the TPM channel used */
#define TPM_CHANNEL_INTERRUPT_ENABLE kTPM_Chnl5InterruptEnable

/* Interrupt number and interrupt handler for the TPM instance used */
#define TPM_INTERRUPT_NUMBER TPM0_IRQn
#define TPM_CHANNEL_FLAG kTPM_Chnl5Flag
#define TPM_LED_HANDLER TPM0_IRQHandler

/* Get source clock for TPM driver */
#define TPM_SOURCE_CLOCK CLOCK_GetFreq(kCLOCK_McgIrc48MClk)

/*******************************************************************************
 * Prototypes
 ******************************************************************************/
/*!
 * @brief delay a while.
 */
void delay(void);

/*******************************************************************************
 * Variables
 ******************************************************************************/
volatile bool brightnessUp        = true; /* Indicate LED is brighter or
dimmer */
volatile uint8_t updatedDutycycle = 10U;
volatile uint8_t getCharValue     = 0U;

/*******************************************************************************
 * Code
 ******************************************************************************/
/*!
 * @brief Main function
 */
```

```
int main(void)
{
    tpm_config_t tpmInfo;
    tpm_chnl_pwm_signal_param_t tpmParam;

#ifndef TPM_LED_ON_LEVEL
#define TPM_LED_ON_LEVEL kTPM_LowTrue
#endif

    /* Configure tpm params with frequency 24kHZ */
    tpmParam.chnlNumber       = (tpm_chnl_t)BOARD_TPM_CHANNEL;
    tpmParam.level            = TPM_LED_ON_LEVEL;
    tpmParam.dutyCyclePercent = updatedDutycycle;

    /* Board pin, clock, debug console init */
    BOARD_InitPins();
    BOARD_BootClockRUN();
    BOARD_InitDebugConsole();
    /* Select the clock source for the TPM counter as kCLOCK_McgIrc48MClk */
    CLOCK_SetTpmClock(1U);

    /* Print a note to terminal */
    PRINTF("\r\nTPM example to output center-aligned PWM signal\r\n");
    PRINTF(
        "\r\nIf an LED is connected to the TPM pin, you will see a change in
LED brightness if you enter different "
        "values");
    PRINTF("\r\nIf no LED is connected to the TPM pin, then probe the signal
using an oscilloscope");

    TPM_GetDefaultConfig(&tpmInfo);
    /* Initialize TPM module */
    TPM_Init(BOARD_TPM_BASEADDR, &tpmInfo);

    TPM_SetupPwm(BOARD_TPM_BASEADDR, &tpmParam, 1U, kTPM_CenterAlignedPwm,
24000U, TPM_SOURCE_CLOCK);

    TPM_StartTimer(BOARD_TPM_BASEADDR, kTPM_SystemClock);

    while (1)
    {
        do
        {
            PRINTF("\r\nPlease enter a value to update the Duty cycle:\r\n");
            PRINTF("Note: The range of value is 0 to 9.\r\n");
            PRINTF("For example: If enter '5', the duty cycle will be set to
50 percent.\r\n");
            PRINTF("Value:");
            getCharValue = GETCHAR() - 0x30U;
            PRINTF("%d", getCharValue);
            PRINTF("\r\n");
        } while (getCharValue > 9U);

        updatedDutycycle = getCharValue * 10U;
```

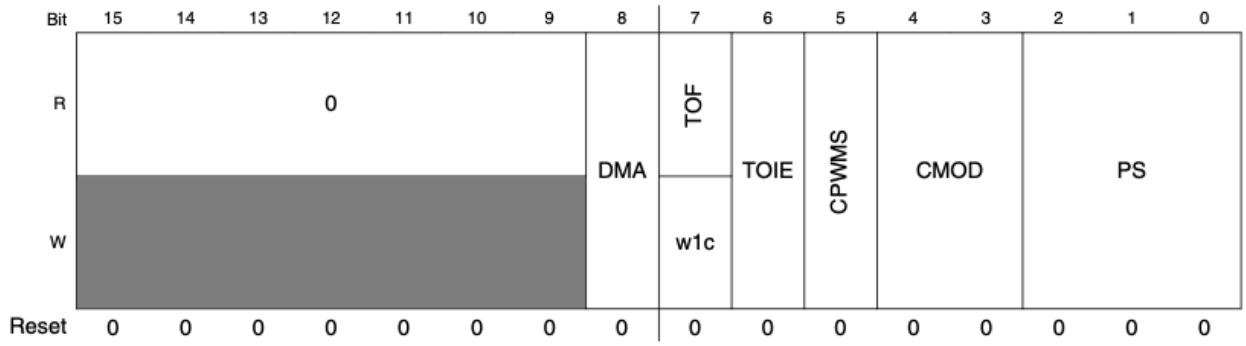
```
        /* Disable channel output before updating the dutycycle */
        TPM_UpdateChnlEdgeLevelSelect(BOARD_TPM_BASEADDR,
(tpm_chnl_t)BOARD_TPM_CHANNEL, 0U);

        /* Update PWM duty cycle */
        TPM_UpdatePwmDutycycle(BOARD_TPM_BASEADDR,
(tpm_chnl_t)BOARD_TPM_CHANNEL, kTPM_CenterAlignedPwm,
                               updatedDutycycle);

        /* Start channel output with updated dutycycle */
        TPM_UpdateChnlEdgeLevelSelect(BOARD_TPM_BASEADDR,
(tpm_chnl_t)BOARD_TPM_CHANNEL, TPM_LED_ON_LEVEL);

        PRINTF("The duty cycle was successfully updated!\r\n");
    }
}
```

Appendix B Register definitions and other information (from the Manual)



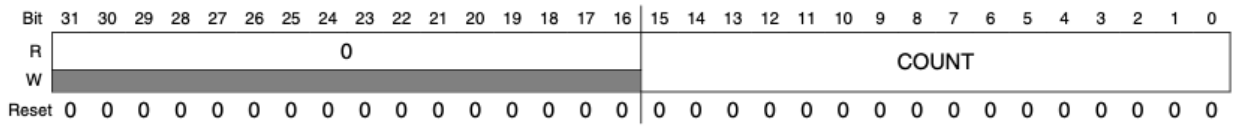
TPMx_SC field descriptions

Field	Description
31–9 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
8 DMA	DMA Enable Enables DMA transfers for the overflow flag. 0 Disables DMA transfers. 1 Enables DMA transfers.
7 TOF	Timer Overflow Flag Set by hardware when the TPM counter equals the value in the MOD register and increments. Writing a 1 to TOF clears it. Writing a 0 to TOF has no effect. If another TPM overflow occurs between the flag setting and the flag clearing, the write operation has no effect; therefore, TOF remains set indicating another overflow has occurred. In this case a TOF interrupt request is not lost due to a delay in clearing the previous TOF.

Notice that all of these registers are actually 32 bits, but the top 16 bits are ignored. The addresses are given in some cases. You should be using symbolic "base address" constants like "TPM0" rather than actual addresses, which are defined inside the "KL43Z4.h" file that is included in all projects.

	0 TPM counter has not overflowed. 1 TPM counter has overflowed.
6 TOIE	Timer Overflow Interrupt Enable Enables TPM overflow interrupts. 0 Disable TOF interrupts. Use software polling or DMA request. 1 Enable TOF interrupts. An interrupt is generated when TOF equals one.
5 CPWMS	Center-Aligned PWM Select Selects CPWM mode. This mode configures the TPM to operate in up-down counting mode. This field is write protected. It can be written only when the counter is disabled. 0 TPM counter operates in up counting mode. 1 TPM counter operates in up-down counting mode.
4–3 CMOD	Clock Mode Selection Selects the TPM counter clock modes. When disabling the counter, this field remain set until acknolwedged in the TPM clock domain. 00 TPM counter is disabled 01 TPM counter increments on every TPM counter clock 10 TPM counter increments on rising edge of TPM_EXTCLK synchronized to the TPM counter clock 11 Reserved.
PS	Prescale Factor Selection Selects one of 8 division factors for the clock mode selected by CMOD. This field is write protected. It can be written only when the counter is disabled. 000 Divide by 1 001 Divide by 2 010 Divide by 4 011 Divide by 8 100 Divide by 16 101 Divide by 32 110 Divide by 64 111 Divide by 128

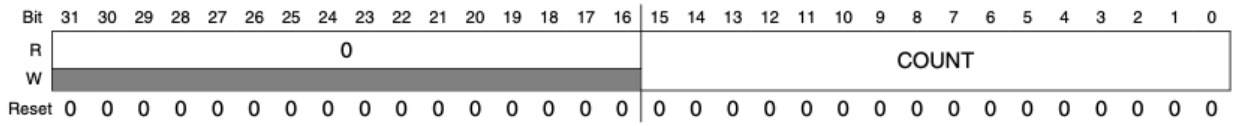
Address: Base address + 4h offset



TPMx_CNT field descriptions

Field	Description
31–16 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
COUNT	Counter value

Address: Base address + 4h offset



TPMx_CNT field descriptions

Field	Description
31–16 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
COUNT	Counter value

For setting the channel registers (TPMnCHm):

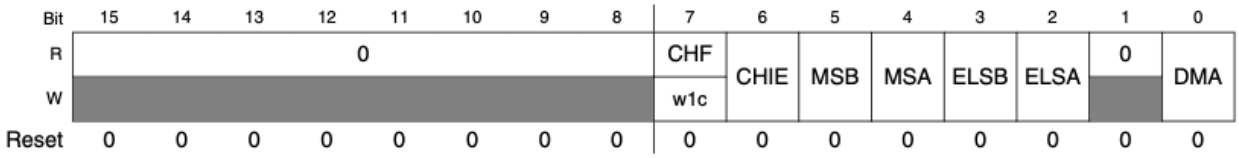


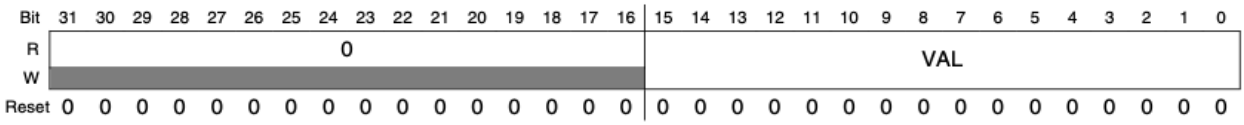
Table 29-5. Mode, Edge, and Level Selection

CPWMS	MSnB:MSnA	ELSnB:ELSnA	Mode	Configuration
X	00	00	None	Channel disabled
X	01	00	Software compare	Pin not used for TPM
0	00	01	Input capture	Capture on Rising Edge Only
		10		Capture on Falling Edge Only
		11		Capture on Rising or Falling Edge
	01	01	Output compare	Toggle Output on match
		10		Clear Output on match
		11		Set Output on match
	10	10	Edge-aligned PWM	High-true pulses (clear Output on match, set Output on reload)
		X1		Low-true pulses (set Output on match, clear Output on reload)
	11	10	Output compare	Pulse Output low on match
		01		Pulse Output high on match
1	10	10	Center-aligned PWM	High-true pulses (clear Output on match-up, set Output on match-down)
		01		Low-true pulses (set Output on match-up, clear Output on match-down)

TPMx_CnSC field descriptions

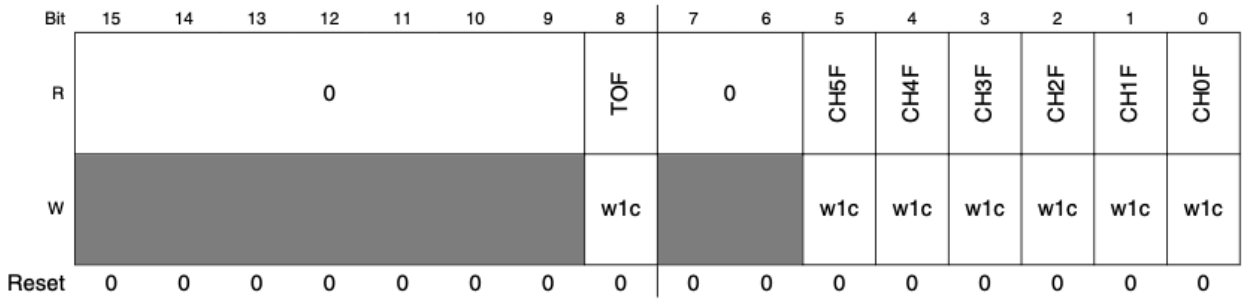
Field	Description
31–8 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
7 CHF	Channel Flag Set by hardware when an event occurs on the channel. CHF is cleared by writing a 1 to the CHF bit. Writing a 0 to CHF has no effect. If another event occurs between the CHF sets and the write operation, the write operation has no effect; therefore, CHF remains set indicating another event has occurred. In this case a CHF interrupt request is not lost due to the delay in clearing the previous CHF. 0 No channel event has occurred. 1 A channel event has occurred.
6 CHIE	Channel Interrupt Enable Enables channel interrupts. 0 Disable channel interrupts. 1 Enable channel interrupts.
5 MSB	Channel Mode Select Used for further selections in the channel logic. Its functionality is dependent on the channel mode. When a channel is disabled, this field will not change state until acknowledged in the TPM counter clock domain.
4 MSA	Channel Mode Select Used for further selections in the channel logic. Its functionality is dependent on the channel mode. When a channel is disabled, this field will not change state until acknowledged in the TPM counter clock domain.
3 ELSB	Edge or Level Select The functionality of ELSB and ELSA depends on the channel mode. When a channel is disabled, this field will not change state until acknowledged in the TPM counter clock domain.
2 ELSA	Edge or Level Select The functionality of ELSB and ELSA depends on the channel mode. When a channel is disabled, this field will not change state until acknowledged in the TPM counter clock domain.
0 DMA	DMA Enable Enables DMA transfers for the channel. 0 Disable DMA transfers. 1 Enable DMA transfers.

Address: Base address + 10h offset + (8d × i), where i=0d to 5d



TPMx_CnV field descriptions

Field	Description
31–16 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
VAL	Channel Value Captured TPM counter value of the input modes or the match value for the output modes. This field must be written with single 16-bit or 32-bit access.



TPMx_STATUS field descriptions

Field	Description
31–9 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
8 TOF	Timer Overflow Flag See register description 0 TPM counter has not overflowed. 1 TPM counter has overflowed.
7–6 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
5 CH5F	Channel 5 Flag See the register description. 0 No channel event has occurred. 1 A channel event has occurred.
4 CH4F	Channel 4 Flag See the register description. 0 No channel event has occurred. 1 A channel event has occurred.

(and other channel flags similar)

Polarity settings:

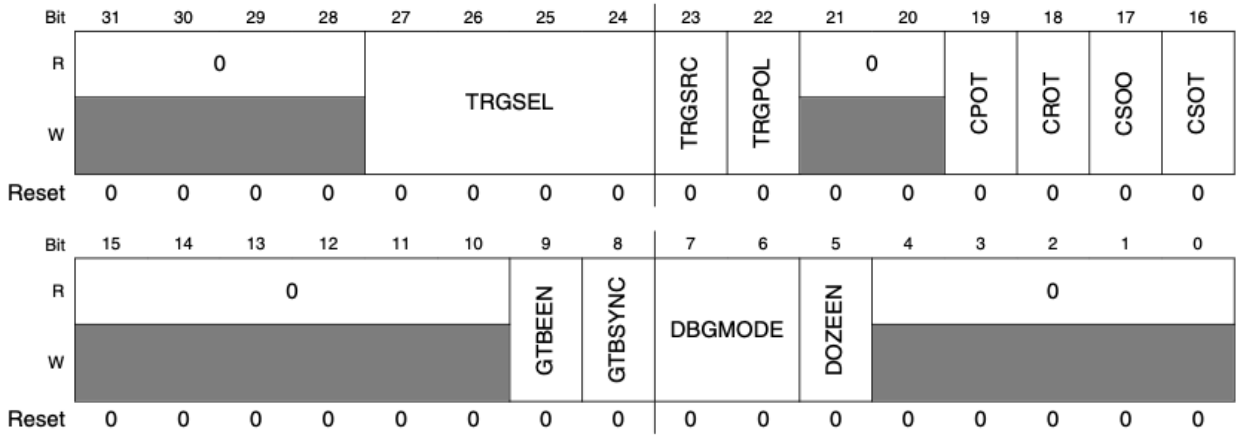
TPMx_POL field descriptions

Field	Description
31–6 Reserved	This field is reserved. This read-only field is reserved and always has the value 0.
5 POL5	Channel 5 Polarity 0 The channel polarity is active high. 1 The channel polarity is active low.
4 POL4	Channel 4 Polarity 0 The channel polarity is active high 1 The channel polarity is active low.

29.4.8 Configuration (TPMx_CONF)

This register selects the behavior in debug and wait modes and the use of an external global time base.

Address: Base address + 84h offset



TPMx_CONF field descriptions

(The details are skipped since this register is primarily concerned with debugging and defaults are usually what you want, but the manual does have the rest of the description of this register and others.)

About interrupts:

The TPM has multiple sources of interrupt. However, these sources are OR'd together to generate a single interrupt request to the interrupt controller. When an TPM interrupt occurs, read the TPM status registers to determine the exact interrupt source.