Customer training workshop:
Device Configurator_Timer configuration

TRAVEO™ T2G CYT4BF series Microcontroller Training
V1.0.0 2022-12

Scope of work

- This document helps application developers understand how to use the Timer configuration of the Device Configurator as part of creating a ModusToolbox™ (MTB) application
  - The Device Configurator is part of a collection of tools included with the MTB software. It provides a GUI to configure the target device.

- ModusToolbox™ tools package version
  - 3.0.0
- Device Configurator version
  - 4.0
- Device
  - The TRAVEO™ T2G CYT4BFBCH device is used in this code example.
- Board
  - The TRAVEO™ T2G KIT_T2G-B-H_EVK board is used for testing.

Introduction

- **TCPWM has the following features:**
  - Supports up to four counter groups (device-specific)
  - Each counter group consists of up to 256 counters (counter group-specific)
  - Each counter can run in one of the following seven function modes:
    - Timer-counter with compare
    - Timer-counter with capture
    - Quadrature decoding
    - Pulse-width modulation/stepper motor control (SMC) for pointer instruments
    - PWM with dead time/three-phase motor control (Brushless-DC, BLDC)
    - Pseudo-random PWM
    - Shift register mode
  - 16-bit or 32-bit counters (counter group-specific)
  - Up, down, and up/down counting modes
  - Clock prescaling (division by 1, 2, 4, ... 64, 128)

Introduction (contd.)

TCPWM has the following features:

- Up to two capture and compare functions (counter group-specific)
- Double buffering of all compare/capture and period registers
- Two output trigger signals for each counter to indicate underflow, overflow, and capture/compare events; they can also directly connects with the line output signal
- Supports interrupt on:
  - Terminal count - Depends on the mode; typically occurs on overflow or underflow
  - Capture/compare - The count has been captured in the capture registers or the counter value equals the value in the compare register
- Line out selection feature for stepper motor application including two complementary output lines with dead time insertion
- Selectable start, reload, stop, count, and two capture event signals for each TCPWM with the rising edge, falling edge, both edges, and level trigger options
- Each counter with up to 254 (device-specific) synchronized input trigger signals and two constant input signals: '0' and '1'.

Introduction (contd.)

- **TCPWM has the following features:**
  - Two types of input triggers for each counter:
    - General-purpose triggers used by all counters
    - One-to-one triggers for specific counter
  - Synchronous operation of multiple counters
  - Debug mode support

Introduction (contd.)

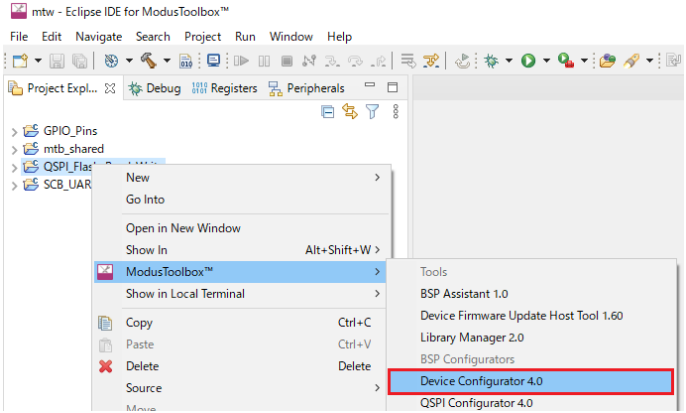
- **Timer configuration in Device Configurator:**
  - Function mode (PWM, timer counter compare/capture, Quadrature decoder, Shift register)
  - Input clock prescaler
  - Alignment (left, right, center, asymmetric)
  - Run mode (Continuous/One Shot)
  - Count direction (Up, Down, Up and Down)
  - Select Run mode (Continuous/One Shot)
  - Programmable period, compare, and capture register
  - Interrupt generation
  - Start, Reload, Stop, Swap (Capture), and Count Inputs
  - Event generation
  - GPIO connection

Launch the Device Configurator

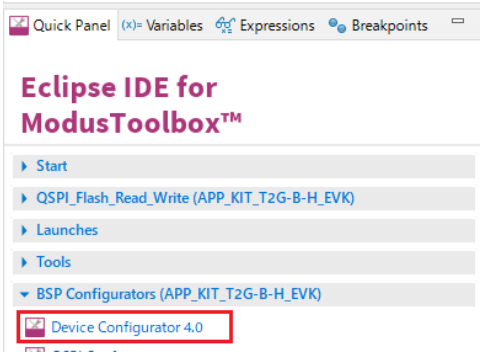
› From Eclipse IDE

You can launch the Device Configurator by either of the following methods:

a) Right-click on the project in the Project Explorer and select **ModusToolbox™** > **Device Configurator** <version>



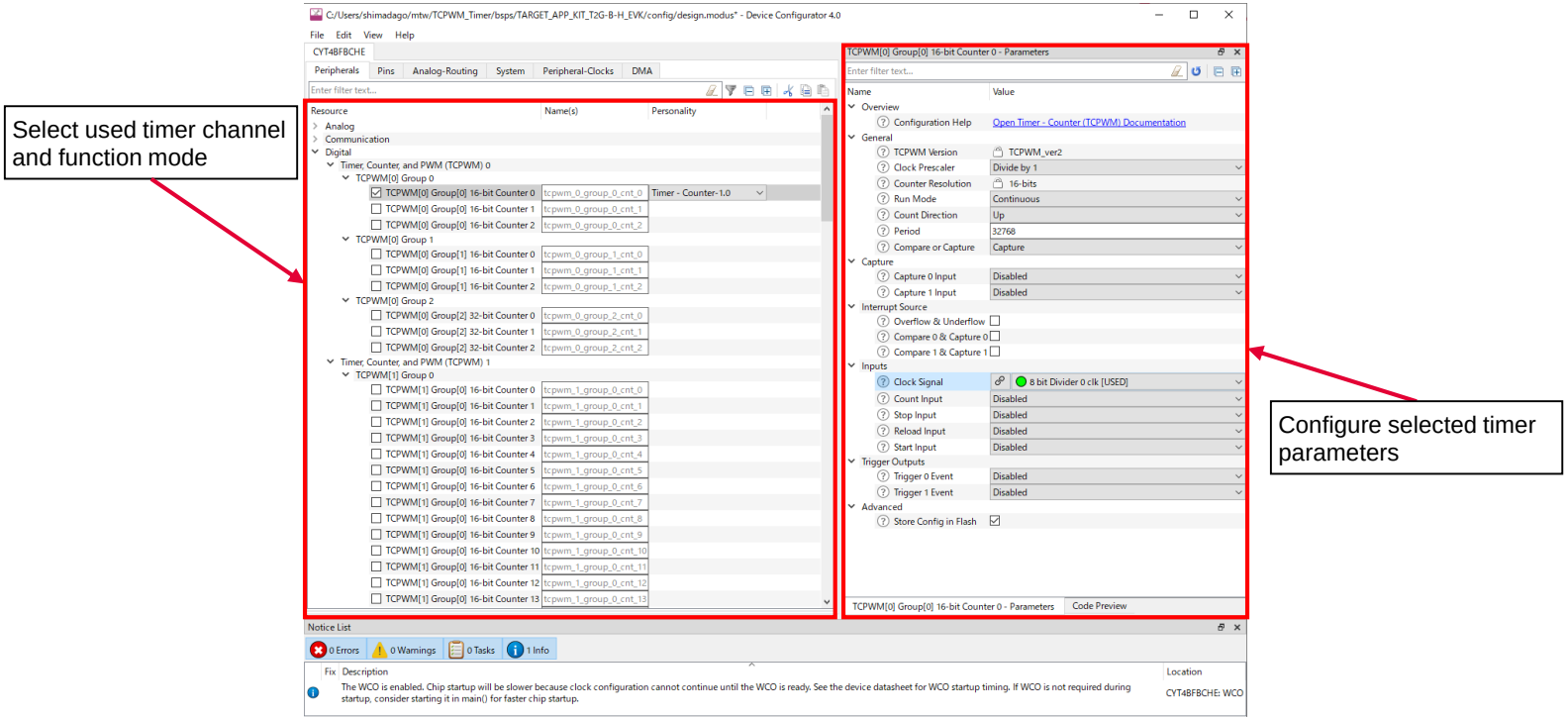
b) Click the Device Configurator link in the Quick Panel



Device configurator view for timer config

› Peripherals tab

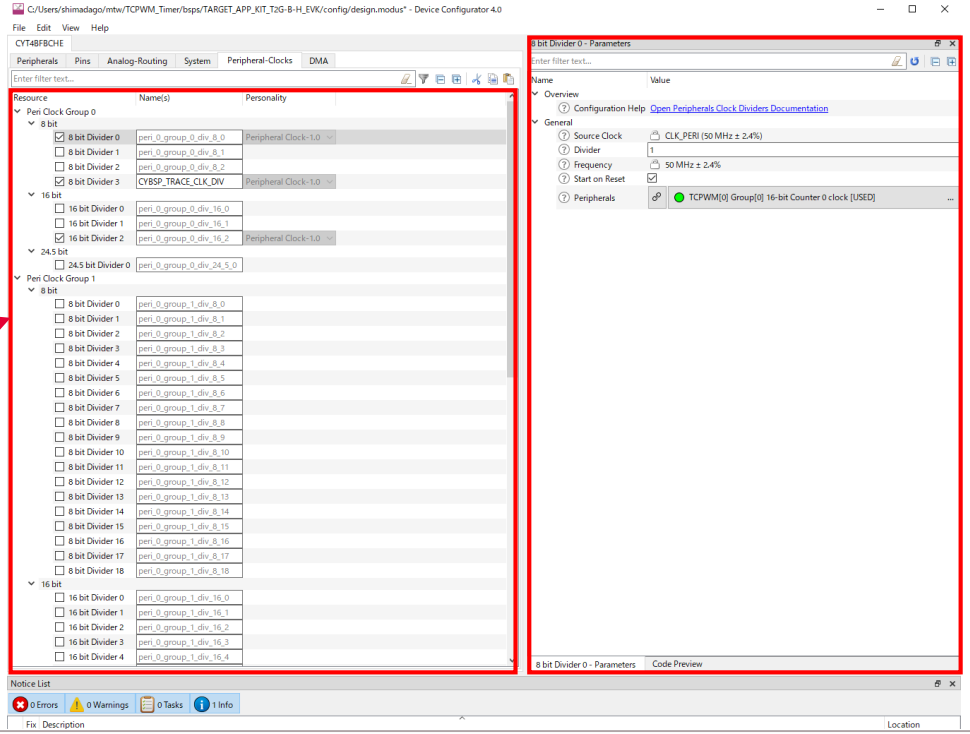
- Set each timer on the Peripheral tab



Device configurator view for timer config (contd.)

› Peripheral-Clocks tab

- Set peripheral clock (PCLK) on the Peripheral-Clocks tab.



Quick start

To use the Device configurator for Timer setting

- Launch the Device configurator.
- Use the various pull-down menus to configure signals. Refer to the descriptions in the Routing tab section for more details.
- Save the file to generate source code.
- The Device Configurator generates code into a "GeneratedSource" directory in your Eclipse IDE application, or in the same location you saved the *.modus file for non-IDE applications. That directory contains the necessary source (.c) and header (.h) files for the generated firmware, which uses the relevant driver APIs to configure the hardware.
- Use the generated structures as input parameters for Timer configuration functions in your application.

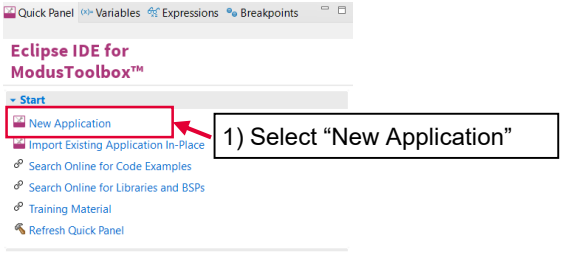
Use case

- CLK_PERI frequency set to 50 MHz
- TCPWM [0] Group [0] 16-bit Counter 1 operates as PWM mode (TCPWM_PWM)
  - It works at 50 MHz and the period is set to 9999
  - It outputs the PWM waveform to start TCPWM_COUNTER
  - Use an 8-bit Divider 1 clock (TCPWM_COUNTER_CLK) to generate 50 MHz
- TCPWM [0] Group [0] 16-bit Counter 0 operates as Counter mode (TCPWM_COUNTER)
  - It works at 200 kHz and the period is set to 60000
  - Compare value is set to 50000 (Generate compare match interrupt)
  - Use an 8-bit Divider 0 clock (TCPWM_COUNTER_CLK) to generate 200 kHz
  - It starts counting by the TCPWM_PWM event
- See the TCPWM_Counter application for operation

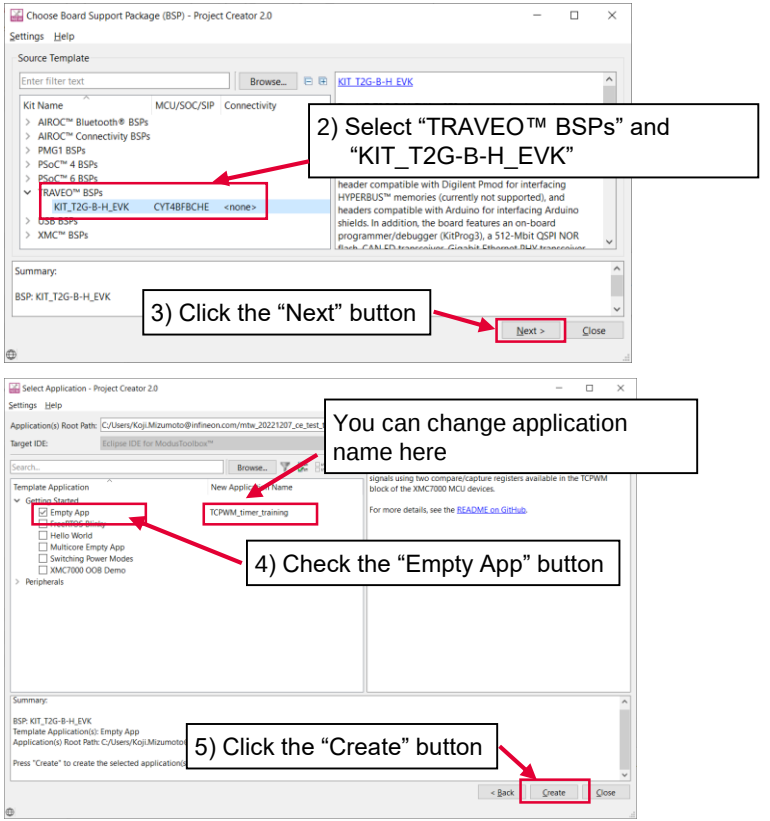
Timer configuration

› Create project

1) Click **New Application** in the Quick Panel and open the **Choose Board Support Package (BSP)** window



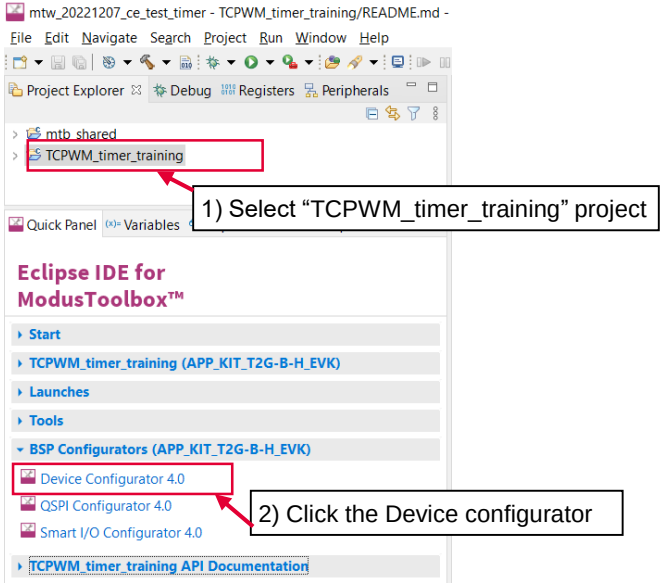
2) Select **TRAVEO™ BSPs** and **KIT_T2G-B-H_EVK**
3) Click the **Next** button and open the Application window
4) In this use case, it changes to “TCPWM_timer_training”
5) Click the **Create** button, and then start application creation

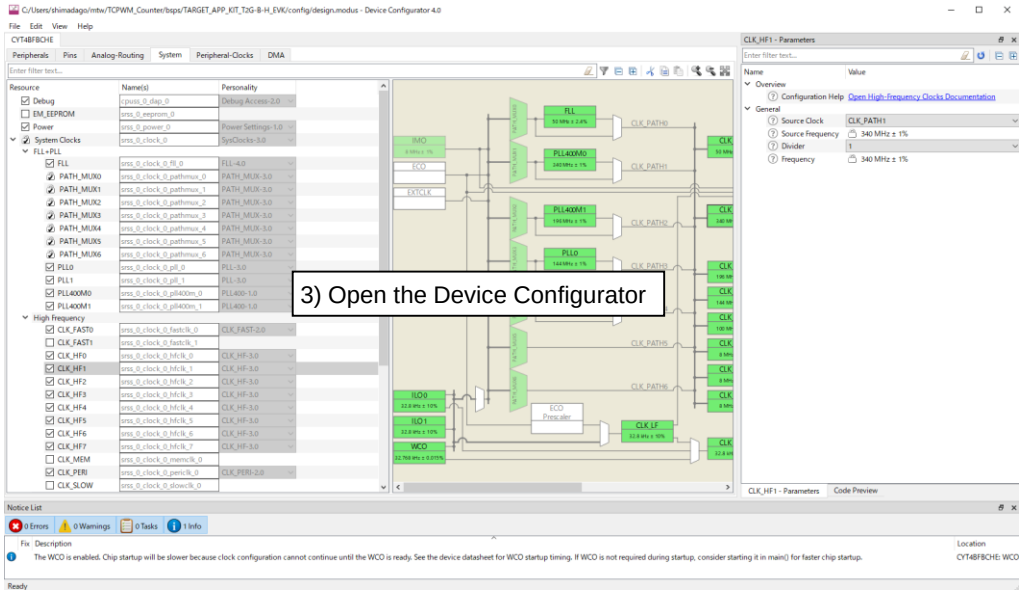


Timer configuration (contd.)

› Launch the Device configurator

1) Select the **TCPWM_timer_training** project.
2) Click the Device configurator in the Quick Panel
3) Then, open the Device Configurator window





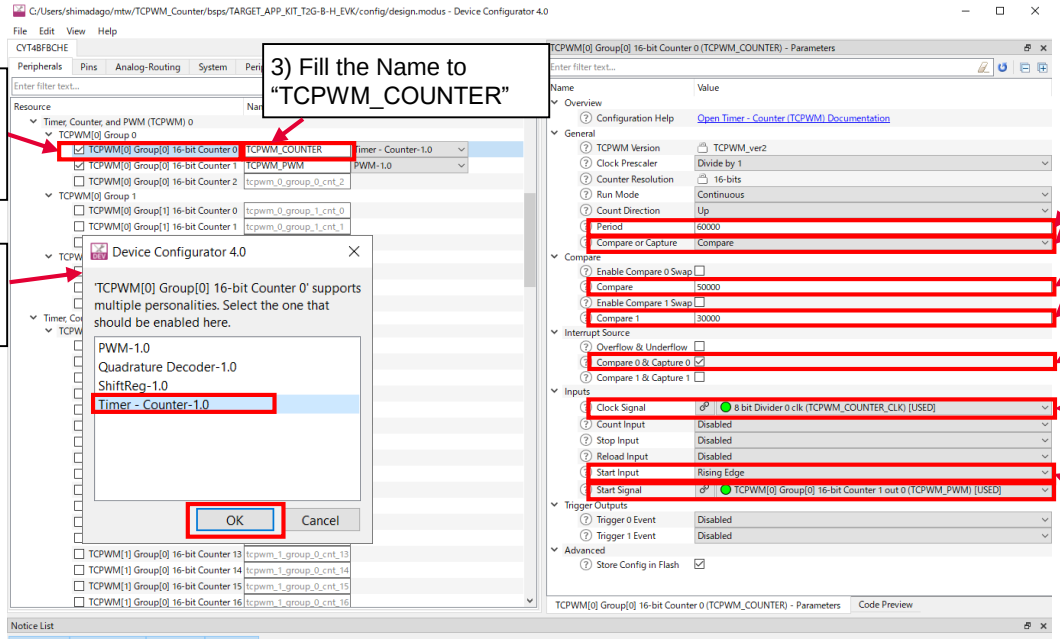
Timer configuration (contd.)

› Configure TCPWM_COUNTER

- Open the Peripherals tab and make the following settings
- The values without description are default settings. You should set them only if needed.

1) Select TCPWM [0] Group [0] 16-bit Counter 0, then open the function mode window

2) Select Timer – Counter – 1.0, then, click “OK” button



Set Period to 60000

Select to Compare

Set Compare to 50000, and Compare 1 to 30000

Check Compare 0 to generate compare match interrupt

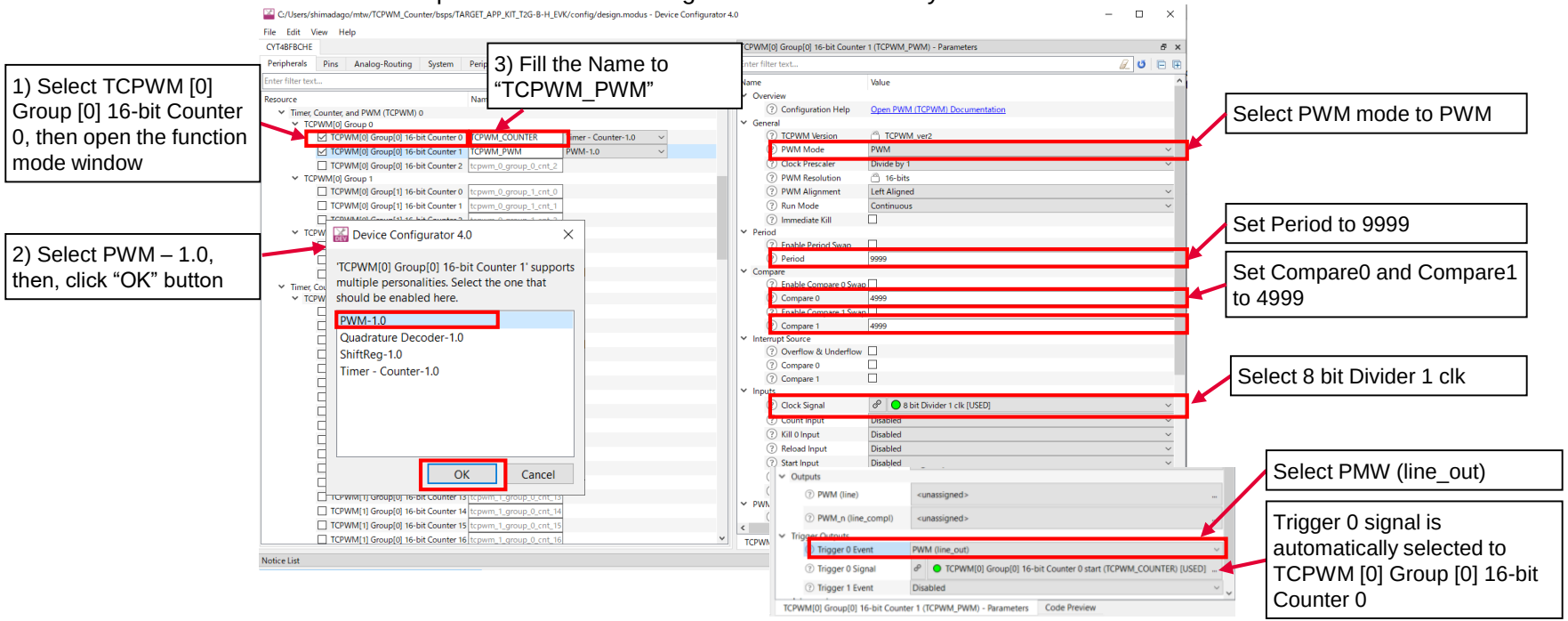
Select 8 bit Divider 0 as Input clock

Select Start Input to Rising Edge, then, select Start Signal to TCPWM [0] / Group [0] / 16-bit Counter 1.

Timer configuration (contd.)

› Configure TCPMW_PWM

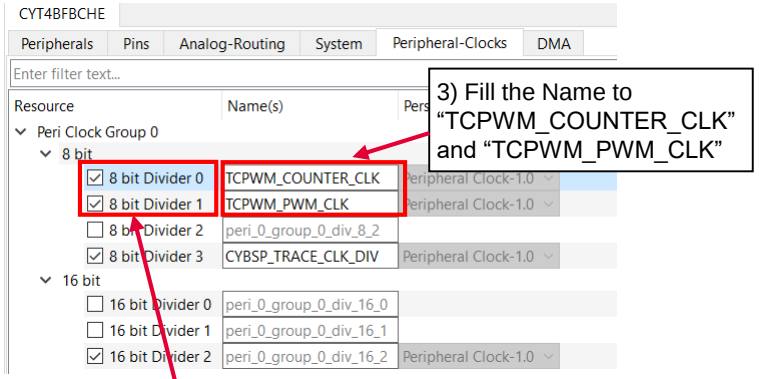
- Open the Peripherals tab and make the following settings.
- The values without description are default settings. You should set only if needed.



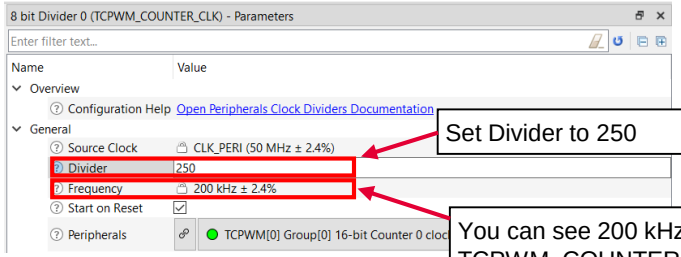
Timer configuration (contd.)

› Configure Peripheral clock divider

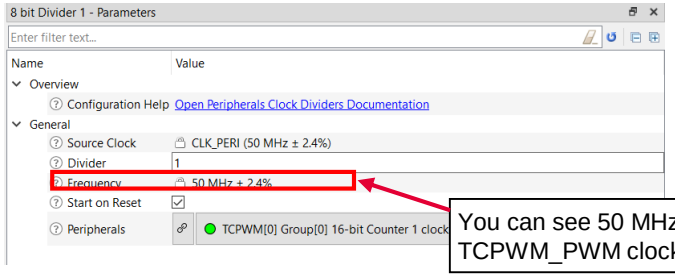
- Open the Peripheral-clocks tab and make the following dividers settings.



TCPWM_COUNTER_CLK



TCPWM_PWM_CLK

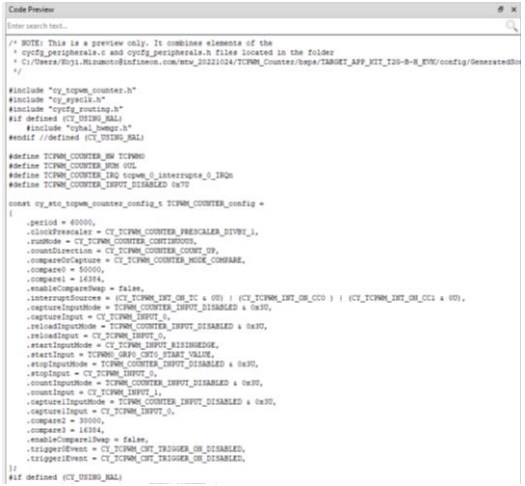


Timer configuration (contd.)

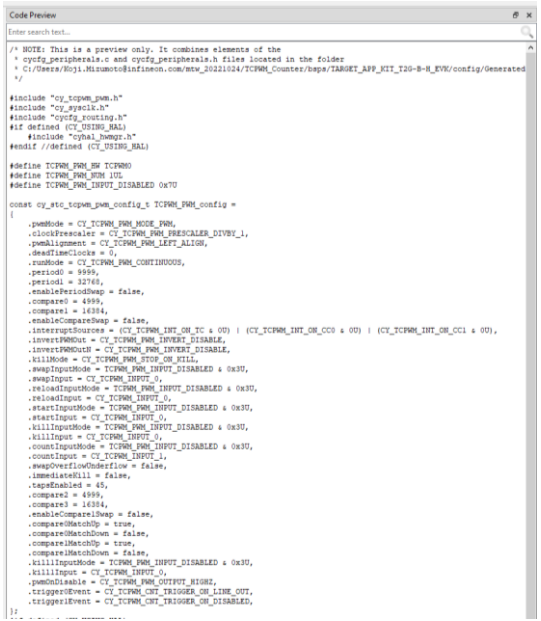
› Confirm configuration result

- You can check the configuration result in the “Code Preview” tab of the Device Configurator

TCPWM_COUNTER



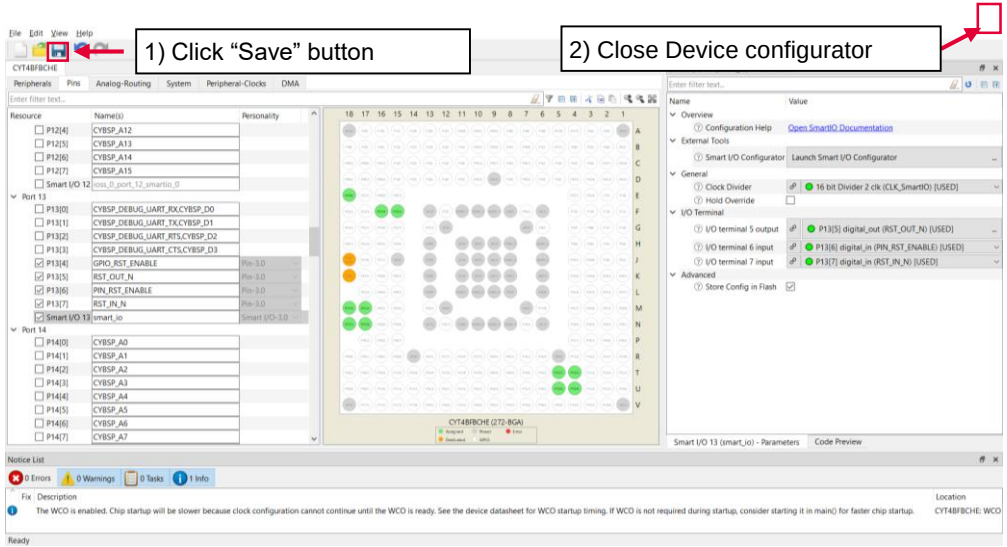
TCPWM_PWM



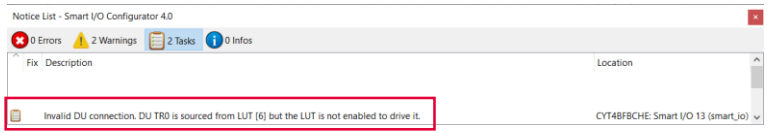
Timer configuration (contd.)

› Close Device configurator

- Click the **Save** button after completing all settings, then close the Device configurator



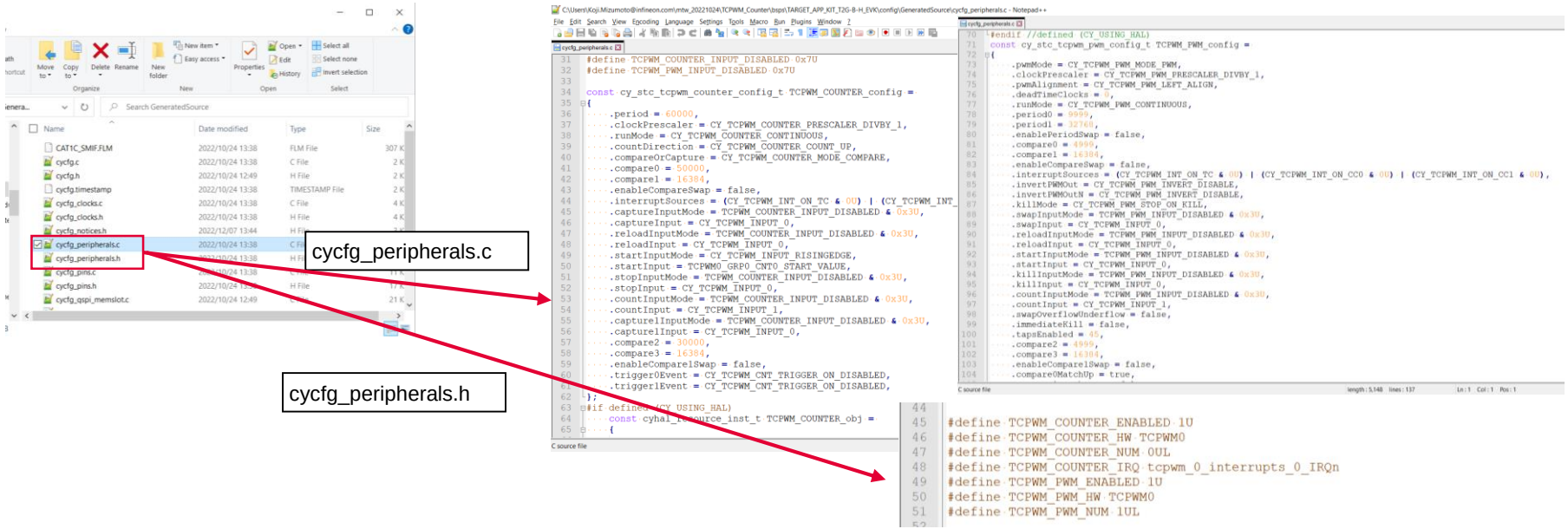
- If an **Errors/Tasks** message appears, it should be resolved according to the instructions



Timer configuration (contd.)

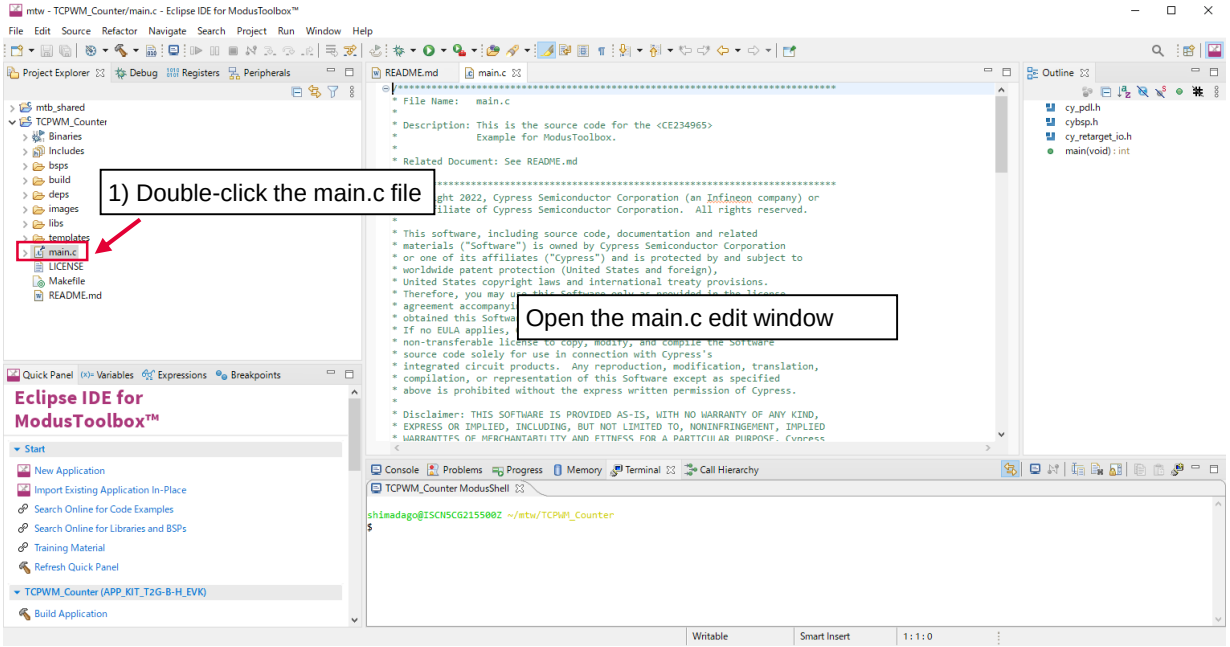
› Configuration file

- The Device Configurator generates code into a "GeneratedSource" directory in your Eclipse IDE application, or in the same location you saved the *.modus file for non-IDE applications.
- In this example, the following code is generated:



Implementation

› The structure generated by the Device Configurator can be used by implementing the following function in your application code.



Implementation (contd.)

› Add include file

README.md | main.c

```
 * reserves the right to make changes to the Software without notice. Cypress
 * does not assume any liability arising out of the application or use of the
 * Software or any product or circuit described in the Software. Cypress does
 * not authorize its products for use in any products where a malfunction or
 * failure of the Cypress product may reasonably be expected to result in
 * significant property damage, injury or death ("High Risk Product"). By
 * including Cypress's product in a High Risk Product, the manufacturer
 * of such system or application assumes all risk of such use and in doing
 * so agrees to indemnify Cypress against all liability.
 *******************************************************************************/

#include "cy_pdl.h"
#include "cybsp.h"
#include "cy_retarget_io.h"

/*******************************************************************************
 * Macros
 *******************************************************************************/
```

Add include file in main.c

Implementation (contd.)

› Add TCPWM_PWM initialization enable function

README.md | main.c

```
int main(void)
{
    cy_rslt_t result;
    uint32_t intrMask;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Enable global interrupts */
    __enable_irq();

    /*Initialize the User LED*/
    Cy_GPIO_Pin_Init(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_PIN, &CYBSP_USER_LED1_config );

    /*TCPWM PWM Mode initial*/
    if (CY_TCPWM_SUCCESS != Cy_TCPWM_PWM_Init(TCPWM_PWM_HW, TCPWM_PWM_NUM, &TCPWM_PWM_config))
    {
        CY_ASSERT(0);
    }

    /* Enable the initialized PWM */
    Cy_TCPWM_PWM_Enable(TCPWM_PWM_HW, TCPWM_PWM_NUM);

    /* Then start the PWM */
    Cy_TCPWM_TriggerReloadOrIndex_Single(TCPWM_PWM_HW, TCPWM_PWM_NUM);

    /*TCPWM Counter Mode initial*/
    if (CY_TCPWM_SUCCESS != Cy_TCPWM_Counter_Init(TCPWM_COUNTER_HW, TCPWM_COUNTER_NUM, &TCPWM_COUNTER_config))
    {
        CY_ASSERT(0);
    }
    /* Enable the initialized counter */
    Cy_TCPWM_Counter_Enable(TCPWM_COUNTER_HW, TCPWM_COUNTER_NUM);

    /* Then start the counter, the counter will be trigger by PWM signal */

    for (;;)
    {
        /*get the counter CC0 compare interrupt mask*/
        intrMask = Cy_TCPWM_GetInterruptStatusMasked(TCPWM_COUNTER_HW, TCPWM_COU
        if (intrMask == CY_TCPWM_INT_ON_CC0)
        {
            /*toggle user LED1*/
            Cy_GPIO_Inv(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_PIN);

            /*clear the counter CC0 compare interrupt*/
            Cy_TCPWM_ClearInterrupt(TCPWM_COUNTER_HW, TCPWM_COUNTER_NUM, CY_TCPWM_INT_ON_CC0);
        }
    }
}
```

There is structure to configure TCPWM_PWM in the cycfg_peripherals.c file

Add TCPWM_PWM initialization function

Add TCPWM_PWM enable function

Add TCPWM_PWM start function (Software reload event generation)

README.md | main.c | cycfg_peripherals.c

```
#endif //defined (CY_USING_HAL)
const cy_stc_tcpwm_pwm_config_t TCPWM_PWM_config =
{
    .pwmMode = CY_TCPWM_PWM_MODE_PWM,
    .clockPrescaler = CY_TCPWM_PWM_PRESCALER_DIVBY_1,
    .pwmAlignment = CY_TCPWM_PWM_LEFT_ALIGN,
    .deadTimeClocks = 0,
    .runMode = CY_TCPWM_PWM_CONTINUOUS,
    .period0 = 9999,
    .period1 = 32768,
    .enablePeriodSwap = false,
    .compare0 = 4999,
    .compare1 = 16384,
    .enableCompareSwap = false,
    .interruptSources = (CY_TCPWM_INT_ON_TC & 0U) | (CY_TCPWM_INT_ON_CC0 & 0U) | (CY_TCPWM_INT_ON_CC1 & 0U),
    .invertPWMOut = CY_TCPWM_PWM_INVERT_DISABLE,
    .invertPWMOutN = CY_TCPWM_PWM_INVERT_DISABLE,
    .killMode = CY_TCPWM_PWM_STOP_ON_KILL,
    .swapInputMode = TCPWM_PWM_INPUT_DISABLED & 0x3U,
    .swapInput = CY_TCPWM_INPUT_0,
    .reloadInputMode = TCPWM_PWM_INPUT_DISABLED & 0x3U,
    .reloadInput = CY_TCPWM_INPUT_0,
    .startInputMode = TCPWM_PWM_INPUT_DISABLED & 0x3U,
    .startInput = CY_TCPWM_INPUT_0,
    .killInputMode = TCPWM_PWM_INPUT_DISABLED & 0x3U,
    .killInput = CY_TCPWM_INPUT_0,
    .countInputMode = TCPWM_PWM_INPUT_DISABLED & 0x3U,
    .countInput = CY_TCPWM_INPUT_1,
    .swapOverflowUnderflow = false,
    .immediateKill = false,
    .tapsEnabled = 45,
    .compare2 = 4999,
    .compare3 = 16384,
    .enableCompare1Swap = false,
    .compare0MatchUp = true,
    .compare0MatchDown = false,
    .compare1MatchUp = true,
    .compare1MatchDown = false,
    .killInputMode = TCPWM_PWM_INPUT_DISABLED & 0x3U,
    .killInput = CY_TCPWM_INPUT_0,
    .pwmOnDisable = CY_TCPWM_PWM_OUTPUT_HIGHZ,
    .trigger0Event = CY_TCPWM_CNT_TRIGGER_ON_LINE_OUT,
    .trigger1Event = CY_TCPWM_CNT_TRIGGER_ON_DISABLED,
};
#if defined (CY_USING_HAL)
    const cyhal_resource_inst_t TCPWM_PWM_obj =
    {
        .type = CYHAL_RSC_TCPWM,
        .block_num = 0U,
        .channel_num = 1U,
    };
#endif //defined (CY_USING_HAL)

void init_cycfg_peripherals(void)
{
    Cy_SysClk_PeriphAssignDivider(PCLK_TCPWM0_CLOCKS0, CY_SYSCLK_DIV_8_BIT, 0U);
    Cy_SysClk_PeriphAssignDivider(PCLK_TCPWM0_CLOCKS1, CY_SYSCLK_DIV_8_BIT, 1U);
}

void reserve_cycfg_peripherals(void)
{
#if defined (CY_USING_HAL)
```

You can use the “TCPWM_PWM_HW” (TCPWM#0) and “TCPWM_PWM_NUM” (Counter#1) to specify the hardware

```
#define TCPWM_COUNTER_ENABLED 1U
#define TCPWM_COUNTER_HW TCPWM0
#define TCPWM_COUNTER_NUM 0UL
#define TCPWM_COUNTER_IRQ tcpwm_0_interrupts_0_IRQn
#define TCPWM_PWM_ENABLED 1U
#define TCPWM_PWM_HW TCPWM0
#define TCPWM_PWM_NUM 1UL
```

Implementation (contd.)

› Add TCPWM_COUNTER initialization enable function

README.md | main.c

```
int main(void)
{
    cy_rslt_t result;
    uint32_t intrMask;

    /* Initialize the device and board peripherals */
    result = cybsp_init() ;
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Enable global interrupts */
    __enable_irq();

    /*Initialize the User LED*/
    Cy_GPIO_Pin_Init(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_PIN, &CYBSP_USER_LED1_config );

    /*TCPWM PWM Mode initial*/
    if (CY_TCPWM_SUCCESS != Cy_TCPWM_PWM_Init(TCPWM_PWM_HW, TCPWM_PWM_NUM, &TCPWM_PWM_config))
    {
        CY_ASSERT(0);
    }

    /* Enable the initialized PWM */
    Cy_TCPWM_PWM_Enable(TCPWM_PWM_HW, TCPWM_PWM_NUM);

    /* Then start the PWM */
    Cy_TCPWM_TriggerReloadOrIndex_Single(TCPWM_PWM_HW, TCPWM_PWM_NUM);

    /*TCPWM Counter Mode initial*/
    if (CY_TCPWM_SUCCESS != Cy_TCPWM_Counter_Init(TCPWM_COUNTER_HW, TCPWM_COUNTER_NUM, &TCPWM_COUNTER_config))
    {
        CY_ASSERT(0);
    }
    /* Enable the initialized counter */
    Cy_TCPWM_Counter_Enable(TCPWM_COUNTER_HW, TCPWM_COUNTER_NUM);

    /* Then start the counter, the counter will be trigger by PWM signal */

    for (;;)
    {
        /*get the counter CC0 compare interrupt mask*/
        intrMask = Cy_TCPWM_GetInterruptStatusMasked(TCPWM_COUNTER_HW, TCPWM_COUNTER
        if (intrMask == CY_TCPWM_INT_ON_CC0)
        {
            /*toggle user LED1*/
            Cy_GPIO_Inv(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_PIN);

            /*clear the counter CC0 compare interrupt*/
            Cy_TCPWM_ClearInterrupt(TCPWM_COUNTER_HW, TCPWM_COUNTER_NUM, CY_TCPWM_INT_ON_CC0);
        }
    }
}
```

README.md | main.c | cycfg_peripherals.c

```
#include "cycfg_peripherals.h"

#define TCPWM_COUNTER_INPUT_DISABLED 0x7U
#define TCPWM_PWM_INPUT_DISABLED 0x7U

const cy_stc_tcpwm_counter_config_t TCPWM_COUNTER_config =
{
    .period = 60000,
    .clockPrescaler = CY_TCPWM_COUNTER_PRESCALER_DIVBY_1,
    .runMode = CY_TCPWM_COUNTER_CONTINUOUS,
    .countDirection = CY_TCPWM_COUNTER_COUNT_UP,
    .compareOrCapture = CY_TCPWM_COUNTER_MODE_COMPARE,
    .compare0 = 30000,
    .compare1 = 16384,
    .enableCompareSwap = false,
    .interruptSources = (CY_TCPWM_INT_ON_TC & 0U) | (CY_TCPWM_INT_ON_CC0 ) | (CY_TCPWM_INT_ON_CC1 & 0U),
    .captureInputMode = TCPWM_COUNTER_INPUT_DISABLED & 0x3U,
    .captureInput = CY_TCPWM_INPUT_0,
    .reloadInputMode = TCPWM_COUNTER_INPUT_DISABLED & 0x3U,
    .reloadInput = CY_TCPWM_INPUT_0,
    .startInputMode = CY_TCPWM_INPUT_RISINGEDGE,
    .startInput = TCPWM0_GRP0_CNT0_START_VALUE,
    .stopInputMode = TCPWM_COUNTER_INPUT_DISABLED & 0x3U,
    .stopInput = CY_TCPWM_INPUT_0,
    .countInputMode = TCPWM_COUNTER_INPUT_DISABLED & 0x3U,
    .countInput = CY_TCPWM_INPUT_1,
    .capture1InputMode = TCPWM_COUNTER_INPUT_DISABLED & 0x3U,
    .capture1Input = CY_TCPWM_INPUT_0,
    .compare2 = 30000,
    .compare3 = 16384,
    .enableCompare1Swap = false,
    .trigger0Event = CY_TCPWM_CNT_TRIGGER_ON_DISABLED,
    .trigger1Event = CY_TCPWM_CNT_TRIGGER_ON_DISABLED,
};
#if defined (CY_USING_HAL)
    const cyhal_resource_inst_t TCPWM_COUNTER_obj =
    {
        .type = CYHAL_RSC_TCPWM,
        .block_num = 0U,
        .channel_num = 0U,
    };
#endif //defined (CY_USING_HAL)
```

There is structure to configure TCPWM_COUNTER in the cycfg_peripherals.c file

Add TCPWM_COUNTER initialization function

Add TCPWM_COUNTER enable function

You can use the “TCPWM_COUNTER_HW” (TCPWM#0) and “TCPWM_COUNTER_NUM” (Counter#0) to specify the hardware

```
#define TCPWM_COUNTER_ENABLED 1U
#define TCPWM_COUNTER_HW TCPWM0
#define TCPWM_COUNTER_NUM 0UL
#define TCPWM_COUNTER_IRQ tcpwm_0_interrupts_0_IRQn
#define TCPWM_PWM_ENABLED 1U
#define TCPWM_PWM_HW TCPWM0
#define TCPWM_PWM_NUM 1UL
```

In this example, TCPWM_COUNTER starts with a TCPWM_PWM trigger. So you don't use the start function.

Implementation (contd.)

TCPWM_PWM initialization:

› Call the ***Cy_TCPWM_PWM_Init()*** function to initialize TCPWM_PWM
  – Configure TCPWM with parameters in the ***TCPWM_PWM_config*** structure

TCPWM_PWM enable:

› Call the ***Cy_TCPWM_PWM_Enable()*** function to enable TCPWM_PWM

TCPWM_PWM start:

› Call the ***Cy_TCPWM_TriggerReloadOrIndex_Single()*** function to start TCPWM_PWM
  – Triggers a software reload event on the specified TCPWM

Implementation (contd.)

TCPWM_COUNTER initialization:

› The ***Cy_TCPWM_Counter_Init()*** function initializes TCPWM as a counter mode

  – Configure TCPWM with parameters in ***TCPWM_COUNTER_config*** structure

TCPWM_COUNTER enable:

› Call the ***Cy_TCPWM_Counter_Enable()*** function to enable the TCPWM counter

References

Datasheet

› CYT4BF datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family

Architecture Technical reference manual

› TRAVEO™ T2G automotive body controller high family architecture technical reference manual

Registers Technical reference manual

› TRAVEO™ T2G Automotive body controller high registers technical reference manual

PDL/HAL

› PDL

› HAL

Training

› TRAVEO™ T2G Training

Revision History

Revision	ECN	Submission Date	Description of Change
**	7847266	2022/12/13	Initial release

Important notice and warnings

All referenced product or service names and trademarks are the property of their respective owners.

Edition 2022-12
Published by
Infineon Technologies AG
81726 Munich, Germany

Do you have a question about this document?
Go to:
www.infineon.com/support

Document reference
002-36709 Rev. **

IMPORTANT NOTICE

The information given in this document shall in no event be regarded as a guarantee of conditions or characteristics ("Beschaffenheitsgarantie") .

With respect to any examples, hints or any typical values stated herein and/or any information regarding the application of the product, Infineon Technologies hereby disclaims any and all warranties and liabilities of any kind, including without limitation warranties of non-infringement of intellectual property rights of any third party.

In addition, any information given in this document is subject to customer's compliance with its obligations stated in this document and any applicable legal requirements, norms and standards concerning customer's products and any use of the product of Infineon Technologies in customer's applications.

The data contained in this document is exclusively intended for technically trained staff. It is the responsibility of customer's technical departments to evaluate the suitability of the product for the intended application and the completeness of the product information given in this document with respect to such application.

For further information on the product, technology, delivery terms and conditions and prices please contact your nearest Infineon Technologies office (**www.infineon.com**).

WARNINGS

Due to technical requirements products may contain dangerous substances. For information on the types in question please contact your nearest Infineon Technologies office.

Except as otherwise explicitly approved by Infineon Technologies in a written document signed by authorized representatives of Infineon Technologies, Infineon Technologies' products may not be used in any applications where a failure of the product or any consequences of the use thereof can reasonably be expected to result in personal injury.

Part of your life. Part of tomorrow.