**Please note that Cypress is an Infineon Technologies Company.**
The document following this cover page is marked as “Cypress” document as this is the company that originally developed the product. Please note that Infineon will continue to offer the product to new and existing customers as part of the Infineon product portfolio.

**Continuity of document content**
The fact that Infineon offers the following product as part of the Infineon product portfolio does not lead to any changes to this document. Future revisions will occur when appropriate, and any changes will be set out on the document history page.

**Continuity of ordering part numbers**
Infineon continues to support existing part numbers. Please continue to use the ordering part numbers listed in the datasheet for ordering.

AN228104

# How to Use Trigger Multiplexer in Traveo II Family

**Author: Wafa Syakira Binti Azmi**

**Associated Part Family: Traveo™ II Family**

**Related Documents: For a complete list, see Related Documents**

This application note describes Trigger Multiplexer for Traveo™ II Family and explains how to route trigger signals from the source peripherals to the destinations.

## Contents

1 Introduction....1
2 Trigger Multiplexer Overview....1
2.1 Group Trigger....4
2.2 One-to-One Trigger....8
2.3 Software Trigger....10
3 Application....12
3.1 Triggering P-DMA Transfer by TCPWM Timer....12
3.2 Simultaneous ADC Conversion on Three SARs by Single TCPWM Timer....17
3.3 Triggering ADC Conversion by TCPWM Timer....24
3.4 Simultaneous Starting of TCPWM Timer by SW Trigger....30
4 Glossary....36
5 Related Documents....36
6 Other References....36
Document History....37
Worldwide Sales and Design Support....38

## 1 Introduction

Every peripheral in the Traveo II device is interconnected using trigger signals. Trigger signals are means by which peripherals inform the occurrence of an event or transition to a different state. Trigger signals are used to initiate an action in other peripherals. For example, triggers can initiate data transfer over DMA (see section 3.1), conversion on SAR ADC (see sections 3.2 and 3.3), or start a timer (see section 3.4). Trigger multiplexers are simple multiplexers that are designed to route these trigger signals from the source peripherals to the desired destinations.

In this application note, you will learn how to set up trigger routes from the source peripherals to the desired destinations.

To know more about the functionality and terminology used in this application note, see the “Trigger Multiplexer” chapter of the Architecture Technical Reference Manual (TRM).

## 2 Trigger Multiplexer Overview

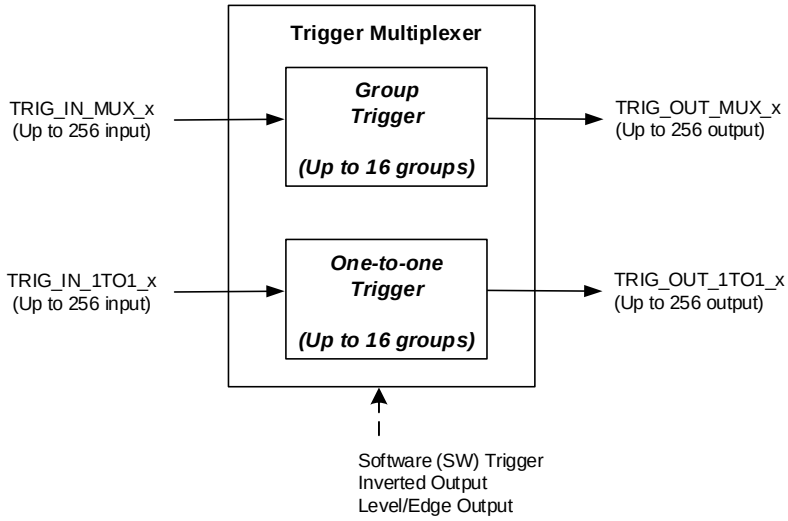
The trigger inputs are output signals from the source peripheral. The trigger outputs are typically input signals to the destination peripheral. The trigger multiplexer multiplexes trigger input and trigger output.

Trigger multiplexer has two group types:

- Multiplexer-based group type (Group trigger)
  - Connects a peripheral input trigger to multiple peripheral output triggers
- One-to-one-based group type (One-to-one trigger)
  - Connects a peripheral input trigger to a specific output trigger

Each group type consists of multiple trigger groups (up to 16). Each group type is associated with the trigger inputs of a specific peripheral. Figure 1 shows the trigger multiplexer block diagram.

Figure 1. Trigger Multiplexer Block Diagram




Trigger multiplexer has the following input and output signals:

- TRIG_IN_MUX_x is the input trigger of group trigger. It has up to 256 input signals.
- TRIG_IN_1TO1_x is the input trigger of one-to-one trigger. It has up to 256 input signals.
- TRIG_OUT_MUX_x is the output trigger of group trigger. It has up to 256 output signals.
- TRIG_OUT_1TO1_x is the output trigger of one-to-one trigger. It has up to 256 output signals.

Trigger multiplexer features SW trigger, inverted output trigger, and level or edge sensitive trigger.

SW trigger is initiated by SW and can trigger any signal in the trigger multiplexer. Inverted output specifies the polarity of output signals. Level or edge trigger specifies if the output trigger is treated as a level sensitive or edge sensitive trigger.

The suffix “x” represents the name of the peripheral block. Table 1 shows the output triggers and the input triggers of a group trigger that can be routed to each other, and the availability of routing in Traveo II device series. However, some combinations cannot be routed to some unit and channel numbers. For details on the units and channels of each peripheral, see the datasheets.

Table 1. Routing between Output Trigger and Input Trigger of Group Trigger by Series

| TRIG_OUT_MUX_x | TRIG_IN_MUX_x | CYT2B7 | CYT2B9 | CYT4BF | CYT4DN |
|---|---|---|---|---|---|
| PDMA | PDMA | ✓ | ✓ | ✓ | ✓ |
| | MDMA | ✓ | ✓ | ✓ | ✓ |
| | FAULT | ✓ | ✓ | ✓ | ✓ |
| | CTI | ✓ | ✓ | ✓ | ✓ |
| | EVTGEN | ✓ | ✓ | ✓ | ✓ |
| | HSIOM | ✓ | ✓ | ✓ | ✓ |
| | TCPWM | ✓ | ✓ | ✓ | ✓ |
| | CAN0_TT | ✓ | ✓ | ✓ | |
| | CAN1_TT | ✓ | ✓ | ✓ | |
| | PASS_GEN | ✓ | ✓ | ✓ | ✓ |
| | FLEXRAY_TT | | | ✓ | |
| MDMA | MDMA | ✓ | ✓ | | |
| | TCPWM | | | ✓ | ✓ |

| TRIG_OUT_MUX_x | TRIG_IN_MUX_x | CYT2B7 | CYT2B9 | CYT4BF | CYT4DN |
|---|---|---|---|---|---|
| TCPWM | TCPWM | ✓ | ✓ | ✓ | ✓ |
| | CAN_TT | ✓ | ✓ | | ✓ |
| | PDMA | ✓ | ✓ | ✓ | ✓ |
| | MDMA | ✓ | ✓ | ✓ | ✓ |
| | CTI | ✓ | ✓ | ✓ | ✓ |
| | FAULT | ✓ | ✓ | ✓ | ✓ |
| | PASS_GEN | ✓ | ✓ | ✓ | ✓ |
| | HSIOM | ✓ | ✓ | ✓ | ✓ |
| | SCB | ✓ | ✓ | ✓ | ✓ |
| | SCB_I2C_SCL | ✓ | ✓ | ✓ | ✓ |
| | CAN_DBG | ✓ | ✓ | ✓ | ✓ |
| | CAN_FIFO | ✓ | ✓ | ✓ | ✓ |
| | CXPI | | ✓ | | ✓ |
| | EVTGEN | ✓ | ✓ | ✓ | ✓ |
| | SMIF | | | ✓ | ✓ |
| | I2S | | | ✓ | ✓ |
| | TDM | | | | ✓ |
| | SG | | | | ✓ |
| | PWM | | | | ✓ |
| | MIXER | | | | ✓ |
| | AUDIODAC | | | | ✓ |
| | FLEXRAY_TT | | | ✓ | |
| | FLEXRAY_IBUF | | | ✓ | |
| | FLEXRAY_OBUF | | | ✓ | |
| PASS_GEN | PDMA | ✓ | ✓ | ✓ | ✓ |
| | CTI | ✓ | ✓ | | |
| | FAULT | ✓ | ✓ | | |
| | EVTGEN | ✓ | ✓ | ✓ | ✓ |
| | PASS_GEN | ✓ | ✓ | | |
| | HSIOM | ✓ | ✓ | ✓ | ✓ |
| | TCPWM | ✓ | ✓ | ✓ | ✓ |
| CAN_TT | CAN_TT | ✓ | ✓ | ✓ | ✓ |
| | FLEXRAY_TT | | | ✓ | |
| FLEXRAY_TT | CAN_TT | | | ✓ | |
| | FLEXRAY_TT | | | ✓ | |
| HSIOM<br>PERI_DEBUG_FREEZE<br>PASS_DEBUG_FREEZE<br>SRSS_WDT_DEBUG_FREEZE<br>SRSS_MCWDT_DEBUG_FREEZE<br>TCPWM_DEBUG_FREEZE | TR_GROUP[i]_OUTPUT | ✓ | ✓ | ✓ | ✓ |
| I2S_DEBUG_FREEZE<br>TDM_DEBUG_FREEZE<br>SG_DEBUG_FREEZE<br>PWM_DEBUG_FREEZE<br>PWM_DEBUG_FREEZE<br>MIXER_DEBUG_FREEZE<br>AUDIODAC_DEBUG_FREEZE | TR_GROUP[i]_OUTPUT | | | | ✓ |

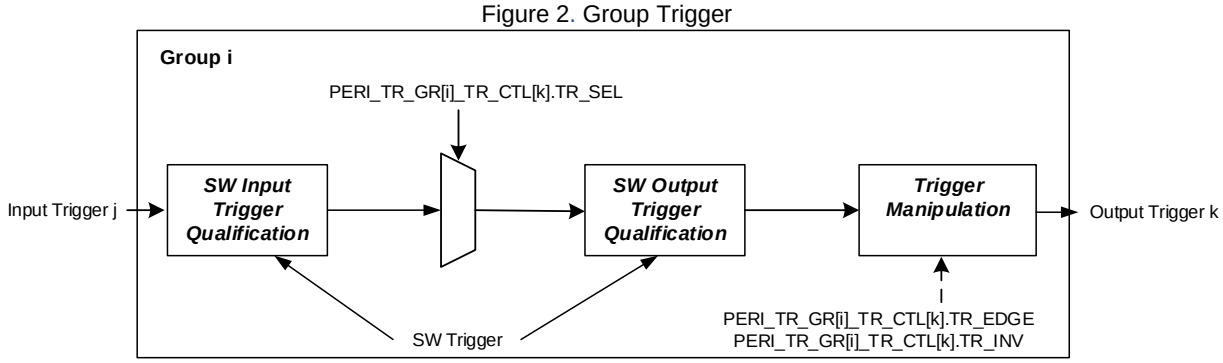
| TRIG_OUT_MUX_x | TRIG_IN_MUX_x | CYT2B7 | CYT2B9 | CYT4BF | CYT4DN |
|---|---|---|---|---|---|
| TR_GROUP[i]_INPUT | PDMA | ✓ | ✓ | ✓ | ✓ |
| | SCB | ✓ | ✓ | ✓ | ✓ |
| | SCB_I2C_SCL | ✓ | ✓ | ✓ | ✓ |
| | CAN_DBG | ✓ | ✓ | ✓ | ✓ |
| | CAN_FIFO | ✓ | ✓ | ✓ | ✓ |
| | CAN_TT | ✓ | ✓ | ✓ | ✓ |
| | CTI | ✓ | ✓ | ✓ | ✓ |
| | FAULT | ✓ | ✓ | ✓ | ✓ |
| | TCPWM | ✓ | ✓ | ✓ | ✓ |
| | MDMA | ✓ | ✓ | ✓ | ✓ |
| | PASS_GEN | ✓ | ✓ | ✓ | ✓ |
| | EVTGEN | ✓ | ✓ | ✓ | ✓ |
| | CXPI | | ✓ | | ✓ |
| | SMIF | | | ✓ | ✓ |
| | FLEXRAY_TT | | | ✓ | |
| | FLEXRAY_IBUF | | | ✓ | |
| | FLEXRAY_OBUF | | | ✓ | |
| | I2S | | | ✓ | ✓ |
| | HSIOM | | | ✓ | ✓ |
| | TDM | | | | ✓ |
| | SG | | | | ✓ |
| | PWM | | | | ✓ |
| | AUDIODAC | | | | ✓ |
| | MIXER | | | | ✓ |

In one-to-one group trigger, one input trigger is directly connected to a specific output trigger. For the routing of one-to-one trigger, see the datasheets.

## 2.1 Group Trigger

For a group trigger, an input trigger, TRIG_IN_MUX_x, is selected for each output trigger TRIG_OUT_MUX_x. Note that all output triggers in a group “i” share the same input triggers. Figure 2 shows the group trigger block diagram.

Figure 2. Group Trigger




One among the multiple output signals from the source peripheral is selected. For a specific output trigger in the trigger group, the input trigger can be specified via the PERI_TR_GR[i]_TR_CTL[k] register. Here, suffices “i” and “k” indicate the trigger group number and the output trigger number, respectively. Input trigger number can be specified by PERI_TR_GR[i]_TR_CTL[k].TR_SEL=j. This section describes a few examples of group triggers.

**Example 1**:

This example demonstrates the trigger of 16-bit TCPWM counter start upon receiving SCB RX pulse. This trigger routing is included in Multiplexer (MUX) Group 5. This example extracted from the CYT2B7 series datasheet that provides trigger group, input trigger, and output trigger. Table 2 and Table 3 show trigger outputs and trigger inputs of MUX Group 5 extracted from CYT2B7 series datasheet, respectively.

Table 2. Trigger Outputs of MUX Group 5 of CYT2B7 Series

| Output ("k") | Trigger Label | Description |
|---|---|---|
| **MUX Group 5:** TCPWM_IN (TCPWM0 Trigger Multiplexer) | | |
| 0:10 | TCPWM_ALL_CNT_TR_IN | Triggers to TCPWM0 |

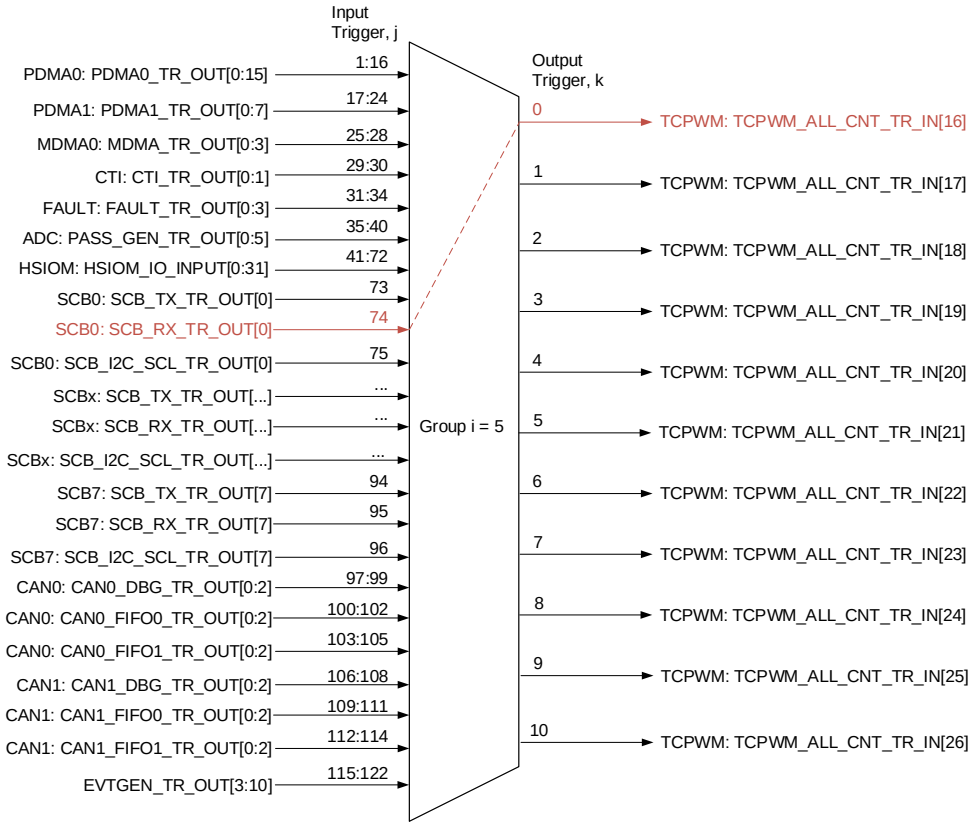
Table 3. Trigger Inputs of MUX Group 5 of CYT2B7 Series

| Input ("j") | Trigger Label | Description |
|---|---|---|
| **MUX Group 5:** TCPWM_IN (TCPWM0 Trigger Multiplexer) | | |
| 1:16 | PDMA0_TR_OUT[0:15] | General purpose P-DMA0 triggers |
| 17:24 | PDMA1_TR_OUT[0:7] | General purpose P-DMA1 triggers |
| 25:28 | MDMA_TR_OUT[0:3] | M-DMA0 triggers |
| 29:30 | CTI_TR_OUT[0:1] | Trace events |
| 31:34 | FAULT_TR_OUT[0:3] | Fault events |
| 35:40 | PASS_GEN_TR_OUT[0:5] | PASS SAR events |
| 41:72 | HSIOM_IO_INPUT[0:31] | I/O inputs |
| 73 | SCB_TX_TR_OUT[0] | SCB0 TX trigger |
| 74 | SCB_RX_TR_OUT[0] | SCB0 RX trigger |
| 75 | SCB_I2C_SCL_TR_OUT[0] | SCB0 I2C trigger |
| 76 | SCB_TX_TR_OUT[1] | SCB1 TX trigger |
| 77 | SCB_RX_TR_OUT[1] | SCB1 RX trigger |
| 78 | SCB_I2C_SCL_TR_OUT[1] | SCB1 I2C trigger |
| 79 | SCB_TX_TR_OUT[2] | SCB2 TX trigger |
| 80 | SCB_RX_TR_OUT[2] | SCB2 RX trigger |
| 81 | SCB_I2C_SCL_TR_OUT[2] | SCB2 I2C trigger |
| 82 | SCB_TX_TR_OUT[3] | SCB3 TX trigger |
| 83 | SCB_RX_TR_OUT[3] | SCB3 RX trigger |
| 84 | SCB_I2C_SCL_TR_OUT[3] | SCB3 I2C trigger |
| 85 | SCB_TX_TR_OUT[4] | SCB4 TX trigger |
| 86 | SCB_RX_TR_OUT[4] | SCB4 RX trigger |
| 87 | SCB_I2C_SCL_TR_OUT[4] | SCB4 I2C trigger |
| 88 | SCB_TX_TR_OUT[5] | SCB5 TX trigger |
| 89 | SCB_RX_TR_OUT[5] | SCB5 RX trigger |
| 90 | SCB_I2C_SCL_TR_OUT[5] | SCB5 I2C trigger |
| 91 | SCB_TX_TR_OUT[6] | SCB6 TX trigger |
| 92 | SCB_RX_TR_OUT[6] | SCB6 RX trigger |
| 93 | SCB_I2C_SCL_TR_OUT[6] | SCB6 I2C trigger |
| 94 | SCB_TX_TR_OUT[7] | SCB7 TX trigger |
| 95 | SCB_RX_TR_OUT[7] | SCB7 RX trigger |
| 96 | SCB_I2C_SCL_TR_OUT[7] | SCB7 I2C trigger |
| 97:99 | CAN0_DBG_TR_OUT[0:2] | CAN0 M-DMA0 events |
| 100:102 | CAN0_FIFO0_TR_OUT[0:2] | CAN0 FIFO0 events |

| Input (“j”) | Trigger Label | Description |
|---|---|---|
| 103:105 | CAN0_FIFO1_TR_OUT[0:2] | CAN0 FIFO1 events |
| 106:108 | CAN1_DBG_TR_OUT[0:2] | CAN1 M-DMA0 events |
| 109:111 | CAN1_FIFO0_TR_OUT[0:2] | CAN1 FIFO0 events |
| 112:114 | CAN1_FIFO1_TR_OUT[0:2] | CAN1 FIFO1 events |
| 115:122 | EVTGEN_TR_OUT[3:10] | EVTGEN triggers |

Figure 3 shows the input trigger and output trigger structure.

Figure 3. Trigger Routing between 16-bit TCPWM and P-DMA0 of CYT2B7 Series




The following explains how to connect P-DMA0 channel 9 and 16-bit TCPWM channel 0:

1. Configuration register selection:

   MUX Group 5 (i=5)

   Output trigger TCPWM_ALL_CNT_TR_IN[16] (k=0)

   Therefore, the register is specified as PERI_TR_GR**5**_TR_CTL**0**.

2. Input trigger selection:

   Input trigger SCB_RX_TR_OUT[0] (j=74)

The configuration will be PERI_TR_GR5_TR_CTL0.TR_SEL = **74**.

**Example 2**:

Activation of SAR ADC unit, on ADC channel, can be triggered by Event Generator. Generic trigger inputs are shared between ADC channels. One of the five generic triggers can be routed to any ADC channel. Group trigger for SAR ADC typically specializes in trigger control of multiple SAR ADC units. One-to-one trigger for SAR ADC is typically useful for trigger control of ADC logical channels within a SAR ADC unit. For more details on SAR ADC generic trigger inputs, see the “SAR ADC” chapter of the Architecture TRM.

For example, in CYT2B7 series, the input trigger Event Generator 0 is routed to a generic trigger 0 of SAR. This trigger routing is included in MUX Group 6. This example is based on Table 4 and Table 5 extracted from CYT2B7 series datasheet that provides trigger group, input trigger, and output trigger.

Table 4. Trigger Outputs of MUX Group 6 of CYT2B7 Series

| Output (“k”) | Trigger Label | Description |
|---|---|---|
| **MUX Group 6:** PASS (PASS SAR trigger multiplexer) | | |
| 0:11 | PASS_GEN_TR_IN[0:11] | Generic triggers to SAR ADCs |

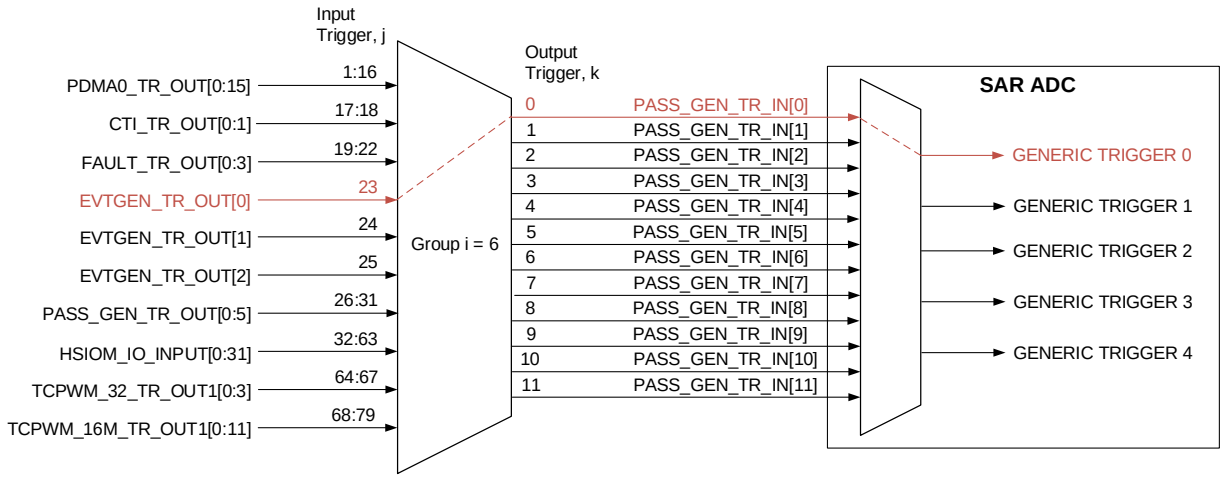
Table 5. Trigger Inputs of MUX Group 6 of CYT2B7 Series

| Input (“j”) | Trigger Label | Description |
|---|---|---|
| **MUX Group 6:** PASS (PASS SAR trigger multiplexer) | | |
| 1:16 | PDMA0_TR_OUT[0:15] | General purpose P-DMA0 triggers |
| 17:18 | CTI_TR_OUT[0:1] | Trace events |
| 19:22 | FAULT_TR_OUT[0:3] | Fault events |
| 23:25 | EVTGEN_TR_OUT[0:2] | EVTGEN triggers |
| 26:31 | PASS_GEN_TR_OUT[0:5] | PASS SAR done signals |
| 32:63 | HSIOM_IO_INPUT[0:31] | I/O inputs |
| 64:67 | TCPWM_32_TR_OUT1[0:3] | 32-bit TCPWM0 counters |
| 68:79 | TCPWM_16M_TR_OUT1[0:11] | 16-bit Motor enhanced TCPWM0 counters |

For the Trigger Group Inputs and Trigger Group Outputs tables of each series, see the device datasheets.

Figure 4 shows the input trigger and output trigger structure.

Figure 4. Trigger Routing between TCPWM 16-bit Motor and Generic Trigger 2 in ADC of CYT2B7 Series




The following explains how the trigger source Event Generator 0 is routed to a generic trigger 0 of SAR0:

1. Configuration register selection:

   MUX Group 6 (i=6)

Output trigger PASS_GEN_TR_IN[0] (k=0)

Therefore, the register is specified as PERI_TR_GR**6**_TR_CTL**0**.

2. Input trigger selection:

   Input trigger EVTGEN_TR_OUT[0] (j=23)

   The configuration will be PERI_TR_GR6_TR_CTL0.TR_SEL = **23**.

3. Generic trigger selection:

   Generic trigger SAR_TR_IN_SEL0.IN0_SEL = **0**.

Table 6 shows the bit registers for a group trigger that explains the inverted output trigger and the level or edge sensitive trigger. For more details, see the Registers TRM.

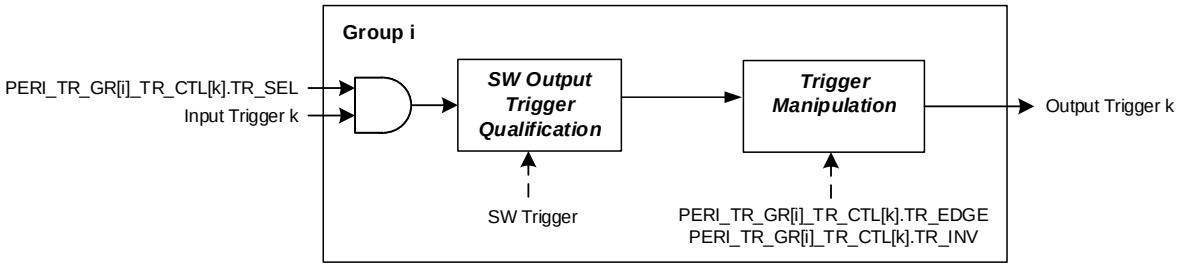
Table 6. Group Trigger Control Bit Registers

| Bit Register | Description |
|---|---|
| PERI_TR_GR[i]_TR_CTL[k].TR_INV | Specifies if the output trigger is inverted.<br>'0': Not inverted<br>'1': Inverted |
| PERI_TR_GR[i]_TR_CTL[k].TR_EDGE | Specifies if the (inverted) output trigger is treated as a level sensitive or edge sensitive trigger.<br>'0': Level sensitive<br>'1': Edge sensitive trigger |

## 2.2 One-to-One Trigger

For a one-to-one trigger, an input trigger, TRIG_IN_1TO1_x, is connected to the output trigger TRIG_OUT_1TO1_x. A one-to-one group has AND gate functionality to disable an input trigger. Figure 5 shows the one-to-one trigger block diagram.

Figure 5. One-to-One Trigger




One-to-one group trigger can be specified by the PERI_TR_1TO1_GR[i]_TR_CTL[k] register. Here, suffices "i" and "k" indicate the trigger group number and the output trigger number, respectively.

PERI_TR_1TO1_GR[i]_TR_CTL[k].TR_SEL is used to control the enabling/disabling of input trigger. If set to '1', the input trigger is enabled. If set to '0', the input trigger is disabled and set to constant signal level '0'.

This section describes an example of one-to-one trigger.

**Example:**

This is an example to trigger data transmission on P-DMA1 channel 15 by SCB UART channel 3. This trigger routing is included in MUX Group 8. The example is based on Table 7 extracted from the CYT2B7 series datasheet that provides trigger group and input trigger.

Table 7. Trigger One-to-One MUX Group 8 of CYT2B7 Series

| Input trigger (“k”) | Trigger Inputs | Trigger Outputs |
|---|---|---|
| **MUX Group 8:** SCB to P-DMA1 Triggers | | |
| 0 | SCB0_TX_TR_OUT | PDMA1_TR_IN[8] |
| 1 | SCB0_RX_TR_OUT | PDMA1_TR_IN[9] |
| 2 | SCB1_TX_TR_OUT | PDMA1_TR_IN[10] |
| 3 | SCB1_RX_TR_OUT | PDMA1_TR_IN[11] |
| 4 | SCB2_TX_TR_OUT | PDMA1_TR_IN[12] |
| 5 | SCB2_RX_TR_OUT | PDMA1_TR_IN13] |
| 6 | SCB3_TX_TR_OUT | PDMA1_TR_IN[14] |
| 7 | SCB3_RX_TR_OUT | PDMA1_TR_IN[15] |
| 8 | SCB4_TX_TR_OUT | PDMA1_TR_IN[16] |
| 9 | SCB4_RX_TR_OUT | PDMA1_TR_IN[17] |
| 10 | SCB5_TX_TR_OUT | PDMA1_TR_IN[18] |
| 11 | SCB5_RX_TR_OUT | PDMA1_TR_IN[19] |
| 12 | SCB6_TX_TR_OUT | PDMA1_TR_IN[20] |
| 13 | SCB6_RX_TR_OUT | PDMA1_TR_IN[21] |
| 14 | SCB7_TX_TR_OUT | PDMA1_TR_IN[22] |
| 15 | SCB7_RX_TR_OUT | PDMA1_TR_IN[23] |

For the Trigger One-to-One table of each series, see the device datasheets.

The following describes how to connect SCB channel 3 to P-DMA1 channel 15. Note that in a one-to-one group trigger, one input trigger is directly connected to a specific output trigger. Thus, specifying only the input trigger number is sufficient to indicate its output trigger.

1. Configuration register selection:

   MUX Group 8 (i=8)

   Input trigger SCB3_RX_TR_OUT, which directly connects to output trigger PDMA1_TR_IN[15] (k=7)

   Therefore, the register is specified as PERI_TR_1TO1_GR**8**_TR_CTL**7**.

2. Input trigger configuration:

   Typically, input trigger of one-to-one trigger is enabled.

   The configuration will be PERI_TR_1TO1_GR8_TR_CTL7.TR_SEL = **1**.

Table 8 shows the bit registers for one-to-one trigger that explains the inverted output trigger and the level or edge sensitive trigger. For more details, see the Registers TRM.

Table 8. Group Trigger Control Bit Registers

| Bit Register | Description |
|---|---|
| PERI_TR_1TO1_GR[i]_TR_CTL[k].TR_INV | Specifies if the output trigger is inverted.<br>‘0’: Not inverted<br>‘1’: Inverted |
| PERI_TR_1TO1_GR[i]_TR_CTL[k].TR_EDGE | Specifies if the (inverted) output trigger is treated as a level sensitive or edge sensitive trigger.<br>‘0’: Level sensitive<br>‘1’: Edge sensitive trigger |

## 2.3 Software Trigger

Each group supports a software trigger. For a group trigger, either an input trigger or output trigger can be selected for SW trigger. For one-to-one trigger, an output trigger can be selected for SW trigger. Figure 6 shows the group trigger with SW trigger block diagram. Figure 7 shows the one-to-one trigger with SW trigger block diagram.

Figure 6. SW Trigger in Group Trigger

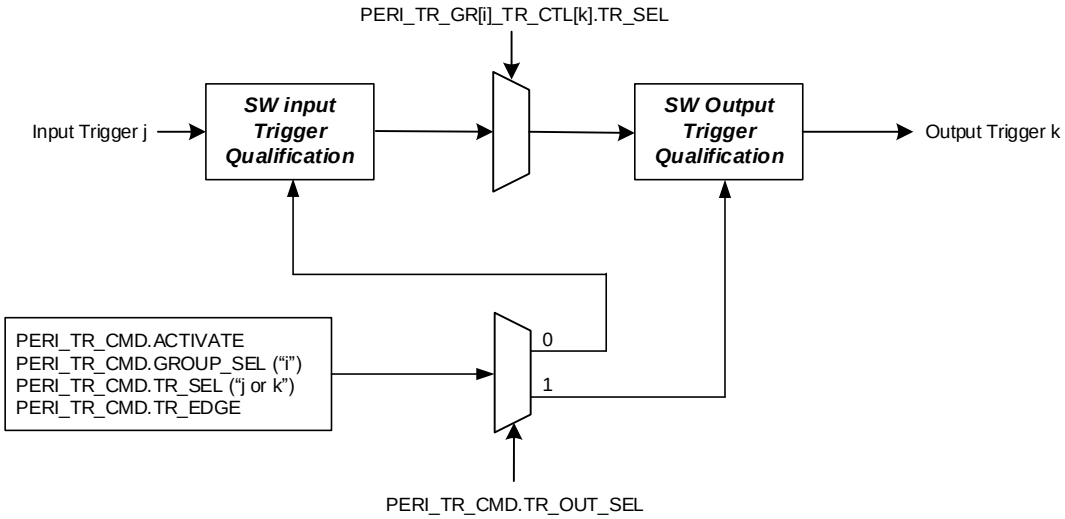
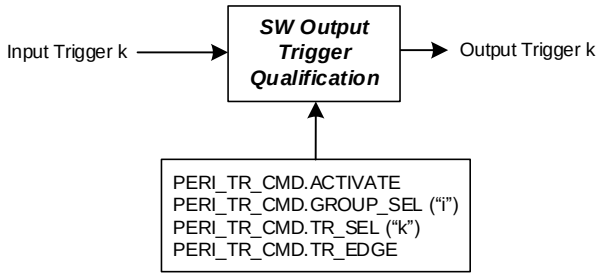



Figure 7. SW Trigger in One-to-one Trigger




The activation of a trigger through software can be made via PERI_TR_CMD. This register is useful for SW initiated triggers or for debugging purposes. For group trigger, to specify an activated trigger as input trigger (=j) or output trigger (=k), configure PERI_TR_CMD.OUT_SEL. If set to ‘0’, input trigger is selected. If set to ‘1’, output trigger is selected. Note that this configuration is not used for one-to-one trigger. The activated trigger number can be specified by PERI_TR_CMD.TR_SEL=j or k. The trigger group number can be specified by PERI_TR_CMD.GROUP_SEL=i. The level or edge sensitive trigger of the activated trigger can be specified by PERI_TR_CMD.TR_EDGE. PERI_TR_CMD.ACTIVATE can be set to '1' to activate a trigger. PERI_TR_CMD.ACTIVATE cannot be set together with the other PERI_TR_CMD field. This section describes an example of software trigger.

**Example:**

This is an example to trigger data transfer on P-DMA0 channel 0. This trigger routing is included in MUX Group 0 of group trigger and specifies the output trigger as the activated trigger. Table 9 lists the values that need to be set for PERI_TR_CMD.GROUP_SEL (=i) and PERI_TR_CMD.TR_SEL (=k). Table 9, extracted from the CYT2B7 series datasheet, provides trigger group and output trigger.

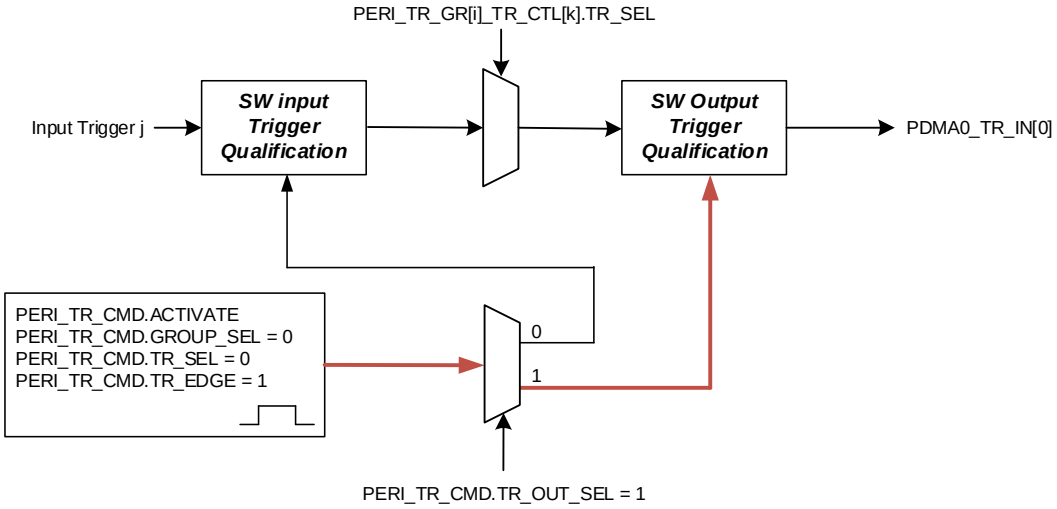
Table 9. Trigger Outputs of MUX Group 0 of CYT2B7 Series

| Output (“k”) | Trigger Label | Description |
|---|---|---|
| **MUX Group 0:** PDMA0_TR (P-DMA0 trigger multiplexer) | | |
| 0:7 | PDMA0_TR_IN[0:7] | Triggers to P-DMA0[0:7] |

For the Trigger Group Outputs table of each series, see the device datasheets.

Figure 8 shows the trigger of data transfer on P-DMA0 by SW.

Figure 8. P-DMA0 Data Transfer Start by SW




The following describes how to initiate data transfer on P-DMA0:

1. Activated trigger selection:

   PERI_TR_CMD.TR_SEL = **0**: Output trigger PDMA0_TR_IN[0] (k=0)

   PERI_TR_CMD.GROUP_SEL = **0**: MUX Group 0 of group trigger (i=0)

   PERI_TR_CMD.TR_EDGE = **1**: Edge sensitive trigger

2. Specifying activated trigger:

   PERI_TR_CMD.OUT_SEL = **1**: Output trigger

3. Activate SW trigger:

   PERI_TR_CMD.ACTIVATE = **1**

For details on Trigger Group Inputs and Trigger Group Outputs of each series, see the device datasheets.

# 3 Application

This section describes the following use cases for group trigger, one-to-one trigger, and SW trigger:

- Triggering P-DMA Transfer by TCPWM Timer: Group trigger use case
- Simultaneous ADC Conversion on Three SARs by Single TCPWM Timer: Group trigger use case
- Triggering ADC Conversion by TCPWM Timer: One-to-one trigger use case
- Simultaneous Starting of TCPWM Timer by SW Trigger: SW trigger use case

## 3.1 Triggering P-DMA Transfer by TCPWM Timer

This section explains how an input trigger in a group trigger initiates data transfer of a P-DMA0 channel.

### 3.1.1 Use Case Description of Triggering P-DMA Transfer by TCPWM Timer

Figure 9 shows an example application for group trigger in CYT2B7 series. Data transfer of P-DMA0 channel 9 is triggered by TCPWM 16-bit channel 0 on group trigger Multiplexer Group 3. TCPWM counter overflow event triggers P-DMA0 channel 9 to start its data transfer from the SRAM source buffer to the SRAM destination buffer. Figure 10 demonstrates the operation. For more details on TCPWM and DMA Components, see the “TCPWM” and “Direct Memory Access” chapters of the Architecture TRM.

Figure 9. Example of Group Trigger in CYT2B7 Series

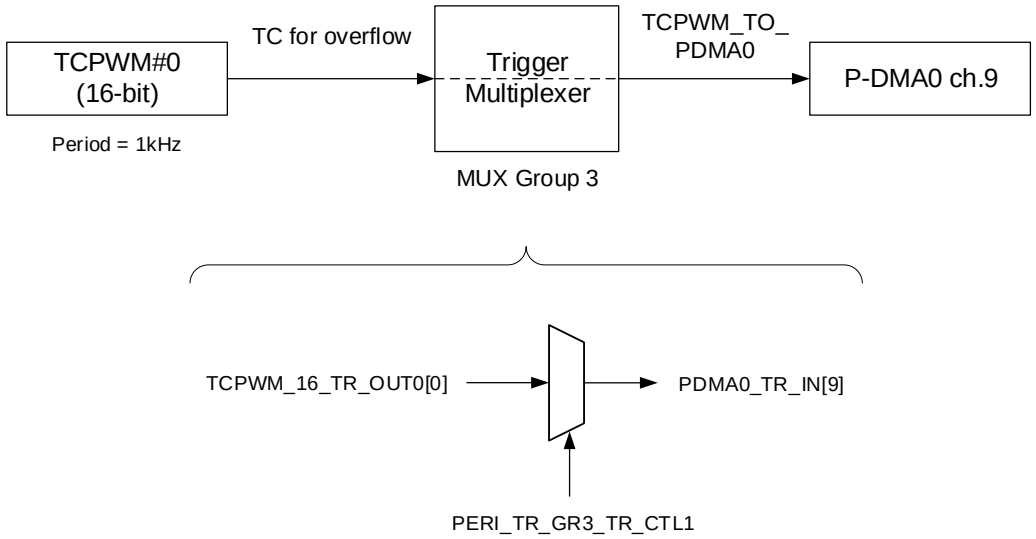
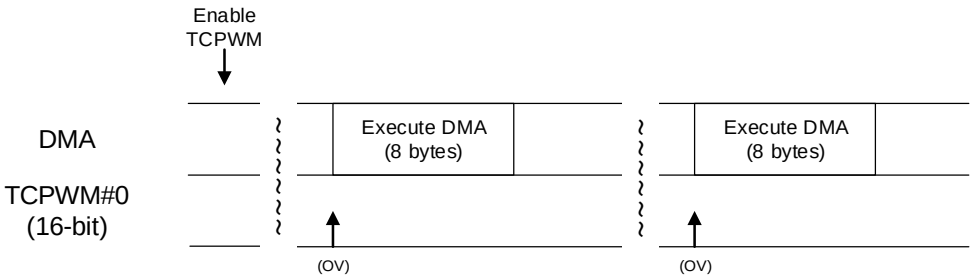
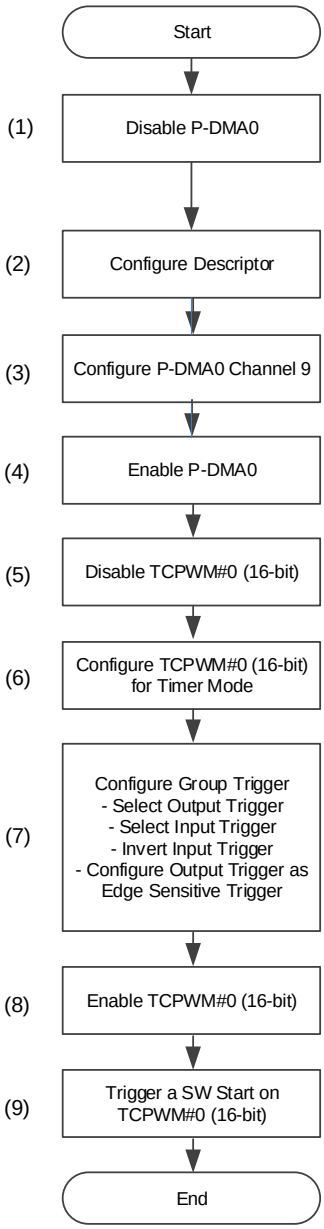



Figure 10. Example Operation: Triggering P-DMA Transfer by TCPWM Counter Overflow




Triggers must be set to implement this application. Figure 11 describes the procedure for setting the trigger. Figure 11 also provides the setting for TCPWM counter and P-DMA0 channel.

Figure 11. Example of Setting for Group Trigger in CYT2B7 Series




The configuration for output trigger and input trigger for group trigger in Step (7) are described as follows.

**Trigger Multiplexer Setting**

Table 10 and Table 11 list the values to be set. These tables are extracted from the CYT2B7 series datasheet that provides trigger group, input trigger, and output trigger.

Table 10. Trigger Outputs of MUX Group 3 of CYT2B7 Series

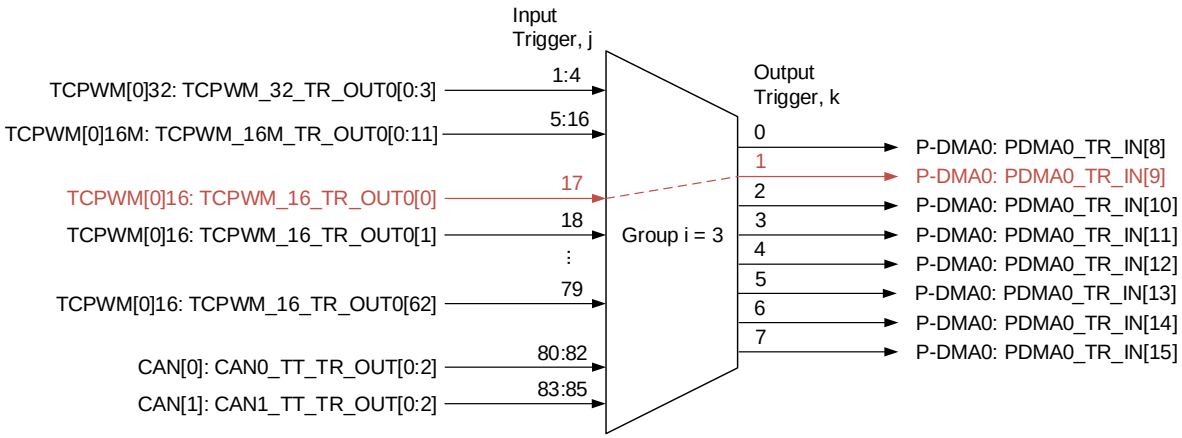
| Output ("k") | Trigger Label | Description |
|---|---|---|
| **MUX Group 3:** TCPWM_TO_PDMA0 (TCPWM0 to P-DMA0 trigger multiplexer) | | |
| 0:7 | PDMA0_TR_IN[8:15] | Triggers to P-DMA0[8:15] |

Table 11. Trigger Inputs of MUX Group 3 of CYT2B7 Series

| Input ("j") | Trigger Label | Description |
|---|---|---|
| **MUX Group 3:** TCPWM_TO_PDMA0 (TCPWM0 to P-DMA0 trigger multiplexer) | | |
| 1:4 | TCPWM_32_TR_OUT0[0:3] | 32-bit TCPWM0 counters |
| 5:16 | TCPWM_16M_TR_OUT0[0:11] | 16-bit Motor enhanced TCPWM0 counters |
| 17:79 | TCPWM_16_TR_OUT0[0:62] | 16-bit TCPWM0 counters |
| 80:82 | CAN0_TT_TR_OUT[0:2] | CAN0 TT Sync Outputs |
| 83:85 | CAN1_TT_TR_OUT[0:2] | CAN1 TT Sync Outputs |

Figure 12 shows the input trigger and output trigger structure.

Figure 12. Trigger Routing between 16-bit TCPWM and P-DMA0 of CYT2B7 Series




The following explains how to connect P-DMA0 channel 9 and 16-bit TCPWM channel 0:

1. Configuration register selection:

   MUX Group 3 (i=3)

   Output trigger PDMA0_TR_IN[9] (k=1)

   Therefore, the register is specified as PERI_TR_GR**3**_TR_CTL**1**.

2. Input trigger selection:

   Input trigger TCPWM_16_TR_OUT0[0] (j=17)

The configuration will be PERI_TR_GR3_TR_CTL1.TR_SEL = **17**. It is necessary to refer to Trigger Multiplexer diagram of each series to make sure the correct multiplexer group and accurate input and output trigger are selected. To find the trigger setting of group trigger of another MUX group and another series, see the device datasheets.

### 3.1.2 Example Program for Triggering P-DMA Transfer by TCPWM Timer

Code 1 shows the example program for Figure 11. The code snippets in this application note are part of the Sample Driver Library (SDL). See Other References for the SDL.

Code 1. Example for Triggering P-DMA Transfer by TCPWM Timer

```
void Cy_PDMA_Disable(volatile stc_DW_t *pstcPDMA)
{
    pstcPDMA->unCTL.stcField.u1ENABLED  =       0;
}

cy_en_pdma_status_t Cy_PDMA_Descr_Init(cy_stc_pdma_descr_t* descriptor, const cy_stc_pdma_descr_config_t*
config)
{
    cy_en_pdma_status_t retVal = CY_PDMA_ERR_UNC;

    if ((descriptor != NULL) && (config != NULL))
```

(1) Disable P-DMA0

(2) Configure Descriptor

```
    {
        descriptor->unPDMA_DESCR_CTL.stcField.u2WAIT_FOR_DEACT          =       config->deact;
        descriptor->unPDMA_DESCR_CTL.stcField.u2INTR_TYPE               =       config->intrType;
        descriptor->unPDMA_DESCR_CTL.stcField.u2TR_OUT_TYPE             =       config->trigoutType;
        descriptor->unPDMA_DESCR_CTL.stcField.u2TR_IN_TYPE              =       config->triginType;
        descriptor->unPDMA_DESCR_CTL.stcField.u1SRC_TRANSFER_SIZE       =       config->srcTxfrSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u1DST_TRANSFER_SIZE       =       config->destTxfrSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u1CH_DISABLE              =       config->chStateAtCmplt;
        descriptor->unPDMA_DESCR_CTL.stcField.u2DATA_SIZE               =       config->dataSize;
        descriptor->unPDMA_DESCR_CTL.stcField.u2DESCR_TYPE              =       config->descrType;

        descriptor->u32PDMA_DESCR_SRC                                   =       (uint32_t)config->srcAddr;
        descriptor->u32PDMA_DESCR_DST                                   =       (uint32_t)config->destAddr;

        switch(config->descrType)
        {
        case (uint32_t)CY_PDMA_1D_TRANSFER:
            {
                descriptor->unPDMA_DESCR_X_CTL.stcField.u12SRC_X_INCR   =       (uint32_t)config->srcXincr;
                descriptor->unPDMA_DESCR_X_CTL.stcField.u12DST_X_INCR   =       (uint32_t)config->destXincr;
                descriptor->unPDMA_DESCR_X_CTL.stcField.u8X_COUNT       =       (uint32_t)((config->xCount)
- 1u);
                descriptor->unPDMA_DESCR_Y_CTL.u32Register              =       (uint32_t)config->descrNext;
                break;
            }
        }
        retVal = CY_PDMA_SUCCESS;
    }
    return retVal;
}

cy_en_pdma_status_t Cy_PDMA_Chnl_Init(volatile stc_DW_t *pstcPDMA, uint32_t chNum, const
cy_stc_pdma_chnl_config_t* chnlConfig)
{
    cy_en_pdma_status_t retVal = CY_PDMA_ERR_UNC;

    if ((pstcPDMA != NULL) && (chnlConfig != NULL))
    {
        /* Set current descriptor */
        pstcPDMA->CH_STRUCT[chNum].unCH_CURR_PTR.u32Register            =  (uint32_t)chnlConfig-
>PDMA_Descriptor;
        /* Set if the channel is preemtable */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1PREEMPTABLE      =  chnlConfig->preemptable;
        /* Set channel priority */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u2PRIO             =  chnlConfig->priority;
        /* Set enabled status */
        pstcPDMA->CH_STRUCT[chNum].unCH_CTL.stcField.u1ENABLED          =  chnlConfig->enable;
        retVal = CY_PDMA_SUCCESS;
    }
    return (retVal);
}

void Cy_PDMA_Enable(volatile stc_DW_t *pstcPDMA)
{
    pstcPDMA->unCTL.stcField.u1ENABLED  =       1;
}

void Cy_Tcpwm_Counter_Disable(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unCTRL.stcField.u1ENABLED = 0x00;
}

uint32_t Cy_Tcpwm_Counter_Init(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM, cy_stc_tcpwm_counter_config_t const
*config)
{
    uint32_t status = CY_RET_BAD_PARAM;

    if ((NULL != ptscTCPWM) && (NULL != config))
    {
        ptscTCPWM->unCTRL.stcField.u1ONE_SHOT = config->runMode;

        ptscTCPWM->unCTRL.stcField.u2UP_DOWN_MODE = config-> countDirection;
        ptscTCPWM->unCTRL.stcField.u3MODE = config->CompareOrCapture;
        ptscTCPWM->unCTRL.stcField.u1DBG_FREEZE_EN = config->debug_pause;
        ptscTCPWM->unCTRL.stcField.u1AUTO_RELOAD_CC0 = config->enableCompare0Swap;
        ptscTCPWM->unDT.stcField.u8DT_LINE_OUT_L = config->clockPrescaler;

        if (CY_TCPWM_COUNTER_COUNT_UP == config->runMode)
        {
```

(3) Configure P-DMA0 Channel 9

(4) Enable P-DMA0

(5) Disable TCPWM#0 (16-bit)

(6) Configure TCPWM#0 (16-bit) for Timer Mode

```
            ptscTCPWM->unCOUNTER.u32Register = CY_TCPWM_CNT_UP_INIT_VAL;
        }

        ptscTCPWM->unCC0.u32Register = config->compare0;
        ptscTCPWM->unCC0_BUFF.u32Register = config->compare0_buff;
        ptscTCPWM->unPERIOD.u32Register = config->period;
        ptscTCPWM->unTR_IN_SEL0.stcField.u8CAPTURE0_SEL = config->capture0Input;
        ptscTCPWM->unTR_IN_SEL0.stcField.u8RELOAD_SEL = config->reloadInput;
        ptscTCPWM->unTR_IN_SEL0.stcField.u8STOP_SEL = config->stopInput;
        ptscTCPWM->unTR_IN_SEL0.stcField.u8COUNT_SEL = config->countInput;
        ptscTCPWM->unTR_IN_SEL1.stcField.u8START_SEL = config->startInput;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2CAPTURE0_EDGE = config->capture0InputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2RELOAD_EDGE = config->reloadInputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2START_EDGE = config->startInputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2STOP_EDGE = config->stopInputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2COUNT_EDGE = config->countInputMode;
        ptscTCPWM->unTR_OUT_SEL.stcField.u3OUT0 = config->trigger1;
        ptscTCPWM->unTR_OUT_SEL.stcField.u3OUT1 = config->trigger2;
        ptscTCPWM->unINTR_MASK.u32Register = config->interruptSources;
        ptscTCPWM->unCTRL.stcField.u1AUTO_RELOAD_CC1 = config->enableCompare1Swap;
        ptscTCPWM->unCC1.u32Register = config->compare1;
        ptscTCPWM->unCC1_BUFF.u32Register = config->compare1_buff;
        ptscTCPWM->unTR_IN_SEL1.stcField.u8CAPTURE1_SEL = config->capture1Input;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2CAPTURE1_EDGE = config->capture1InputMode;

        status = CY_RET_SUCCESS;
    }
    return(status);
}

cy_en_trigmux_status_t Cy_TrigMux_Connect(uint32_t inTrig, uint32_t outTrig, uint32_t invert, en_trig_type_t
trigType, uint32_t dbg_frz_en)
{
    volatile    stc_PERI_TR_GR_TR_CTL_field_t*  pTR_CTL;
                cy_en_trigmux_status_t          retVal = CY_TRIGMUX_BAD_PARAM;
    if ((inTrig & CY_TR_GROUP_MASK) == (outTrig & CY_TR_GROUP_MASK))
    {
        pTR_CTL             =   &(PERI->TR_GR[(outTrig & CY_TR_GROUP_MASK) >>
        CY_TR_GROUP_SHIFT].unTR_CTL[outTrig & CY_TR_MASK].stcField); //Select output trigger
        TRIG_OUT_MUX_3_PDMA0_TR_IN9
        pTR_CTL->u8TR_SEL   =   inTrig;   //Select input trigger TRIG_IN_MUX_3_TCPWM_16_TR_OUT00
        pTR_CTL->u1TR_INV   =   invert;   //Invert input trigger
        pTR_CTL->u1TR_EDGE  =   trigType; //Select edge sensitive trigger as output trigger type
        retVal              =   CY_TRIGMUX_SUCCESS;
    }
    return retVal;
}

void Cy_Tcpwm_Counter_Enable(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unCTRL.stcField.u1ENABLED = 0x1;
}

void Cy_Tcpwm_TriggerStart(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unTR_CMD.stcField.u1START = 0x01;
}
```

(7) Configure Group Trigger

(8) Enable TCPWM#0 (16-bit)

(9) Trigger a SW Start on TCPWM#0 (16-bit)

**Note:** The highlighted sections of the code snippet are not explained in this application note. For details, see the Architecture TRM.

## 3.2 Simultaneous ADC Conversion on Three SARs by Single TCPWM Timer

This section explains how an input trigger in a group trigger initiates ADC channel conversion on three SAR units.

### 3.2.1 Use Case Description of Simultaneous ADC Conversion on Three SARs by Single TCPWM Timer

Activation of multiple SAR ADC units, on ADC channel, can be triggered simultaneously by a single TCPWM timer. Generic trigger inputs are shared between ADC channels. One of the five generic triggers can be routed to any ADC channel. Figure 13 shows an example application in which ADC channel conversion of SAR0, SAR1, and SAR2 are triggered by TCPWM 16-bit Motor Control counter on group trigger Multiplexer Group 6. Compare match 0 on TCPWM counter generates a trigger to SAR ADC, which is connected to the generic triggers. The generic triggers are routed to ADC channel 3 of SAR0, SAR1, and SAR2, which initiates the simultaneous conversion.

Figure 14 demonstrates the operation. For more details on TCPWM and SAR ADC, see the “TCPWM” and “SAR ADC” chapters of the Architecture TRM.

Figure 13. Example of Group Trigger and ADC Generic Trigger Input in CYT2B7 Series

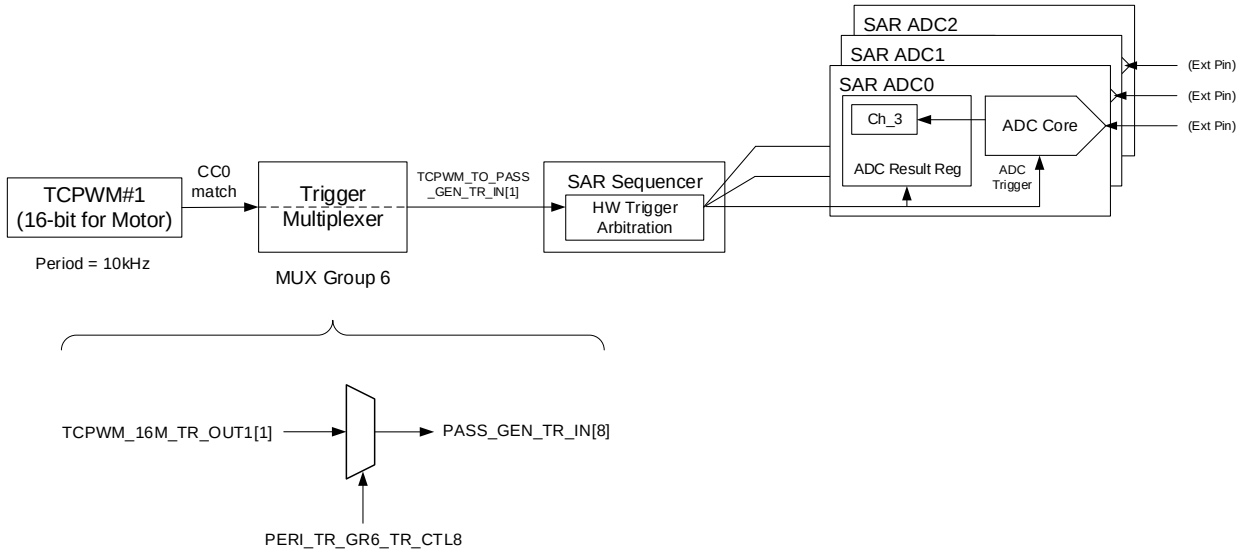
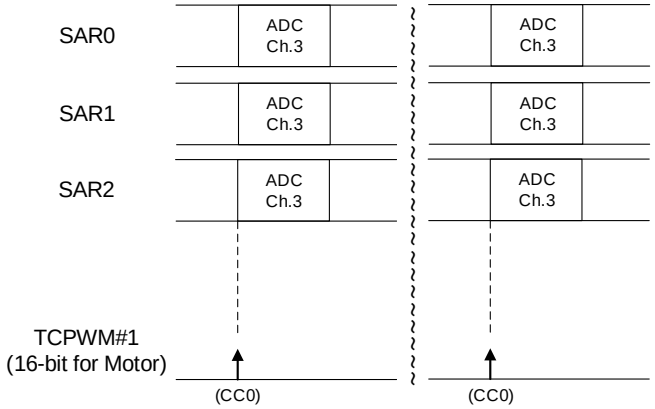
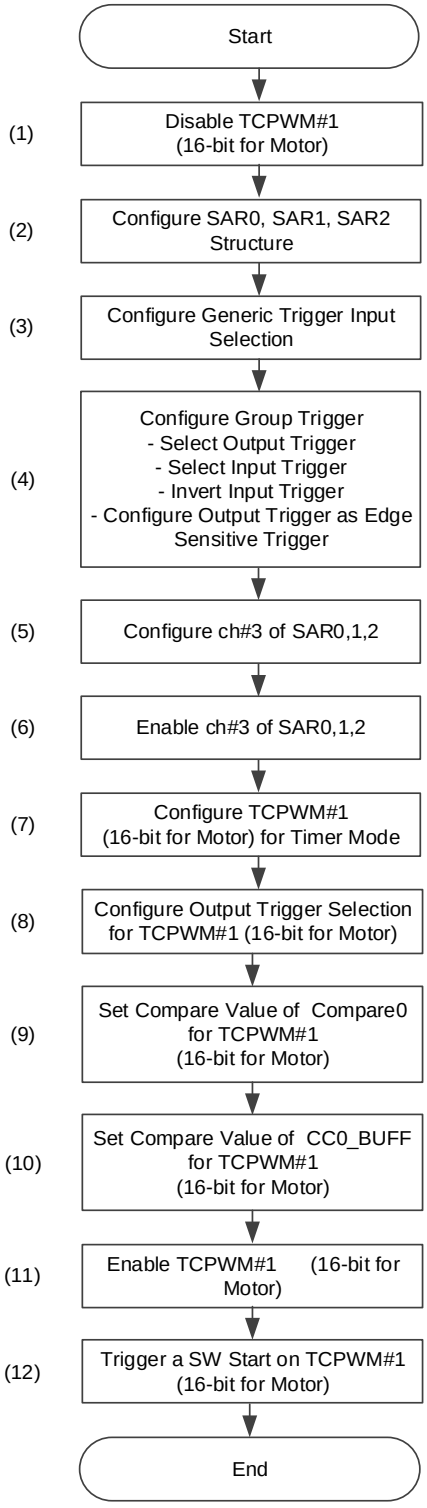



Figure 14. Example Operation: Simultaneous ADC Conversion on Three SARs by Single TCPWM Timer




Triggers must be set to implement this application. Figure 15 demonstrates the procedure for setting the trigger.

Figure 15. Setting for ADC Generic Trigger Input and Group Trigger in CYT2B7 Series




The configuration for the output trigger and input trigger for a group trigger in Step (4) are described as follows.

### Trigger Multiplexer Setting

Table 12 and Table 13 list the values to be set. These tables are extracted from the CYT2B7 series datasheet that provides trigger group, input trigger, and output trigger.

Table 12. Trigger Outputs of MUX Group 6 of CYT2B7 Series

| Output (“k”) | Trigger Label | Description |
|---|---|---|
| **MUX Group 6:** PASS (PASS SAR trigger multiplexer) | | |
| 0:11 | PASS_GEN_TR_IN[0:11] | Generic triggers to SAR ADCs |

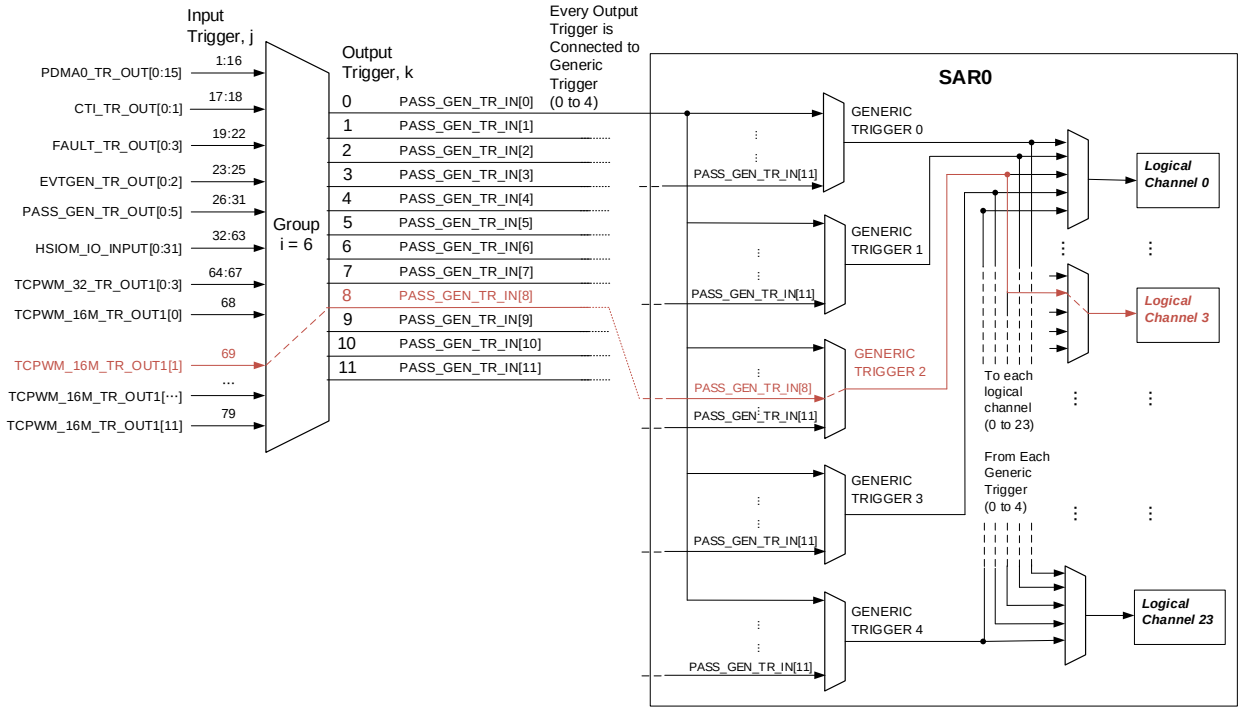
Table 13. Trigger Inputs of MUX Group 6 of CYT2B7 Series

| Input (“j”) | Trigger Label | Description |
|---|---|---|
| **MUX Group 6:** PASS (PASS SAR trigger multiplexer) | | |
| 1:16 | PDMA0_TR_OUT[0:15] | General purpose P-DMA0 triggers |
| 17:18 | CTI_TR_OUT[0:1] | Trace events |
| 19:22 | FAULT_TR_OUT[0:3] | Fault events |
| 23:25 | EVTGEN_TR_OUT[0:2] | EVTGEN triggers |
| 26:31 | PASS_GEN_TR_OUT[0:5] | PASS SAR done signals |
| 32:63 | HSIOM_IO_INPUT[0:31] | I/O inputs |
| 64:67 | TCPWM_32_TR_OUT1[0:3] | 32-bit TCPWM0 counters |
| 68:79 | TCPWM_16M_TR_OUT1[0:11] | 16-bit Motor enhanced TCPWM0 counters |

For Trigger Group Inputs and Trigger Group Outputs tables of each series, see the device datasheets.

Figure 16 shows the input trigger and output trigger structure.

Figure 16. Trigger Routing between TCPWM 16-bit Motor and Generic Trigger 2 in SAR0 Unit of CYT2B7 Series



The following explains how the trigger source TCPWM 16-bit Motor Control counter is routed to a generic trigger 2 of each SAR:

1. Configuration register selection:

   MUX Group 6 (i=6)

   Output trigger, generic trigger 8 or PASS_GEN_TR_IN[8] (k=8)

   Therefore, the register is specified as PERI_TR_GR**6**_TR_CTL**8**.

2. Input trigger selection:

   Input trigger TCPWM_16M_TR_OUT1[1] (j=69)

   The configuration will be PERI_TR_GR6_TR_CTL8.TR_SEL = **69**.

3. Generic trigger 8 is forwarded to SAR generic trigger input 2, IN2 of the corresponding SAR unit:

   - For SAR0, the configuration will be
     - SAR_TR_IN_SEL**0.**IN**2**_SEL = 8
   - For SAR1, the configuration will be
     - SAR_TR_IN_SEL**1.**IN**2**_SEL = 8
   - For SAR2, the configuration will be
     - SAR_TR_IN_SEL**2.**IN**2**_SEL = 8

4. IN2 triggers channel 3 of the corresponding SAR unit:

   - For SAR0, the configuration for SAR trigger select will be
     - PASS0_SAR0_CH**3**_TR_CTL.SEL = 4 (This value represents SAR Generic trigger input 2)
   - For SAR1, the configuration for SAR trigger select will be
     - PASS0_SAR1_CH**3**_TR_CTL.SEL = 4
   - For SAR2, the configuration for SAR trigger select will be
     - PASS0_SAR2_CH**3**_TR_CTL.SEL = 4

To find the trigger setting of the group trigger, see the device datasheets.

### 3.2.2 Example Program for Simultaneous ADC Conversion on Three SARs by Single TCPWM Timer

Code 2 shows the example program for Figure 15.

Code 2. Example of Simultaneous ADC Conversion on Three SARs by Single TCPWM Timer

```
void Cy_Tcpwm_Counter_Disable(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unCTRL.stcField.u1ENABLED = 0x00;
}

cy_en_adc_status_t Cy_Adc_Init(volatile stc_PASS_SAR_t * base, const cy_stc_adc_config_t * config)
{
    cy_en_adc_status_t ret = CY_ADC_SUCCESS;
    un_PASS_SAR_CTL_t unSarCtl = { 0 };
    if (NULL != config)
    {
        /* CTL register setting */
        base->unPRECOND_CTL.stcField.u4PRECOND_TIME = config->preconditionTime;

        /* CTL register setting */
        unSarCtl.stcField.u8PWRUP_TIME = config->powerupTime;
        unSarCtl.stcField.u1IDLE_PWRDWN = config->enableIdlePowerDown ? 1u : 0;
        unSarCtl.stcField.u1MSB_STRETCH = config->msbStretchMode;
        unSarCtl.stcField.u1HALF_LSB = config->enableHalfLsbConv ? 1u : 0;
        unSarCtl.stcField.u1SARMUX_EN = config->sarMuxEnable ? 1u : 0;
        unSarCtl.stcField.u1ADC_EN = config->adcEnable ? 1u : 0;
        unSarCtl.stcField.u1ENABLED = config->sarIpEnable ? 1u : 0;
        base->unCTL.u32Register = unSarCtl.u32Register;
    }
    else
    {
```

(1) Disable TCPWM#1 (16-bit for Motor)

(2) Configure SAR0, SAR1, SAR2 Structure

```
        ret = CY_ADC_BAD_PARAM;
    }
    return ret;
}

cy_en_adc_status_t Cy_Adc_SetGenericTriggerInput(volatile stc_PASS_EPASS_MMIO_t * base, uint8_t numOfAdc,
uint8_t triggerInputNumber, uint8_t genericTriggerValue)
{
    cy_en_adc_status_t ret = CY_ADC_SUCCESS;
    {
        switch(triggerInputNumber)
        {
        case 2:
            base->unSAR_TR_IN_SEL[numOfAdc].stcField.u4IN2_SEL = genericTriggerValue; //Select GENERIC
TRIGGER 2
            break;
        }
    }
    return ret;
}

cy_en_trigmux_status_t Cy_TrigMux_Connect(uint32_t inTrig, uint32_t outTrig, uint32_t invert, en_trig_type_t
trigType, uint32_t dbg_frz_en)
{
    volatile    stc_PERI_TR_GR_TR_CTL_field_t*  pTR_CTL;
                cy_en_trigmux_status_t          retVal = CY_TRIGMUX_BAD_PARAM;
    if ((inTrig & CY_TR_GROUP_MASK) == (outTrig & CY_TR_GROUP_MASK))
    {
        pTR_CTL             =   &(PERI->TR_GR[(outTrig & CY_TR_GROUP_MASK) >>
       CY_TR_GROUP_SHIFT].unTR_CTL[outTrig & CY_TR_MASK].stcField); //Select output trigger
       TRIG_OUT_MUX_6_PASS_GEN_TR_IN8
        pTR_CTL->u8TR_SEL   =   inTrig;   //Select input trigger TRIG_IN_MUX_6_TCPWM_16M_TR_OUT11
        pTR_CTL->u1TR_INV   =   invert;   //Invert input trigger
        pTR_CTL->u1TR_EDGE  =   trigType; // Select edge sensitive trigger as output trigger type
        retVal              =   CY_TRIGMUX_SUCCESS;
    }
    return retVal;
}

cy_en_adc_status_t Cy_Adc_Channel_Init(volatile stc_PASS_SAR_CH_t * base, const cy_stc_adc_channel_config_t
* config)
{
    cy_en_adc_status_t ret = CY_ADC_SUCCESS;
    un_PASS_SAR_CH_TR_CTL_t unTrCtl = { 0 };
    un_PASS_SAR_CH_SAMPLE_CTL_t unSampleCtl = { 0 };
    un_PASS_SAR_CH_POST_CTL_t unPostCtl = { 0 };
    un_PASS_SAR_CH_RANGE_CTL_t unRangeCtl = { 0 };
    un_PASS_SAR_CH_INTR_t unIntr = { 0 };

    if (NULL != config)
    {
        /* At first disable the channel */
        {
            base->unENABLE.stcField.u1CHAN_EN = 0u;
        }
        /* Clear whole interrupt flags */
        unIntr.stcField.u1CH_OVERFLOW = 1u;
        unIntr.stcField.u1CH_PULSE = 1u;
        unIntr.stcField.u1CH_RANGE = 1u;
        unIntr.stcField.u1GRP_CANCELLED = 1u;
        unIntr.stcField.u1GRP_DONE = 1u;
        unIntr.stcField.u1GRP_OVERFLOW = 1u;
        base->unINTR.u32Register = unIntr.u32Register;

        unTrCtl.stcField.u3SEL = config->triggerSelection;
        unTrCtl.stcField.u3PRIO = config->channelPriority;
        unTrCtl.stcField.u2PREEMPT_TYPE = config->preenptionType;
        unTrCtl.stcField.u1GROUP_END = config->isGroupEnd ? 1u : 0u;
        unTrCtl.stcField.u1DONE_LEVEL = config->doneLevel ? 1u : 0u;
        base->unTR_CTL.u32Register = unTrCtl.u32Register;

        unSampleCtl.stcField.u6PIN_ADDR = config->pinAddress;
        unSampleCtl.stcField.u2PORT_ADDR = config->portAddress;
        unSampleCtl.stcField.u3EXT_MUX_SEL = config->extMuxSelect;
        unSampleCtl.stcField.u1EXT_MUX_EN = config->extMuxEnable ? 1u : 0u;
        unSampleCtl.stcField.u2PRECOND_MODE = config->preconditionMode;
        unSampleCtl.stcField.u2OVERLAP_DIAG = config->overlapDiagMode;
        unSampleCtl.stcField.u12SAMPLE_TIME = config->sampleTime;
        unSampleCtl.stcField.u1ALT_CAL = config->calibrationValueSelect;
```

(3) Configure Generic Trigger Input Selection

(4) Configure Group Trigger

(5) Configure ch#3 of SAR0,1,2

```
        base->unSAMPLE_CTL.u32Register = unSampleCtl.u32Register;

        unPostCtl.stcField.u3POST_PROC = config->postProcessingMode;
        unPostCtl.stcField.u1LEFT_ALIGN = config->resultAlignment;
        unPostCtl.stcField.u1SIGN_EXT = config->signExtention;
        unPostCtl.stcField.u8AVG_CNT = config->averageCount;
        unPostCtl.stcField.u5SHIFT_R = config->rightShift;
        unPostCtl.stcField.u2RANGE_MODE = config->rangeDetectionMode;
        base->unPOST_CTL.u32Register = unPostCtl.u32Register;

        unRangeCtl.stcField.u16RANGE_LO = config->rangeDetectionLoThreshold;
        unRangeCtl.stcField.u16RANGE_HI = config->rangeDetectionHiThreshold;
        base->unRANGE_CTL.u32Register = unRangeCtl.u32Register;

        {
        cy_en_adc_status_t ret = CY_ADC_SUCCESS;
        un_PASS_SAR_CH_INTR_MASK_t unMask = { 0 };
        if (NULL != mask)
        {
            unMask.stcField.u1CH_OVERFLOW_MASK = mask->chOverflow ? 1u : 0u;
            unMask.stcField.u1CH_PULSE_MASK = mask->chPulse ? 1u : 0u;
            unMask.stcField.u1CH_RANGE_MASK = mask->chRange ? 1u : 0u;
            unMask.stcField.u1GRP_CANCELLED_MASK = mask->grpCancelled ? 1u : 0u;
            unMask.stcField.u1GRP_DONE_MASK = mask->grpDone ? 1u : 0u;
            unMask.stcField.u1GRP_OVERFLOW_MASK = mask->grpOverflow ? 1u : 0u;
            base->unINTR_MASK.u32Register = unMask.u32Register;
        }
        return ret;
    }
    return ret;
}

void Cy_Adc_Channel_Enable(volatile stc_PASS_SAR_CH_t * base)
{
    base->unENABLE.stcField.u1CHAN_EN = 1u;
}

uint32_t Cy_Tcpwm_Counter_Init(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM, cy_stc_tcpwm_counter_config_t const
*config)
{
    uint32_t status = CY_RET_BAD_PARAM;

    if (config->trigger1 > 0x04 || config->trigger2 > 0x04)
    {
        return status;
    }


    if ((NULL != ptscTCPWM) && (NULL != config))
    {
        ptscTCPWM->unCTRL.stcField.u1ONE_SHOT = config->runMode;
        ptscTCPWM->unCTRL.stcField.u2UP_DOWN_MODE = config-> countDirection;
        ptscTCPWM->unCTRL.stcField.u3MODE = config->CompareOrCapture;
        ptscTCPWM->unCTRL.stcField.u1DBG_FREEZE_EN = config->debug_pause;
        ptscTCPWM->unCTRL.stcField.u1AUTO_RELOAD_CC0 = config->enableCompare0Swap;
        ptscTCPWM->unDT.stcField.u8DT_LINE_OUT_L = config->clockPrescaler;

        if (CY_TCPWM_COUNTER_COUNT_UP == config->runMode)
        {
            ptscTCPWM->unCOUNTER.u32Register = CY_TCPWM_CNT_UP_INIT_VAL;
        }
        else if (CY_TCPWM_COUNTER_COUNT_DOWN == config->runMode)
        {
            ptscTCPWM->unCOUNTER.u32Register = config->period;
        }
        else
        {
            ptscTCPWM->unCOUNTER.u32Register = CY_TCPWM_CNT_UP_DOWN_INIT_VAL;
        }

        ptscTCPWM->unCC0.u32Register = config->compare0;
        ptscTCPWM->unCC0_BUFF.u32Register = config->compare0_buff;
        ptscTCPWM->unPERIOD.u32Register = config->period;
        ptscTCPWM->unTR_IN_SEL0.stcField.u8CAPTURE0_SEL = config->capture0Input;
        ptscTCPWM->unTR_IN_SEL0.stcField.u8RELOAD_SEL = config->reloadInput;
        ptscTCPWM->unTR_IN_SEL0.stcField.u8STOP_SEL = config->stopInput;
        ptscTCPWM->unTR_IN_SEL0.stcField.u8COUNT_SEL = config->countInput;
        ptscTCPWM->unTR_IN_SEL1.stcField.u8START_SEL = config->startInput;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2CAPTURE0_EDGE = config->capture0InputMode;
```

(6) Enable ch#3 of SAR0,1,2

(7) Configure TCPWM#1 (16-bit for Motor) for Timer Mode

```
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2RELOAD_EDGE = config->reloadInputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2START_EDGE = config->startInputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2STOP_EDGE = config->stopInputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2COUNT_EDGE = config->countInputMode;
        ptscTCPWM->unTR_OUT_SEL.stcField.u3OUT0 = config->trigger1;
        ptscTCPWM->unTR_OUT_SEL.stcField.u3OUT1 = config->trigger2;
        ptscTCPWM->unINTR_MASK.u32Register = config->interruptSources;
        ptscTCPWM->unCTRL.stcField.u1AUTO_RELOAD_CC1 = config->enableCompare1Swap;
        ptscTCPWM->unCC1.u32Register = config->compare1;
        ptscTCPWM->unCC1_BUFF.u32Register = config->compare1_buff;
        ptscTCPWM->unTR_IN_SEL1.stcField.u8CAPTURE1_SEL = config->capture1Input;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2CAPTURE1_EDGE = config->capture1InputMode;

        status = CY_RET_SUCCESS;
    }

    return(status);
}

    TCPWM0_GRP1_CNT1->unTR_OUT_SEL.stcField.u3OUT1 = CY_TCPWM_COUNTER_CC0_MATCH;

void Cy_Tcpwm_Counter_SetCompare0(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM,  uint32_t compare0)
{
    ptscTCPWM->unCC0.u32Register = compare0;
}

void Cy_Tcpwm_Counter_SetCompare0_Buff(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM,  uint32_t compare1)
{
    ptscTCPWM->unCC0_BUFF.u32Register = compare1;
}

void Cy_Tcpwm_Counter_Enable(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unCTRL.stcField.u1ENABLED = 0x1;
}

void Cy_Tcpwm_TriggerStart(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unTR_CMD.stcField.u1START = 0x01;
}
```

(8) Configure Output Trigger Selection for TCPWM#1 (16-bit for Motor)

(9) Set Compare Value of Compare0 for TCPWM#1 (16-bit for Motor)

(10) Set Compare Value of CC0_BUFF for TCPWM#1 (16-bit for Motor)

(11) Enable TCPWM#1 (16-bit for Motor)

(12) Trigger a SW Start on TCPWM#1 (16-bit for Motor)

**Note:** The highlighted sections of the code snippet are not explained in this application note. For details, see the Architecture TRM.

## 3.3 Triggering ADC Conversion by TCPWM Timer

This section explains how an input trigger in a one-to-one group trigger initiates ADC conversion.

### 3.3.1 Use Case Description of Triggering ADC Conversion by TCPWM Timer

Figure 17 shows an example application of one-to-one trigger in CYT2B7 series. ADC conversion of SAR0 is triggered by TCPWM 16-bit counter on one-to-one trigger Multiplexer Group 1. Compare match of TCPWM counter channel 0 and channel 1 trigger the conversion on channel 4 and 5 of SAR0, respectively. Figure 18 demonstrates the operation. For more details on TCPWM and SAR ADC, see the “TCPWM” and “SAR ADC” chapters of the Architecture TRM.

Figure 17. Example of One-to-One Trigger in CYT2B7 Series

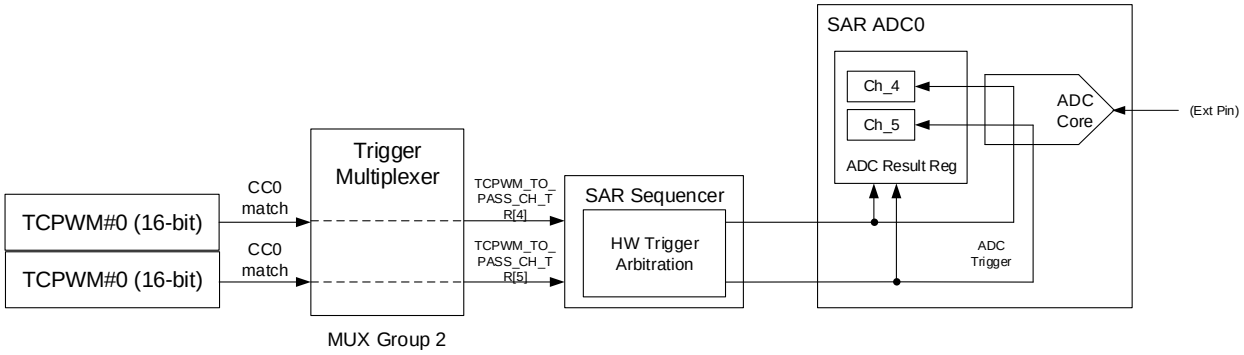
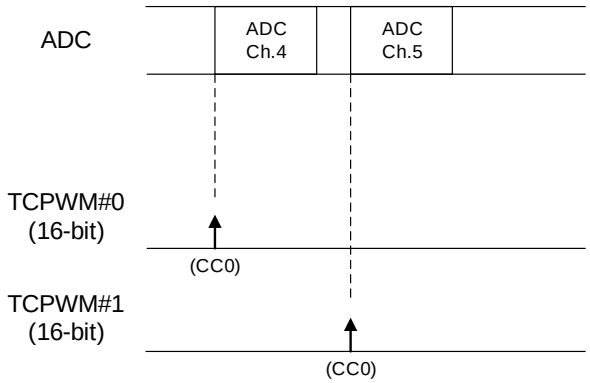
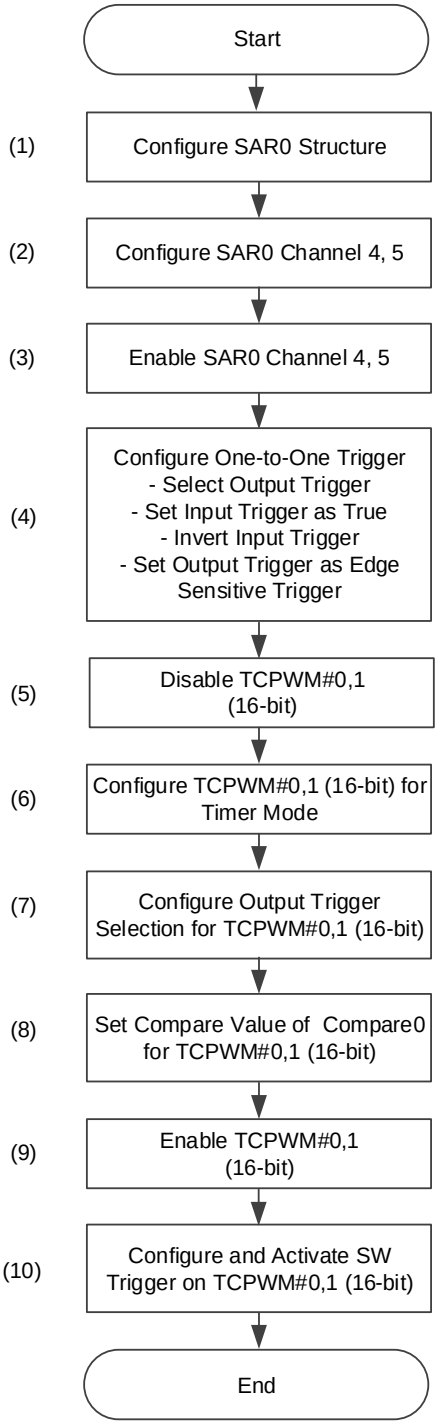



Figure 18. Example Operation: Triggering ADC Conversion by TCPWM Timer




Triggers must be set to implement this application. Figure 19 describes the procedure for setting the trigger. Figure 19 also provides the setting for TCPWM counter and SAR0 structure.

Figure 19. Setting for One-to-One Trigger in CYT2B7 Series




The configuration for the output trigger and input trigger of a one-to-one trigger in Step (4) are described as follows.

### Trigger Multiplexer Setting

Table 14 lists the values to be set. These tables are extracted from CYT2B7 series datasheet that provides the trigger group and input trigger.

Table 14. Trigger One-to-One MUX Group 1 of CYT2B7 Series

| Input trigger k | Trigger Inputs | Trigger Outputs |
|---|---|---|
| **MUX Group 1:** TCPWM0 to PASS SARx direct connect | | |
| 0 | TCPWM0_16M_TR_OUT1[0] | PASS0_CH_TR_IN[0] |
| 1 | TCPWM0_16M_TR_OUT1[1] | PASS0_CH_TR_IN[1] |
| 2 | TCPWM0_16M_TR_OUT1[2] | PASS0_CH_TR_IN[2] |
| 3 | TCPWM0_16M_TR_OUT1[3] | PASS0_CH_TR_IN[3] |
| 4:23 | TCPWM0_16_TR_OUT1[0:19] | PASS0_CH_TR_IN[4:23] |
| 24 | TCPWM0_16M_TR_OUT1[4] | PASS0_CH_TR_IN[32] |
| 25 | TCPWM0_16M_TR_OUT1[5] | PASS0_CH_TR_IN[33] |
| 26 | TCPWM0_16M_TR_OUT1[6] | PASS0_CH_TR_IN[34] |
| 27 | TCPWM0_16M_TR_OUT1[7] | PASS0_CH_TR_IN[35] |
| 28:55 | TCPWM0_16_TR_OUT1[20:47] | PASS0_CH_TR_IN[36:63] |
| 56 | TCPWM0_16M_TR_OUT1[8] | PASS0_CH_TR_IN[64] |
| 57 | TCPWM0_16M_TR_OUT1[9] | PASS0_CH_TR_IN[65] |
| 58 | TCPWM0_16M_TR_OUT1[10] | PASS0_CH_TR_IN[66] |
| 59 | TCPWM0_16M_TR_OUT1[11] | PASS0_CH_TR_IN[67] |
| 60:63 | TCPWM0_16_TR_OUT1[48:51] | PASS0_CH_TR_IN[68:71] |

The following describes how to connect SAR0 channel 4 and channel 5 to 16-bit TCPWM channel 0. Note that in a one-to-one group trigger, one input trigger is directly connected to a specific output trigger. Thus, specifying only the input trigger number is sufficient to indicate its output trigger.

1. Configuration register selection:

   MUX Group 1 (i=1)

   Input trigger TCPWM0_16_TR_OUT1[0], directly connects to output trigger PASS0_CH_TR_IN[4] (k=4)

   Input trigger TCPWM0_16_TR_OUT1[1], directly connects to output trigger PASS0_CH_TR_IN[5] (k=5)

   Therefore, the registers are specified as PERI_TR_1TO1_GR**1**_TR_CTL**4**, and PERI_TR_1TO1_GR**1**_TR_CTL**5**, respectively.

2. Input trigger configuration:

   Typically, input trigger of one-to-one trigger is enabled.

   The configurations are PERI_TR_1TO1_GR1_TR_CTL4.TR_SEL = **1**, and PERI_TR_1TO1_GR1_TR_CTL5.TR_SEL = **1**.

For more details on Trigger Input and Trigger Output, see the device datasheets.

### 3.3.2 Example Program for Triggering ADC Conversion by TCPWM Timer

Code 3 demonstrates the example program for Figure 19.

Code 3. Example of Triggering ADC Conversion by TCPWM Timer

```
cy_en_adc_status_t Cy_Adc_Init(volatile stc_PASS_SAR_t * base, const cy_stc_adc_config_t * config)
{
    cy_en_adc_status_t ret = CY_ADC_SUCCESS;
    un_PASS_SAR_CTL_t unSarCtl = { 0 };
    if (NULL != config)
    {
        /* CTL register setting */
        base->unPRECOND_CTL.stcField.u4PRECOND_TIME = config->preconditionTime;

        /* CTL register setting */
        unSarCtl.stcField.u8PWRUP_TIME = config->powerupTime;
        unSarCtl.stcField.u1IDLE_PWRDWN = config->enableIdlePowerDown ? 1u : 0;
        unSarCtl.stcField.u1MSB_STRETCH = config->msbStretchMode;
        unSarCtl.stcField.u1HALF_LSB = config->enableHalfLsbConv ? 1u : 0;
        unSarCtl.stcField.u1SARMUX_EN = config->sarMuxEnable ? 1u : 0;
        unSarCtl.stcField.u1ADC_EN = config->adcEnable ? 1u : 0;
        unSarCtl.stcField.u1ENABLED = config->sarIpEnable ? 1u : 0;
        base->unCTL.u32Register = unSarCtl.u32Register;
    }
    return ret;
}

cy_en_adc_status_t Cy_Adc_Channel_Init(volatile stc_PASS_SAR_CH_t * base, const cy_stc_adc_channel_config_t
* config)
{
    cy_en_adc_status_t ret = CY_ADC_SUCCESS;
    un_PASS_SAR_CH_TR_CTL_t unTrCtl = { 0 };
    un_PASS_SAR_CH_SAMPLE_CTL_t unSampleCtl = { 0 };
    un_PASS_SAR_CH_POST_CTL_t unPostCtl = { 0 };
    un_PASS_SAR_CH_RANGE_CTL_t unRangeCtl = { 0 };
    un_PASS_SAR_CH_INTR_t unIntr = { 0 };

    if (NULL != config)
    {
        /* At first disable the channel */
        base->unENABLE.stcField.u1CHAN_EN = 0u;

        /* Clear whole interrupt flags */
        unIntr.stcField.u1CH_OVERFLOW = 1u;
        unIntr.stcField.u1CH_PULSE = 1u;
        unIntr.stcField.u1CH_RANGE = 1u;
        unIntr.stcField.u1GRP_CANCELLED = 1u;
        unIntr.stcField.u1GRP_DONE = 1u;
        unIntr.stcField.u1GRP_OVERFLOW = 1u;
        base->unINTR.u32Register = unIntr.u32Register;

        unTrCtl.stcField.u3SEL = config->triggerSelection;
        unTrCtl.stcField.u3PRIO = config->channelPriority;
        unTrCtl.stcField.u2PREEMPT_TYPE = config->preenptionType;
        unTrCtl.stcField.u1GROUP_END = config->isGroupEnd ? 1u : 0u;
        unTrCtl.stcField.u1DONE_LEVEL = config->doneLevel ? 1u : 0u;
        base->unTR_CTL.u32Register = unTrCtl.u32Register;

        unSampleCtl.stcField.u6PIN_ADDR = config->pinAddress;
        unSampleCtl.stcField.u2PORT_ADDR = config->portAddress;
        unSampleCtl.stcField.u3EXT_MUX_SEL = config->extMuxSelect;
        unSampleCtl.stcField.u1EXT_MUX_EN = config->extMuxEnable ? 1u : 0u;
        unSampleCtl.stcField.u2PRECOND_MODE = config->preconditionMode;
        unSampleCtl.stcField.u2OVERLAP_DIAG = config->overlapDiagMode;
        unSampleCtl.stcField.u12SAMPLE_TIME = config->sampleTime;
        unSampleCtl.stcField.u1ALT_CAL = config->calibrationValueSelect;
        base->unSAMPLE_CTL.u32Register = unSampleCtl.u32Register;

        unPostCtl.stcField.u3POST_PROC = config->postProcessingMode;
        unPostCtl.stcField.u1LEFT_ALIGN = config->resultAlignment;
        unPostCtl.stcField.u1SIGN_EXT = config->signExtention;
        unPostCtl.stcField.u8AVG_CNT = config->averageCount;
        unPostCtl.stcField.u5SHIFT_R = config->rightShift;
        unPostCtl.stcField.u2RANGE_MODE = config->rangeDetectionMode;
        base->unPOST_CTL.u32Register = unPostCtl.u32Register;

        unRangeCtl.stcField.u16RANGE_LO = config->rangeDetectionLoThreshold;
        unRangeCtl.stcField.u16RANGE_HI = config->rangeDetectionHiThreshold;
```

(1) Configure SAR0 Structure

(2) Configure SAR0 Channel 4, 5

```
        base->unRANGE_CTL.u32Register = unRangeCtl.u32Register;

        {
        cy_en_adc_status_t ret = CY_ADC_SUCCESS;
        un_PASS_SAR_CH_INTR_MASK_t unMask = { 0 };
        if (NULL != mask)
        {
            unMask.stcField.u1CH_OVERFLOW_MASK = mask->chOverflow ? 1u : 0u;
            unMask.stcField.u1CH_PULSE_MASK = mask->chPulse ? 1u : 0u;
            unMask.stcField.u1CH_RANGE_MASK = mask->chRange ? 1u : 0u;
            unMask.stcField.u1GRP_CANCELLED_MASK = mask->grpCancelled ? 1u : 0u;
            unMask.stcField.u1GRP_DONE_MASK = mask->grpDone ? 1u : 0u;
            unMask.stcField.u1GRP_OVERFLOW_MASK = mask->grpOverflow ? 1u : 0u;
            base->unINTR_MASK.u32Register = unMask.u32Register;
        }
        return ret;
    }
    return ret;
}

void Cy_Adc_Channel_Enable(volatile stc_PASS_SAR_CH_t * base)
{
    base->unENABLE.stcField.u1CHAN_EN = 1u;
}
```

(3) Enable SAR0 Channel 4, 5

```
void Cy_TrigMux_Connect1To1T(uint32_t outTrig, uint32_t invert, en_trig_type_t trigType, uint32_t
dbg_frz_en)
{
    volatile    stc_PERI_TR_1TO1_GR_TR_CTL_field_t*  pTR_CTL;

        pTR_CTL             =   &(PERI->TR_1TO1_GR[(outTrig & CY_TR_GROUP_MASK) >>
       CY_TR_GROUP_SHIFT].unTR_CTL[outTrig & CY_TR_MASK].stcField); //Select output trigger
       TRIG_OUT_1TO1_1_TCPWM_TO_PASS_CH_TR4
        pTR_CTL->u1TR_SEL   =   1; //Set input trigger as true
        pTR_CTL->u1TR_INV   =   invert; //Invert input trigger
        pTR_CTL->u1TR_EDGE  =   trigType; //Select edge sensitive as trigger type
}
```

(4) Configure One-to-One Trigger

```
void Cy_Tcpwm_Counter_Disable(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unCTRL.stcField.u1ENABLED = 0x00;
}
```

(5) Disable TCPWM#0,1 (16-bit)

```
uint32_t Cy_Tcpwm_Counter_Init(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM, cy_stc_tcpwm_counter_config_t const
*config)
{
    uint32_t status = CY_RET_BAD_PARAM;

    if ((NULL != ptscTCPWM) && (NULL != config))
    {
        ptscTCPWM->unCTRL.stcField.u1ONE_SHOT = config->runMode;
        ptscTCPWM->unCTRL.stcField.u2UP_DOWN_MODE = config-> countDirection;
        ptscTCPWM->unCTRL.stcField.u3MODE = config->CompareOrCapture;
        ptscTCPWM->unCTRL.stcField.u1DBG_FREEZE_EN = config->debug_pause;
        ptscTCPWM->unCTRL.stcField.u1AUTO_RELOAD_CC0 = config->enableCompare0Swap;
        ptscTCPWM->unDT.stcField.u8DT_LINE_OUT_L = config->clockPrescaler;

        if (CY_TCPWM_COUNTER_COUNT_UP == config->runMode)
        {
            ptscTCPWM->unCOUNTER.u32Register = CY_TCPWM_CNT_UP_INIT_VAL;
        }

        ptscTCPWM->unCC0.u32Register = config->compare0;
        ptscTCPWM->unCC0_BUFF.u32Register = config->compare0_buff;
        ptscTCPWM->unPERIOD.u32Register = config->period;
        ptscTCPWM->unTR_IN_SEL0.stcField.u8CAPTURE0_SEL = config->capture0Input;
        ptscTCPWM->unTR_IN_SEL0.stcField.u8RELOAD_SEL = config->reloadInput;
        ptscTCPWM->unTR_IN_SEL0.stcField.u8STOP_SEL = config->stopInput;
        ptscTCPWM->unTR_IN_SEL0.stcField.u8COUNT_SEL = config->countInput;
        ptscTCPWM->unTR_IN_SEL1.stcField.u8START_SEL = config->startInput;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2CAPTURE0_EDGE = config->capture0InputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2RELOAD_EDGE = config->reloadInputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2START_EDGE = config->startInputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2STOP_EDGE = config->stopInputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2COUNT_EDGE = config->countInputMode;
        ptscTCPWM->unTR_OUT_SEL.stcField.u3OUT0 = config->trigger1;
        ptscTCPWM->unTR_OUT_SEL.stcField.u3OUT1 = config->trigger2;
        ptscTCPWM->unINTR_MASK.u32Register = config->interruptSources;
        ptscTCPWM->unCTRL.stcField.u1AUTO_RELOAD_CC1 = config->enableCompare1Swap;
```

(6) Configure TCPWM#0,1 (16-bit) for Timer Mode

```
        ptscTCPWM->unCC1.u32Register = config->compare1;
        ptscTCPWM->unCC1_BUFF.u32Register = config->compare1_buff;
        ptscTCPWM->unTR_IN_SEL1.stcField.u8CAPTURE1_SEL = config->capture1Input;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2CAPTURE1_EDGE = config->capture1InputMode;

        status = CY_RET_SUCCESS;
    }
    return(status);
}

void Cy_Tcpwm_Counter_SetTROUT(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unTR_OUT_SEL.stcField.u3OUT0 = 2;
    ptscTCPWM->unTR_OUT_SEL.stcField.u3OUT1 = 3;
}

void Cy_Tcpwm_Counter_SetCompare0(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM,  uint32_t compare0)
{
    ptscTCPWM->unCC0.u32Register = compare0;
}

void Cy_Tcpwm_Counter_Enable(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unCTRL.stcField.u1ENABLED = 0x1;
}
```

(7) Configure Output Trigger Selection for TCPWM#0,1 (16-bit)

(8) Set Compare Value of Compare0 for TCPWM#0,1 (16-bit)

(9) Enable TCPWM#0,1 (16-bit)

```
cy_en_trigmux_status_t Cy_TrigMux_SwTrigger(uint32_t trigLine, en_trig_type_t trigType, uint32_t outSel)
{
    cy_en_trigmux_status_t retVal = CY_TRIGMUX_INVALID_STATE;

    if (PERI->unTR_CMD.stcField.u1ACTIVATE == 0)
    {
        PERI->unTR_CMD.stcField.u8TR_SEL     = (trigLine & CY_TR_MASK) >> CY_TR_SHIFT; // Select activated
        trigger (0)
        PERI->unTR_CMD.stcField.u5GROUP_SEL  = (trigLine & CY_TR_GROUP_MASK) >> CY_TR_GROUP_SHIFT; // Select
        trigger group (4)
        PERI->unTR_CMD.stcField.u1TR_EDGE    = trigType; // Select edge sensitive trigger as trigger type
        PERI->unTR_CMD.stcField.u1OUT_SEL    = outSel; // Select activated trigger as output trigger
        PERI->unTR_CMD.stcField.u1ACTIVATE   = 1; // Activate trigger

        retVal = CY_TRIGMUX_SUCCESS;
    }
    return retVal;
}
```

(10) Configure and Activate SW Trigger on TCPWM#0,1 (16-bit)

**Note:** The highlighted sections of the code snippet are not explained in this application note. For details, see the Architecture TRM.

## 3.4 Simultaneous Starting of TCPWM Timer by SW Trigger

This section explains how a SW trigger simultaneously starts multiple TCPWM counters.

### 3.4.1 Use Case Description of Simultaneous Starting of TCPWM Timer by SW Trigger

Figure 20 shows an example application of SW trigger in CYT2B7 series. In 120-degree commutation control, three TCPWM 16-bit Motor Control counter ccchannel 0, 1, and 2 are triggered simultaneously by SW.

In this case, channel 0, channel 1, and channel 2 represent U-phase, V-phase, and W-phase, respectively. Figure 21 demonstrates the operation. The connection between trigger MUX Group 4 and TCPWM 16-bit motor control counter is illustrated in Figure 22. For more details on TCPWM, see the “TCPWM” chapter of the Architecture TRM.

Figure 20. Example of SW Trigger in CYT2B7 Series

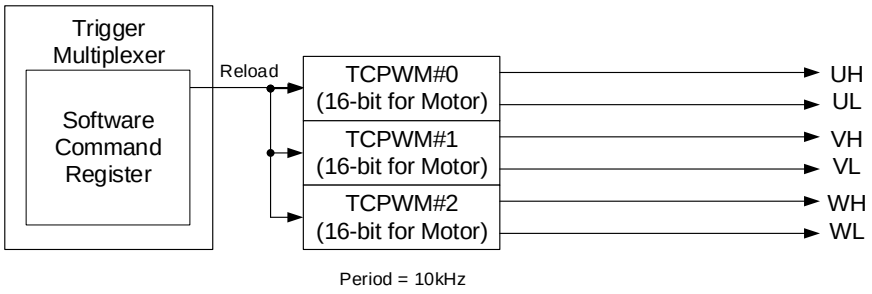



Figure 21. Example Operation: Simultaneous Starting of TCPWM Timer by SW Trigger

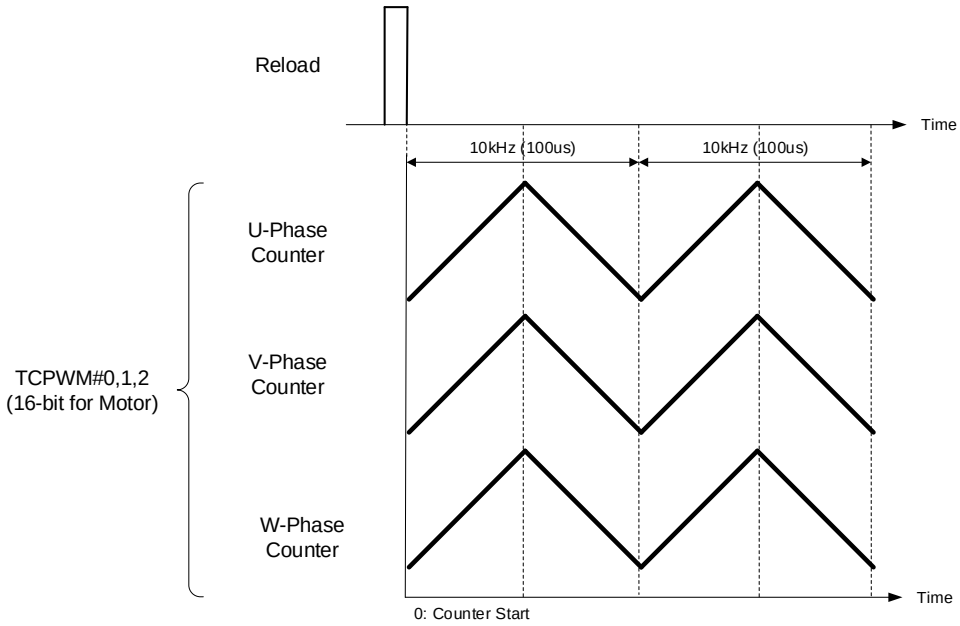
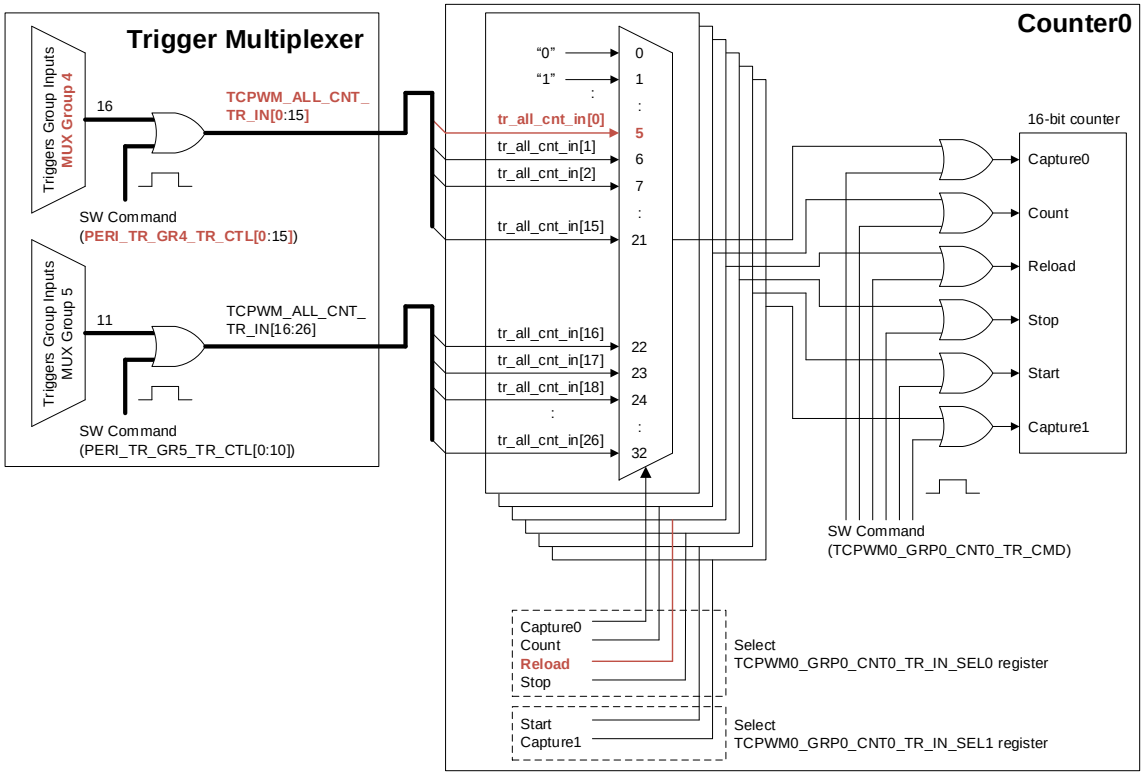
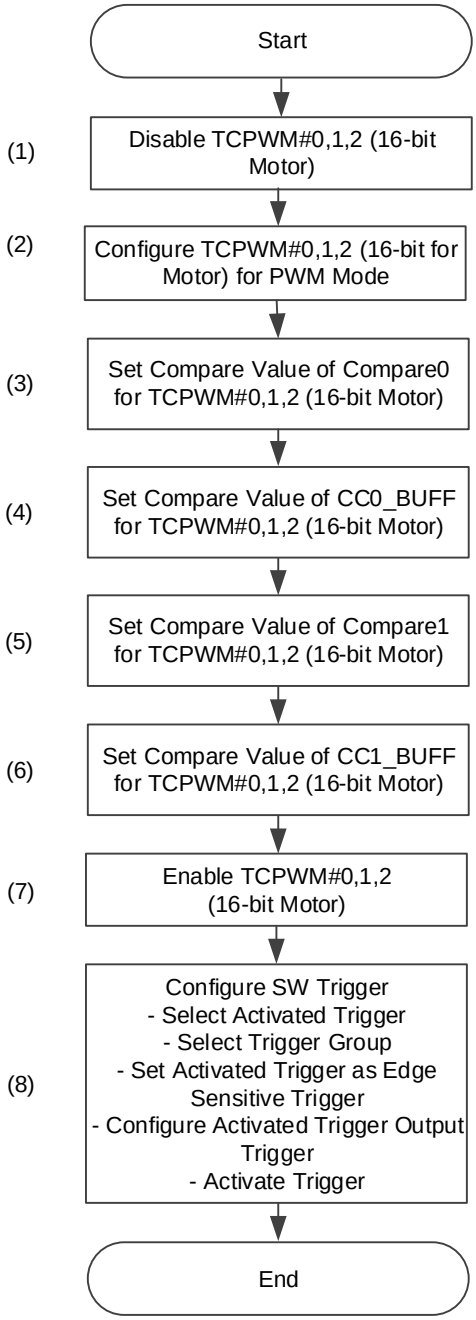


Figure 22. Connection between MUX Group 4 and TCPWM 16-bit Motor Control Counter




Triggers must be set to implement this application. Figure 23 lists the procedure for setting the trigger.

Figure 23. Setting for One-to-One Trigger in CYT2B7 Series




The configuration for SW trigger in Step (8) are described as follows.

**Trigger Multiplexer Setting**

Table 15 lists the values to be set for PERI_TR_CMD.GROUP_SEL (=i) and PERI_TR_CMD.TR_SEL (=k) can be found in. This table is extracted from CYT2B7 series datasheet that provides trigger group and output trigger.

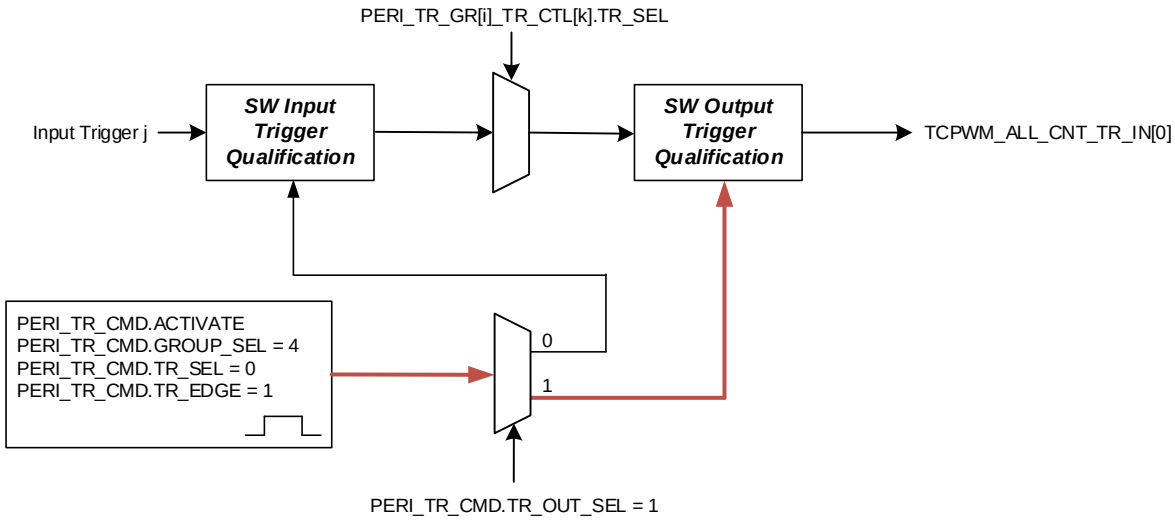
Table 15. Trigger Outputs of MUX Group 4 of CYT2B7 Series

| Output (“k”) | Trigger Label | Description |
|---|---|---|
| **MUX Group 4:** TCPWM_OUT (TCPWM0 to P-DMA0 trigger multiplexer) | | |
| 0:15 | TCPWM_ALL_CNT_TR_IN[0:15] | All counters trigger input |

For Trigger Group Outputs table of each series, see the device datasheets.

Figure 24 shows SW triggering the TCPWM counter.

Figure 24. SW Triggering TCPWM Counter




The following describes how to initiate the timer of TCPWM counter.

1. Activated trigger selection:

   PERI_TR_CMD.TR_SEL = **0**: Output trigger TCPWM_ALL_CNT_TR_IN[0] (k=0)

   PERI_TR_CMD.GROUP_SEL = **4**: MUX Group 4 of group trigger (i=4)

   PERI_TR_CMD.TR_EDGE = **1**: Edge sensitive trigger

2. Specifying activated trigger:

   PERI_TR_CMD.OUT_SEL = **1**: Output trigger

3. Activate SW trigger:

   PERI_TR_CMD.ACTIVATE = **1**

### 3.4.2 Example Program for Simultaneous TCPWM Timer Start by SW Trigger

Code 4 shows the example program for Figure 23.

Code 4. Example of Starting TCPWM Timer Simultaneous by SW Trigger

```
void Cy_Tcpwm_Counter_Disable(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unCTRL.stcField.u1ENABLED = 0x00;
}

uint32_t Cy_Tcpwm_Pwm_Init(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM, cy_stc_tcpwm_pwm_config_t const *config)
{
    uint32_t status = CY_RET_BAD_PARAM;

    if ((NULL != ptscTCPWM) && (NULL != config))
    {

        {
        ptscTCPWM->unCTRL.stcField.u1ONE_SHOT = config->runMode;
        ptscTCPWM->unCTRL.stcField.u2UP_DOWN_MODE = config-> countDirection;
        ptscTCPWM->unCTRL.stcField.u3MODE = config->pwmMode;
        ptscTCPWM->unCTRL.stcField.u1DBG_FREEZE_EN = config->debug_pause;
        ptscTCPWM->unCTRL.stcField.u1AUTO_RELOAD_CC0 = config->enableCompare0Swap;
        ptscTCPWM->unCTRL.stcField.u1AUTO_RELOAD_PERIOD = config->enablePeriodSwap;
        ptscTCPWM->unCTRL.stcField.u1AUTO_RELOAD_LINE_SEL = config->enableLineSelSwap;
        ptscTCPWM->unCTRL.stcField.u1PWM_SYNC_KILL = config->killMode;
        ptscTCPWM->unCTRL.stcField.u1PWM_STOP_ON_KILL = (config->killMode >> 1);
        }
        if(config->pwmMode == CY_TCPWM_PWM_MODE_DEADTIME)
        {
            ptscTCPWM->unDT.stcField.u8DT_LINE_OUT_L = config->deadTime;
        }
        if (CY_TCPWM_COUNTER_COUNT_UP == config->runMode)
        {
            ptscTCPWM->unCOUNTER.u32Register = CY_TCPWM_CNT_UP_INIT_VAL;
        }
        ptscTCPWM->unCC0.u32Register = config->compare0;
        ptscTCPWM->unCC0_BUFF.u32Register = config->compare0_buff;
        ptscTCPWM->unPERIOD.u32Register = config->period;
        ptscTCPWM->unPERIOD_BUFF.u32Register = config->period_buff;
        {
        ptscTCPWM->unTR_IN_SEL0.stcField.u8CAPTURE0_SEL = config->switchInput;
        ptscTCPWM->unTR_IN_SEL0.stcField.u8RELOAD_SEL = config->reloadInput;
        ptscTCPWM->unTR_IN_SEL0.stcField.u8STOP_SEL = config->kill0Input;
        ptscTCPWM->unTR_IN_SEL0.stcField.u8COUNT_SEL = config->countInput;
        }

        ptscTCPWM->unTR_IN_SEL1.stcField.u8START_SEL = config->startInput;

        {
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2CAPTURE0_EDGE = config->switchInputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2RELOAD_EDGE = config->reloadInputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2START_EDGE = config->startInputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2STOP_EDGE = config->kill0InputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2COUNT_EDGE = config->countInputMode;
        }

        ptscTCPWM->unINTR_MASK.u32Register = config->interruptSources;

        ptscTCPWM->unTR_PWM_CTRL.stcField.u2CC0_MATCH_MODE =  config->Cc0MatchMode;
        ptscTCPWM->unTR_PWM_CTRL.stcField.u2OVERFLOW_MODE = config->OverflowMode;
        ptscTCPWM->unTR_PWM_CTRL.stcField.u2UNDERFLOW_MODE = config->UnderflowMode;

       #if defined (tviibe1m) || defined (tviibe2m) || defined (tviibh4m)
        if( (ptscTCPWM == TCPWM0_GRP1_CNT0) || (ptscTCPWM == TCPWM0_GRP1_CNT1) || (ptscTCPWM ==
            TCPWM0_GRP1_CNT2) ||
            (ptscTCPWM == TCPWM0_GRP1_CNT3) || (ptscTCPWM == TCPWM0_GRP1_CNT4) || (ptscTCPWM ==
            TCPWM0_GRP1_CNT5) ||
            (ptscTCPWM == TCPWM0_GRP1_CNT6) || (ptscTCPWM == TCPWM0_GRP1_CNT7) || (ptscTCPWM ==
            TCPWM0_GRP1_CNT8) ||
            (ptscTCPWM == TCPWM0_GRP1_CNT9) || (ptscTCPWM == TCPWM0_GRP1_CNT10) || (ptscTCPWM ==
            TCPWM0_GRP1_CNT11) )

    {
        ptscTCPWM->unCTRL.stcField.u1AUTO_RELOAD_CC1 = config->enableCompare1Swap;
        ptscTCPWM->unTR_IN_SEL1.stcField.u8CAPTURE1_SEL = config->kill1Input;
        ptscTCPWM->unTR_IN_EDGE_SEL.stcField.u2CAPTURE1_EDGE = config->kill1InputMode;
        ptscTCPWM->unTR_PWM_CTRL.stcField.u2CC1_MATCH_MODE =  config->Cc1MatchMode;
```

(1) Disable TCPWM#0,1,2 (16-bit for Motor)

(2) Configure TCPWM#0,1,2 (16-bit for Motor) for PWM Mode

```
        ptscTCPWM->unDT.stcField.u16DT_LINE_COMPL_OUT = config->deadTimeComp;
        ptscTCPWM->unDT.stcField.u8DT_LINE_OUT_H = (config->deadTime >> 8);
    }
        status = CY_RET_SUCCESS;

    }

    return(status);
}

void Cy_Tcpwm_Counter_SetCompare0(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM,  uint32_t compare0)
{
    ptscTCPWM->unCC0.u32Register = compare0;
}

void Cy_Tcpwm_Counter_SetCompare0_Buff(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM,  uint32_t compare1)
{
    ptscTCPWM->unCC0_BUFF.u32Register = compare1;
}

void Cy_Tcpwm_Counter_SetCompare1(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM,  uint32_t compare0)
{
    ptscTCPWM->unCC1.u32Register = compare0;
}

void Cy_Tcpwm_Counter_SetCompare1_Buff(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM,  uint32_t compare1)
{
    ptscTCPWM->unCC1_BUFF.u32Register = compare1;
}

void Cy_Tcpwm_Pwm_Enable(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unCTRL.stcField.u1ENABLED = 0x1;
}

cy_en_trigmux_status_t Cy_TrigMux_SwTrigger(uint32_t trigLine, en_trig_type_t trigType, uint32_t outSel)
{
    cy_en_trigmux_status_t retVal = CY_TRIGMUX_INVALID_STATE;

    if (PERI->unTR_CMD.stcField.u1ACTIVATE == 0)
    {
        PERI->unTR_CMD.stcField.u8TR_SEL     = (trigLine & CY_TR_MASK) >> CY_TR_SHIFT; // Select activated
        trigger (0)
        PERI->unTR_CMD.stcField.u5GROUP_SEL  = (trigLine & CY_TR_GROUP_MASK) >> CY_TR_GROUP_SHIFT; // Select
        trigger group (4)
        PERI->unTR_CMD.stcField.u1TR_EDGE    = trigType; // Select edge sensitive trigger as trigger type
        PERI->unTR_CMD.stcField.u1OUT_SEL    = outSel; // Select activated trigger as output trigger
        PERI->unTR_CMD.stcField.u1ACTIVATE   = 1; // Activate trigger

        retVal = CY_TRIGMUX_SUCCESS;
    }
    return retVal;
}
```

(3) Set Compare Value of Compare0 for TCPWM#0,1,2 (16-bit for Motor)

(4) Set Compare Value of CC0_BUFF for TCPWM#0,1,2 (16-bit for Motor)

(5) Set Compare Value of Compare1 for TCPWM#0,1,2 (16-bit for Motor)

(6) Set Compare Value of CC1_BUFF for TCPWM#0,1,2 (16-bit for Motor)

(7) Enable TCPWM#0,1,2 (16-bit for Motor)

(8) Configure SW Trigger

**Note:** The highlighted sections of the code snippet are not explained in this application note. For details, see the Architecture TRM.

# 4 Glossary

| Terms | Description |
|---|---|
| TCPWM | Timer, Counter, and Pulse Width Modulator |
| SCB | Serial Communications Block |
| SW | Software |
| P-DMA | Peripheral Direct Memory Access |
| SAR ADC | Successive Approximation Register Analog-to-Digital Converter |
| MUX | Multiplexer |
| OV | Overflow |
| UH | U-phase PWM waveform output |
| UL | U-phase PWM complementary waveform output |
| VH | V-phase PWM waveform output |
| VL | V-phase PWM complementary waveform output |
| WH | W-phase PWM waveform output |
| WL | W-phase PWM complementary waveform output |

# 5 Related Documents

The following are the Traveo II family series datasheets and technical reference manuals. Contact Technical Support to obtain these documents.

- Device datasheet
  - CYT2B7 Datasheet 32-Bit Arm® Cortex®-M4F Microcontroller Traveo™ II Family
  - CYT2B9 Datasheet 32-Bit Arm® Cortex®-M4F Microcontroller Traveo™ II Family
  - CYT4BF Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo™ II Family
  - CYT4DN Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo™ II Family
- CYT2B Series
  - Traveo™ II Automotive Body Controller Entry Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B7
  - Traveo™ II Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B9
- CYT4B Series
  - Traveo™ II Automotive Body Controller High Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Body Controller High Registers Technical Reference Manual (TRM)
- CYT4D Series
  - Traveo™ II Automotive Cluster 2D Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Cluster 2D Registers Technical Reference Manual (TRM)

# 6 Other References

Cypress provides the Sample Driver Library (SDL) including startup as sample software to access various peripherals. SDL also serves as a reference to customers, for drivers that are not covered by the official AUTOSAR products. SDL cannot be used for production purposes as it has not been developed using any automotive SW development process. The code snippets in this application note are part of the SDL. Contact Technical Support to obtain the SDL.

## Document History

Document Title: AN228104 - How to Use Trigger Multiplexer in Traveo II Family

Document Number: 002-28104

| Revision | ECN | Submission Date | Description of Change |
|---|---|---|---|
| ** | 6825564 | 03/27/2020 | New application note. |

## Worldwide Sales and Design Support

Cypress maintains a worldwide network of offices, solution centers, manufacturer's representatives, and distributors. To find the office closest to you, visit us at Cypress Locations.

## Products

| | |
|---|---|
| Arm® Cortex® Microcontrollers | cypress.com/arm |
| Automotive | cypress.com/automotive |
| Clocks & Buffers | cypress.com/clocks |
| Interface | cypress.com/interface |
| Internet of Things | cypress.com/iot |
| Memory | cypress.com/memory |
| Microcontrollers | cypress.com/mcu |
| PSoC | cypress.com/psoc |
| Power Management ICs | cypress.com/pmic |
| Touch Sensing | cypress.com/touch |
| USB Controllers | cypress.com/usb |
| Wireless Connectivity | cypress.com/wireless |

## PSoC® Solutions

PSoC 1 | PSoC 3 | PSoC 4 | PSoC 5LP | PSoC 6 MCU

## Cypress Developer Community

Community | Code Examples | Projects | Videos | Blogs | Training | Components

## Technical Support

cypress.com/support



Cypress Semiconductor
198 Champion Court
San Jose, CA 95134-1709