Trigger Multiplexer usage in XMC7000 family

About this document

Scope and purpose

This application note describes trigger multiplexer for the XMC7000 family and explains how to route trigger signals from the source peripherals to the destinations.

Associated part family

XMC7000 family of XMC™ industrial microcontrollers

Table of contents

1 Introduction

Every peripheral in the XMC7000 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 reference manual [2].

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.

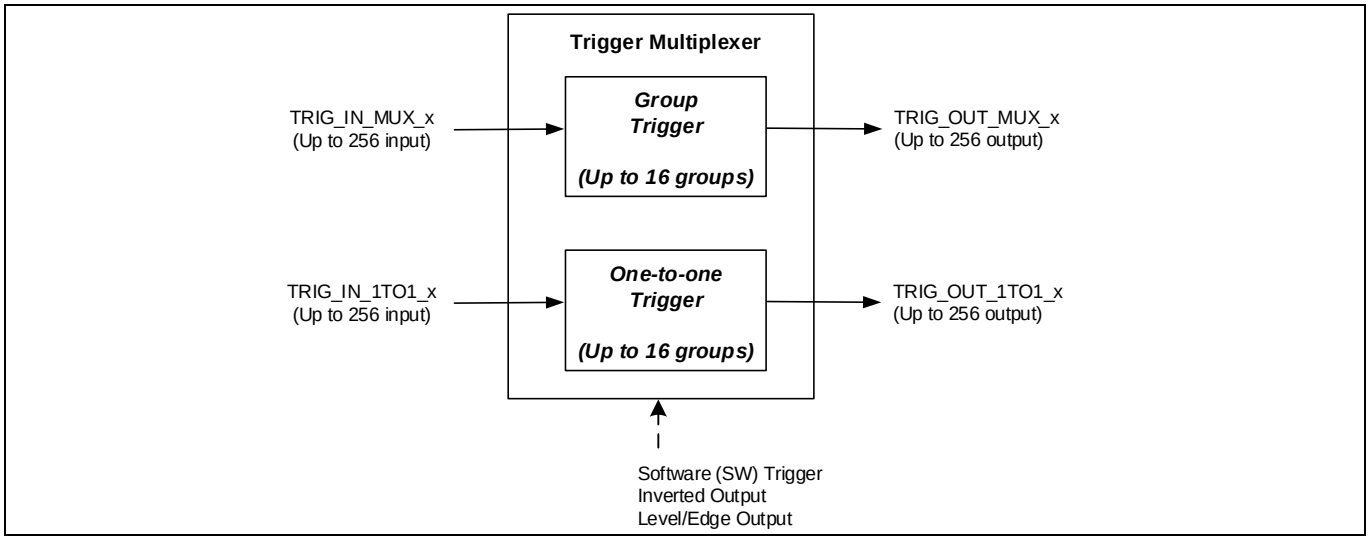

Figure 1 Trigger multiplexer block diagram

Trigger multiplexer has the following input and output signals:

- TRIG_IN_MUX_x is the input trigger of the group trigger. It has up to 256 input signals
- TRIG_IN_1TO1_x is the input trigger of the one-to-one trigger. It has up to 256 input signals
- TRIG_OUT_MUX_x is the output trigger of the group trigger. It has up to 256 output signals
- TRIG_OUT_1TO1_x is the output trigger of the 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 the XMC7000 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 [1].

Trigger multiplexer overview

Table 1 Routing between output trigger and input trigger of group trigger by series

TRIG_OUT_MUX_x	TRIG_IN_MUX_x	XMC7000
PDMA	PDMA	✓
	MDMA	✓
	FAULT	✓
	CTI	✓
	EVTGEN	✓
	HSIOM	✓
	TCPWM	✓
	CAN0_TT	✓
	CAN1_TT	✓
	PASS_GEN	✓
MDMA	MDMA	
	TCPWM	✓
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	
PASS_GEN	PDMA	✓
	CTI	
	FAULT	
	EVTGEN	✓

Trigger multiplexer overview

TRIG_OUT_MUX_x	TRIG_IN_MUX_x	XMC7000
	PASS_GEN	
	HSIOM	✓
	TCPWM	✓
CAN_TT	CAN_TT	✓
HSIOM PERI_DEBUG_FREEZE PASS_DEBUG_FREEZE SRSS_WDT_DEBUG_ FREEZE SRSS_MCWDT_DEBUG_FREEZE TCPWM_DEBUG_ FREEZE	TR_GROUP[i]_OUTPUT	✓
I2S_DEBUG_FREEZE TDM_DEBUG_FREEZE SG_DEBUG_FREEZE PWM_DEBUG_FREEZE PWM_DEBUG_FREEZE MIXER_DEBUG_FREEZE AUDIODAC_DEBUG_ FREEZE	TR_GROUP[i]_OUTPUT	
TR_GROUP[i]_INPUT	PDMA	✓
	SCB	✓
	SCB_I2C_SCL	✓
	CAN_DBG	✓
	CAN_FIFO	✓
	CAN_TT	✓
	CTI	✓
	FAULT	✓
	TCPWM	✓
	MDMA	✓
	PASS_GEN	✓
	EVTGEN	✓
	CXPI	
	SMIF	✓
	I2S	✓
	HSIOM	✓
	TDM	
	SG	
	PWM	
	AUDIODAC	
	MIXER	

Trigger multiplexer overview

In a one-to-one group trigger, one input trigger is directly connected to a specific output trigger. For the routing of the one-to-one trigger, see the datasheets [1].

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: All output triggers in a group "i" share the same input triggers.

Figure 2 shows the group trigger block diagram.

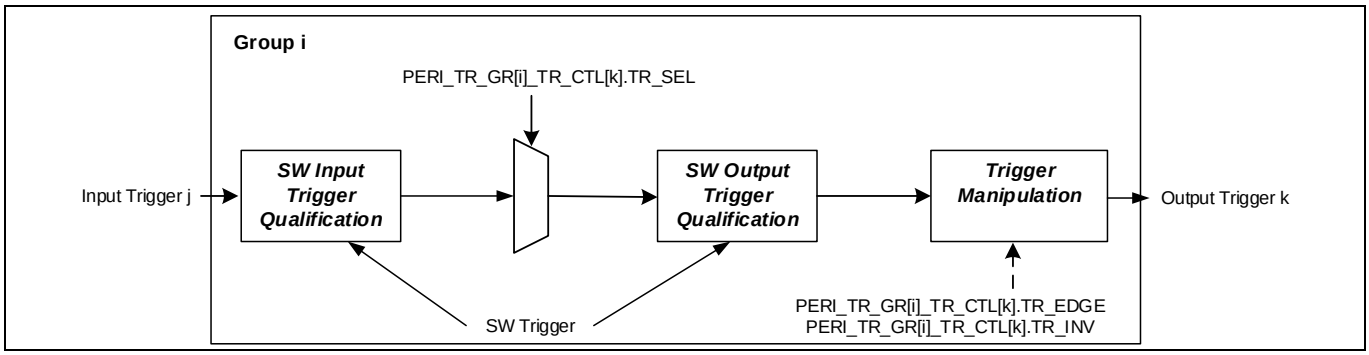

Figure 2 Group trigger

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.

Trigger multiplexer overview

Example 1:

This example demonstrates that the trigger of the 16-bit TCPWM counter starts upon receiving the SCB RX pulse. This trigger routing is included in Multiplexer (MUX) Group 6. This example is extracted from the XMC7200 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 the XMC7200 datasheet respectively.

Table 2 Trigger outputs of MUX Group 6 of XMC7200

Output (“k”)	Trigger label	Description
MUX Group 6: TCPWM1 Trigger Multiplexer		
0:28	TCPWM1_ALL_CNT_TR_IN[12:40]	Triggers to TCPWM1

Table 3 Trigger inputs of MUX Group 6 of XMC7200

Input(“j”)	Trigger	Description
MUX Group 6: TCPWM1 Trigger Multiplexer		
1:16	TCPWM1_16_TR_OUT1[0:15]	16-bit TCPWM1 counters
17	SCB_TX_TR_OUT[0]	SCB0 TX trigger
18	SCB_RX_TR_OUT[0]	SCB0 RX trigger
19	SCB_I2C_SCL_TR_OUT[0]	SCB0 I2C trigger
20	SCB_TX_TR_OUT[1]	SCB1 TX trigger
21	SCB_RX_TR_OUT[1]	SCB1 RX trigger
22	SCB_I2C_SCL_TR_OUT[1]	SCB1 I2C trigger
23	SCB_TX_TR_OUT[2]	SCB2 TX trigger
24	SCB_RX_TR_OUT[2]	SCB2 RX trigger
25	SCB_I2C_SCL_TR_OUT[2]	SCB2 I2C trigger
26	SCB_TX_TR_OUT[3]	SCB3 TX trigger
27	SCB_RX_TR_OUT[3]	SCB3 RX trigger
28	SCB_I2C_SCL_TR_OUT[3]	SCB3 I2C trigger
29	SCB_TX_TR_OUT[4]	SCB4 TX trigger
30	SCB_RX_TR_OUT[4]	SCB4 RX trigger
31	SCB_I2C_SCL_TR_OUT[4]	SCB4 I2C trigger
32	SCB_TX_TR_OUT[5]	SCB5 TX trigger
33	SCB_RX_TR_OUT[5]	SCB5 RX trigger
34	SCB_I2C_SCL_TR_OUT[5]	SCB5 I2C trigger
35	SCB_TX_TR_OUT[6]	SCB6 TX trigger
36	SCB_RX_TR_OUT[6]	SCB6 RX trigger
37	SCB_I2C_SCL_TR_OUT[6]	SCB6 I2C trigger
38	SCB_TX_TR_OUT[7]	SCB7 TX trigger
39	SCB_RX_TR_OUT[7]	SCB7 RX trigger
40	SCB_I2C_SCL_TR_OUT[7]	SCB7 I2C trigger
41	SCB_TX_TR_OUT[8]	SCB8 TX trigger
42	SCB_RX_TR_OUT[8]	SCB8 RX trigger

Trigger multiplexer overview

Input("j")	Trigger	Description
MUX Group 6: TCPWM1 Trigger Multiplexer		
43	SCB_I2C_SCL_TR_OUT[8]	SCB8 I2C trigger
44	SCB_TX_TR_OUT[9]	SCB9 TX trigger
45	SCB_RX_TR_OUT[9]	SCB9 RX trigger
46	SCB_I2C_SCL_TR_OUT[9]	SCB9 I2C trigger
47	SCB_TX_TR_OUT[10]	SCB10 TX trigger
48	SCB_RX_TR_OUT[10]	SCB10 RX trigger
49	SCB_I2C_SCL_TR_OUT[10]	SCB10 I2C trigger
52:57	PASS_GEN_TR_OUT[0:5]	PASS SAR ADC events
58:105	HSIOM_IO_INPUT[0:47]	I/O Inputs
106:107	CTI_TR_IN[0:1]	CPUSS CTI Trace events
108:111	FAULT_TR_OUT[0:3]	Fault events

Figure 3 shows the input trigger and output trigger structure.

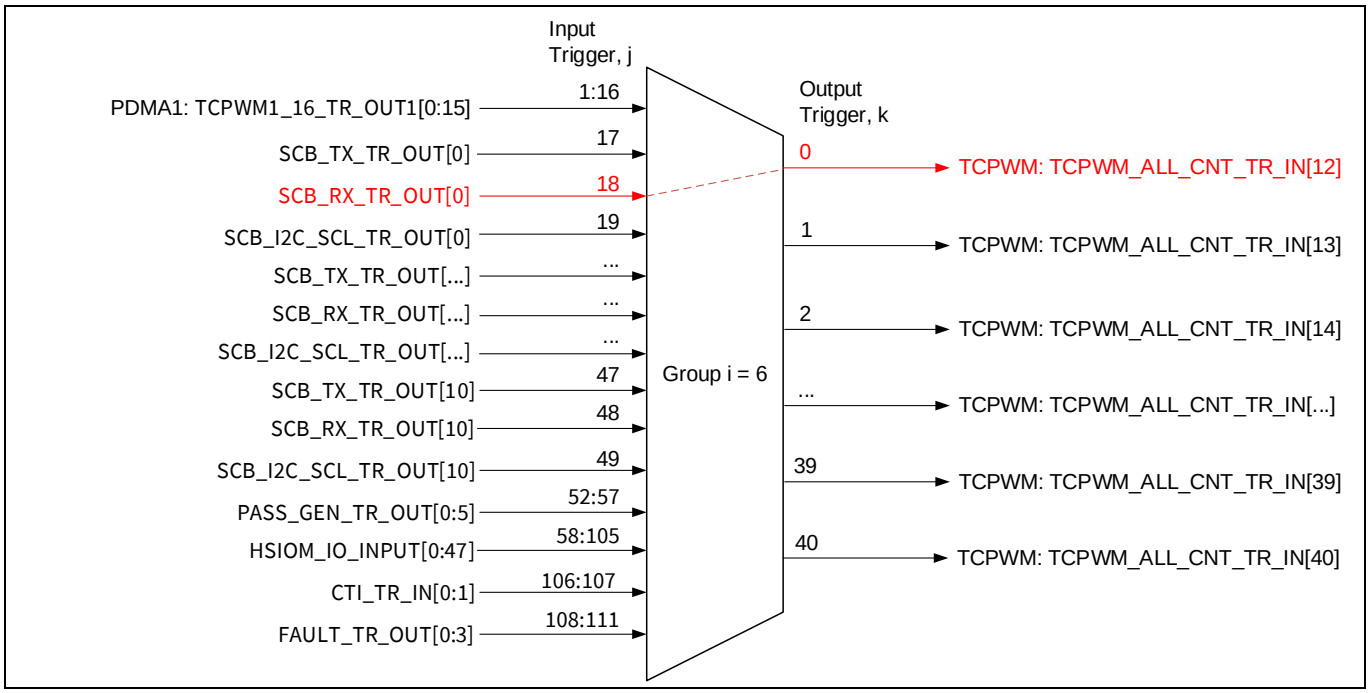

Figure 3 Trigger routing between 16-bit TCPWM and SCB of XMC7200

The following explains how to connect SCB channel 0 Rx and 16-bit TCPWM channel 0:

1. Configuration register selection:
   MUX Group 6 (i=6)
   Output trigger `TCPWM_ALL_CNT_TR_IN[12] (k=0)`
   Therefore, the register is specified as `PERI_TR_GR6_TR_CTL0`.
2. Input trigger selection:
   Input trigger `SCB_RX_TR_OUT[0] (j=18)`
   The configuration will be `PERI_TR_GR6_TR_CTL0.TR_SEL = 18`.

Trigger multiplexer overview

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. The 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 reference manual [2].

For example, in XMC7200, the input trigger Event Generator 0 is routed to a generic trigger 0 of SAR. This trigger routing is included in MUX Group 7. This example is based on Table 4 and Table 5 extracted from the XMC7200 datasheet that provides trigger group, input trigger, and output trigger.

Table 4 Trigger outputs of MUX Group 7 of XMC7200

Output (“k”)	Trigger label	Description
MUX Group 7: PASS trigger multiplexer		
0:11	PASS_GEN_TR_IN[0:11]	Triggers to PASS SAR ADCs

Table 5 Trigger inputs of MUX Group 7 of XMC7200

Input (“j”)	Trigger label	Description
MUX Group 7: PASS (PASS SAR trigger multiplexer)		
1:31	PDMA0_TR_OUT[0:31]	General purpose P-DMA (DW)0 triggers
32:44	TCPWM1_16M_TR_OUT0[0:11]	16-bit Motor enhanced TCPWM1 counters
45:57	TCPWM1_32_TR_OUT0[0:12]	32-bit TCPWM1 counters
58:65	HSIOM_IO_INPUT[0:7]	I/O Inputs
66:68	EVTGEN_TR_OUT[12:14]	Event generator triggers

For the Trigger Group Inputs and Trigger Group Outputs tables of each series, see the device datasheets [1].

Trigger multiplexer overview

Figure 4 shows the input trigger and output trigger structure.

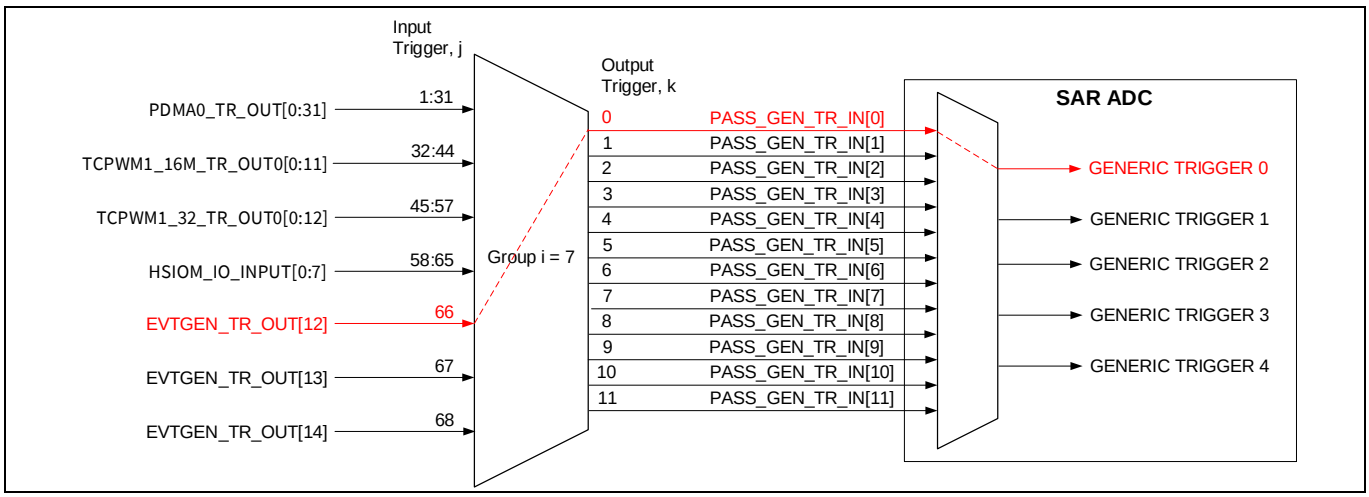

Figure 4 Trigger routing between TCPWM 16-bit Motor and Generic Trigger 2 in ADC of XMC7200

The following explains how the trigger source Event Generator 0 is routed to a generic trigger 12 of SAR0:

1. Configuration register selection:
   MUX Group 7 (i=7)
   Output trigger `PASS_GEN_TR_IN[0] (k=0)`
   Therefore, the register is specified as `PERI_TR_GR7_TR_CTL0`.
2. Input trigger selection:
   Input trigger `EVTGEN_TR_OUT[12] (j=66)`
   The configuration will be `PERI_TR_GR7_TR_CTL0.TR_SEL = 66`.
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 [2].

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. ‘0’: Not inverted ‘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. ‘0’: Level sensitive ‘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.

Trigger multiplexer overview

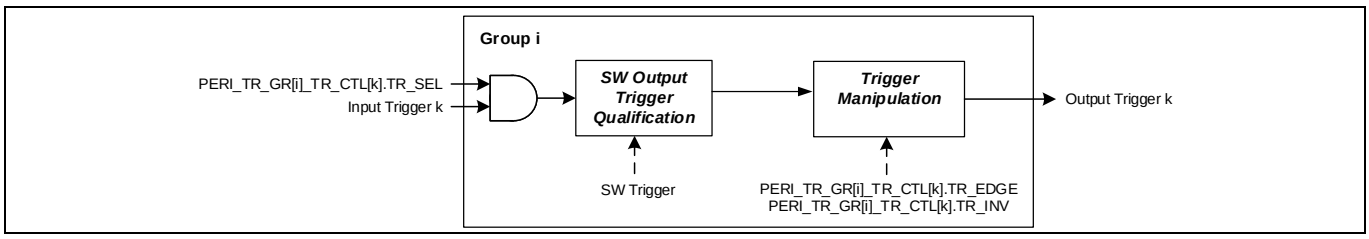

Figure 5 One-to-one trigger

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 enable or disable the 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 a one-to-one trigger.

Example:

This is an example to trigger data transmission on P-DMA (DW)1 channel 23 by SCB UART channel 3. This trigger routing is included in MUX Group 2. The example is based on Table 7 extracted from the XMC7200 datasheet that provides trigger group and input trigger.

Table 7 Trigger one-to-one MUX Group 2 of XMC7200

Input trigger (“k”)	Trigger inputs	Trigger outputs
MUX Group 2: SCB to P-DMA (DW)1 Triggers		
0	SCB0_TX_TR_OUT	PDMA1_TR_IN[16]
1	SCB0_RX_TR_OUT	PDMA1_TR_IN[17]
2	SCB1_TX_TR_OUT	PDMA1_TR_IN[18]
3	SCB1_RX_TR_OUT	PDMA1_TR_IN[19]
4	SCB2_TX_TR_OUT	PDMA1_TR_IN[20]
5	SCB2_RX_TR_OUT	PDMA1_TR_IN[21]
6	SCB3_TX_TR_OUT	PDMA1_TR_IN[22]
7	SCB3_RX_TR_OUT	PDMA1_TR_IN[23]
8	SCB4_TX_TR_OUT	PDMA1_TR_IN[24]
9	SCB4_RX_TR_OUT	PDMA1_TR_IN[25]
10	SCB5_TX_TR_OUT	PDMA1_TR_IN[26]
11	SCB5_RX_TR_OUT	PDMA1_TR_IN[27]
12	SCB6_TX_TR_OUT	PDMA1_TR_IN[28]
13	SCB6_RX_TR_OUT	PDMA1_TR_IN[29]
14	SCB7_TX_TR_OUT	PDMA1_TR_IN[30]
15	SCB7_RX_TR_OUT	PDMA1_TR_IN[31]
16	SCB8_TX_TR_OUT	PDMA1_TR_IN[32]
17	SCB8_RX_TR_OUT	PDMA1_TR_IN[33]
18	SCB9_TX_TR_OUT	PDMA1_TR_IN[34]

Trigger multiplexer overview

Input trigger (“k”)	Trigger inputs	Trigger outputs
MUX Group 2: SCB to P-DMA (DW)1 Triggers		
19	SCB9_RX_TR_OUT	PDMA1_TR_IN[35]
20	SCB10_TX_TR_OUT	PDMA1_TR_IN[36]
21	SCB10_RX_TR_OUT	PDMA1_TR_IN[37]

For the trigger one-to-one table of each series, see the device datasheets [1].

The following describes how to connect SCB channel 3 to P-DMA (DW)1 channel 23.

Note: In a one-to-one group trigger, each 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 2 (i=2)
   Input trigger SCB3_RX_TR_OUT, which directly connects to output trigger PDMA1_TR_IN[23] (k=7)
   Therefore, the register is specified as PERI_TR_1TO1_GR2_TR_CTL7.
2. Input trigger configuration:
   Typically, the input trigger of a one-to-one trigger is enabled.
   The configuration will be PERI_TR_1TO1_GR2_TR_CTL7.TR_SEL = 1.

Table 8 shows the bit registers for a one-to-one trigger that explains the inverted output trigger and the level or edge sensitive trigger. For more details, see the Registers TRM [2].

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. ‘0’: Not inverted ‘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. ‘0’: Level sensitive ‘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 the SW trigger. For the one-to-one trigger, an output trigger can be selected for the SW trigger. Figure 6 shows the group trigger with the SW trigger block diagram. Figure 7 shows the one-to-one trigger with the SW trigger block diagram.

Trigger multiplexer overview

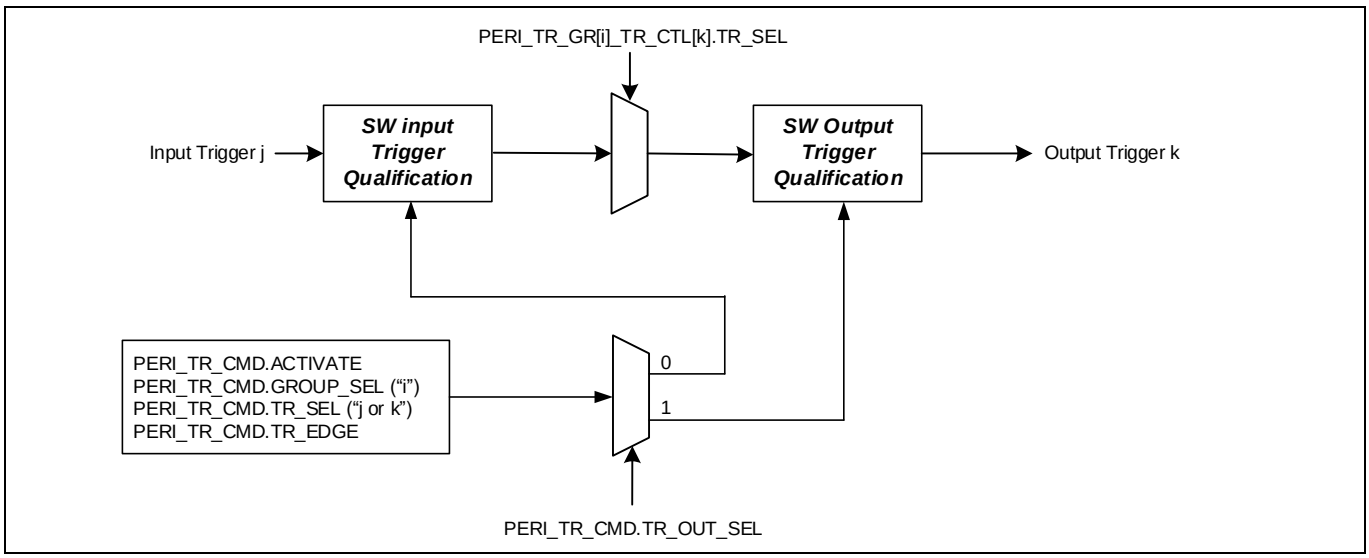

Figure 6 SW trigger in group trigger

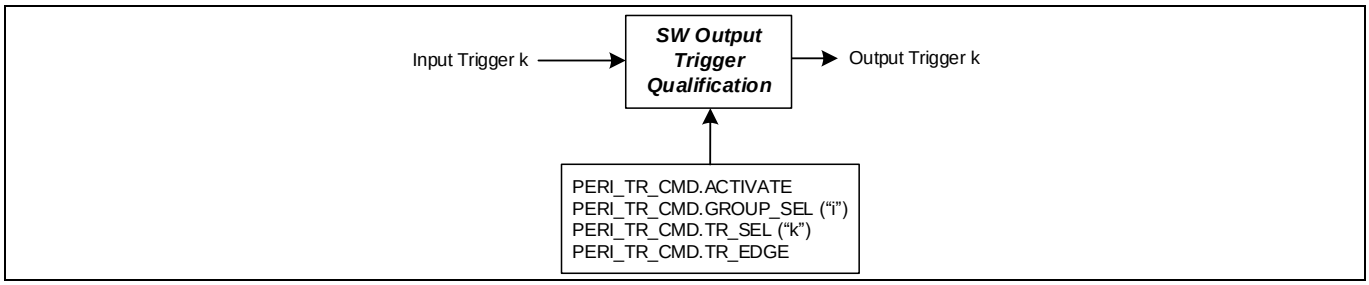

Figure 7 SW trigger in one-to-one trigger

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 a 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’, the input trigger is selected. If set to ‘1’, the output trigger is selected.

Note: This configuration is not used for the 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 a software trigger.

Example:

This is an example to trigger data transfer on P-DMA (DW)0 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.TR_SEL (=k)`.

Trigger multiplexer overview

Table 9 Trigger outputs of MUX Group 0 of XMC7200

Output (“k”)	Trigger label	Description
MUX Group 0: PDMA0_TR (P-DMA (DW)0 trigger multiplexer)		
0:15	PDMA0_TR_IN[0:15]	Triggers to P-DMA (DW)0[0:15]

For the Trigger Group Outputs table of each series, see the device datasheets [1].

Figure 8 shows the trigger of data transfer on P-DMA (DW)0 by SW.

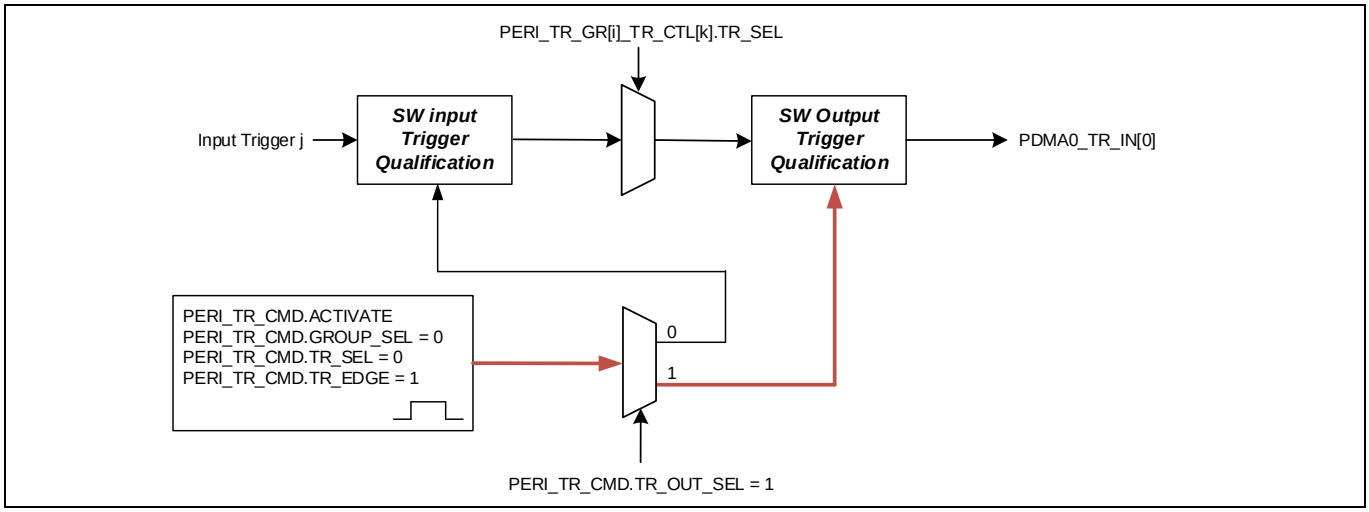

Figure 8 P-DMA (DW)0 data transfer start by SW

The following describes how to initiate data transfer on P-DMA (DW)0:

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 [1].

3 Application

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

- Triggering P-DMA (DW) 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 (DW) transfer by TCPWM timer

This section explains how an input trigger in a group trigger initiates data transfer of a P-DMA (DW)0 channel.

3.1.1 Use case description of triggering P-DMA (DW) transfer by TCPWM timer

Figure 9 shows an example application for group trigger in XMC7200. Data transfer of P-DMA (DW)0 channel 1 is triggered by TCPWM 16-bit channel 0 on group trigger Multiplexer Group 1. TCPWM counter overflow event triggers P-DMA (DW)0 channel 1 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 reference manual [2].

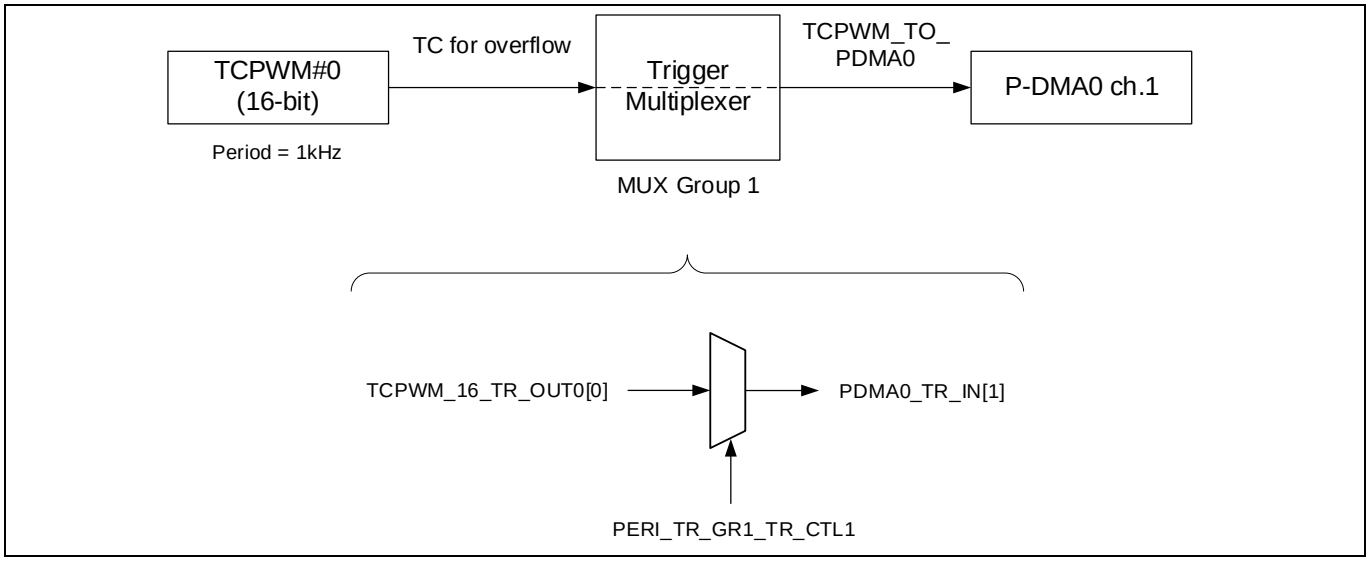

Figure 9 Example of group trigger in XMC7200

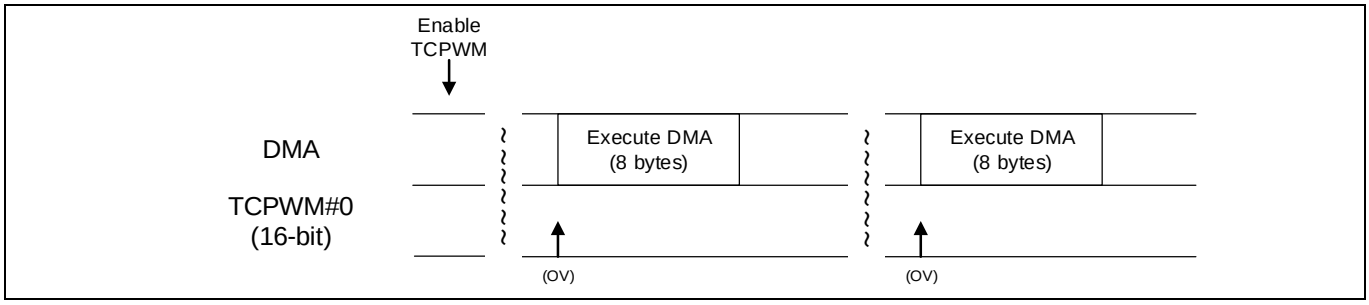

Figure 10 Example operation: Triggering P-DMA (DW) transfer by TCPWM counter overflow

Application

Triggers must be set to implement this application. Figure 11 describes the procedure for setting the trigger. Figure 11 also provides the setting for the TCPWM counter and P-DMA (DW)0 channel.

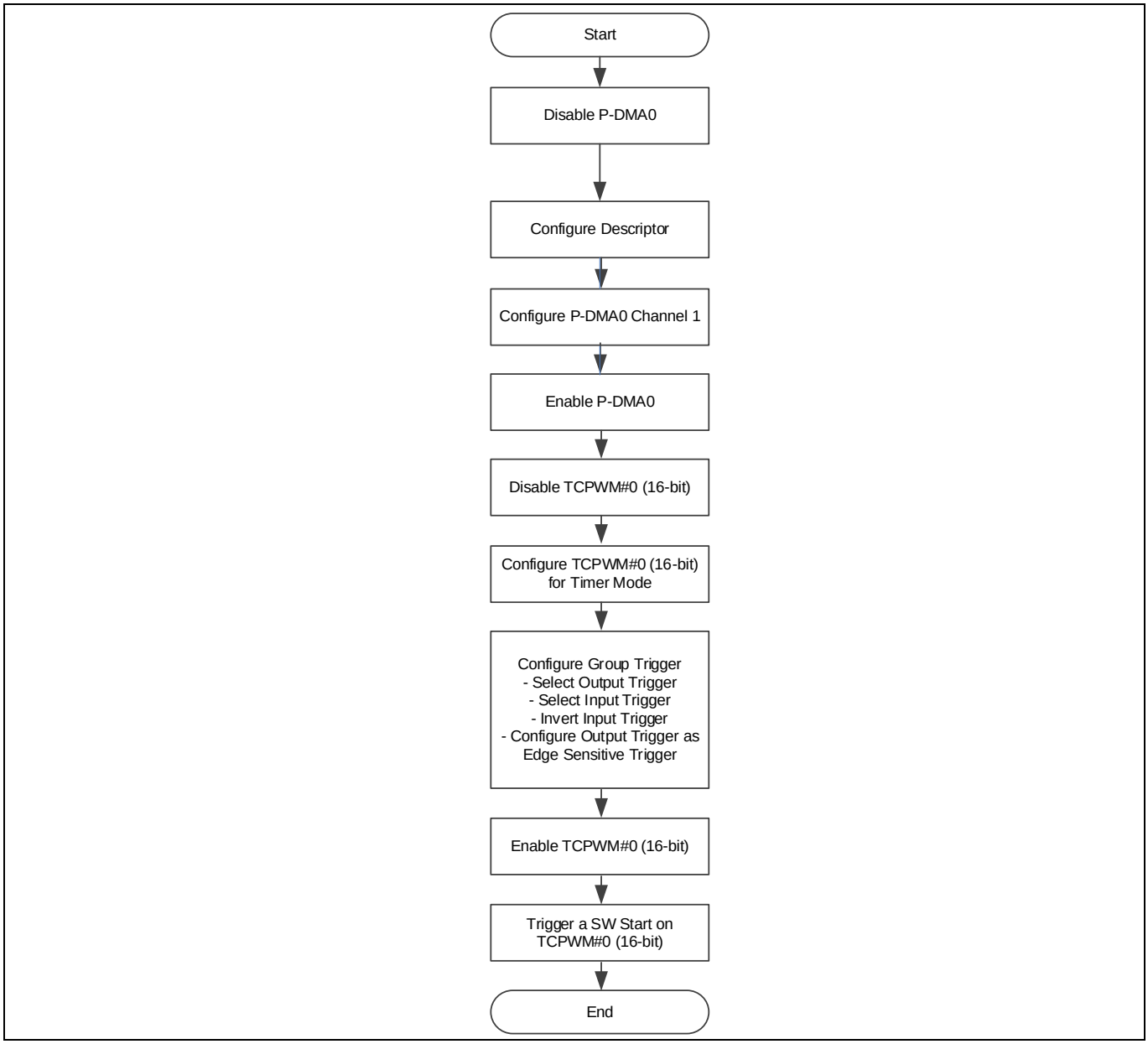

Figure 11 Example of setting for group trigger in XMC7200

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

Trigger multiplexer setting

Table 10 and Table 11 lists the values to be set. These tables are extracted from the XMC7200 datasheet that provides trigger group, input trigger, and output trigger.

Application

Table 10 Trigger outputs of MUX Group 1 of XMC7200

Output (“k”)	Trigger label	Description
MUX Group 1: TCPWM to P-DMA (DW)0 trigger multiplexer		
0:15	PDMA0_TR_IN[16:31]	Triggers to P-DMA0[16:31]

Table 11 Trigger inputs of MUX Group 1 of XMC7200

Input (“j”)	Trigger label	Description
MUX Group 1: TCPWM to P-DMA (DW)0 trigger multiplexer		
1:3	TCPWM0_16_TR_OUT0[0:2]	16-bit TCPWM0 counters
4:6	TCPWM0_16M_TR_OUT0[0:2]	16-bit Motor enhanced TCPWM0 counters
7:9	TCPWM0_32_TR_OUT0[0:2]	32-bit TCPWM0 counters
10:39	TCPWM1_16_TR_OUT0[0:29]	16-bit TCPWM1 counters
40:51	TCPWM1_16M_TR_OUT0[0:11]	16-bit Motor enhanced TCPWM1 counters
52:64	TCPWM1_32_TR_OUT0[0:12]	32-bit TCPWM1 counters
65:70	PASS_GEN_TR_OUT[0:5]	PASS SAR events
71:72	CTI_TR_OUT[0:1]	Trace events
73:76	EVTGEN_TR_OUT[0:3]	Event generator triggers

Figure 12 shows the input trigger and output trigger structure.

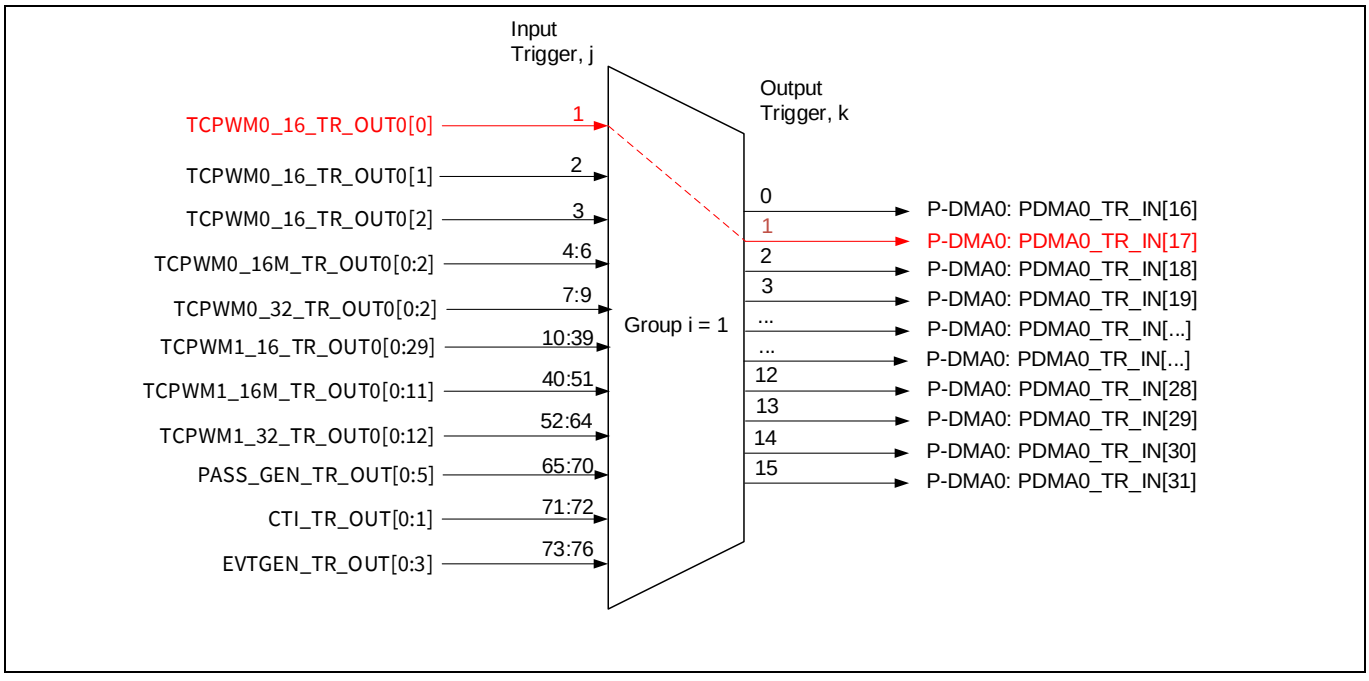

Figure 12 Trigger routing between 16-bit TCPWM and P-DMA (DW)0 of XMC7200

The following explains how to connect P-DMA (DW)0 channel 1 and 16-bit TCPWM channel 0:

1. Configuration register selection:
   MUX Group 1 (i=1)
   Output trigger PDMA0_TR_IN[1] (k=1)
   Therefore, the register is specified as PERI_TR_GR1_TR_CTL1.

2. Input trigger selection:
   Input trigger `TCPWM_16_TR_OUT0[0] (j=1)`

The configuration will be `PERI_TR_GR1_TR_CTL1.TR_SEL = 1`. It is necessary to see the Trigger Multiplexer diagram of each series to make sure the correct multiplexer group and accurate input and output triggers are selected. To find the trigger setting of the group trigger of another MUX group and another series, see the device datasheets [1].

3.1.2 Configuration

Figure 13 shows the parameters of the configuration in the device configurator tool for TCPWM PWM mode.

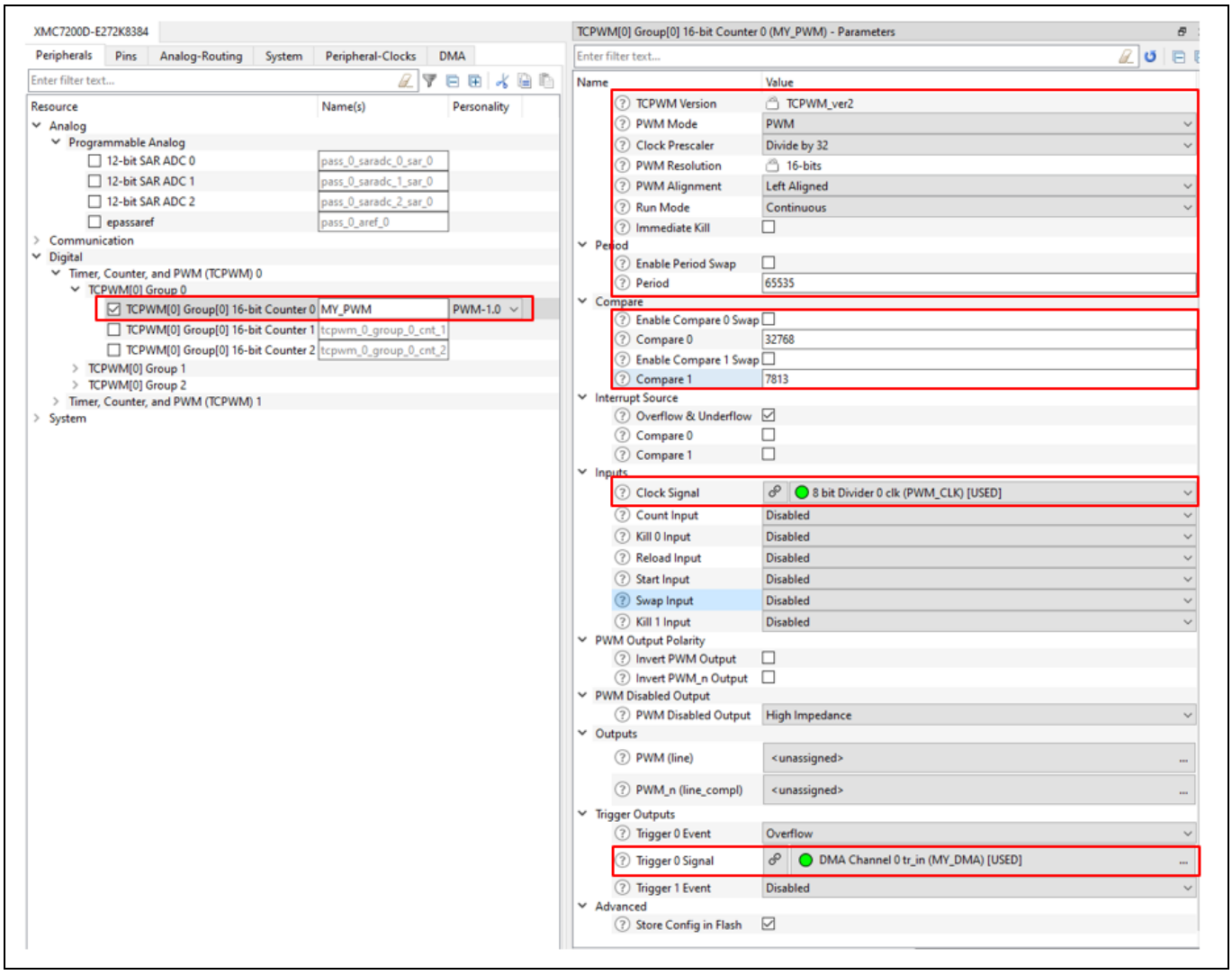

Figure 13 TCPWM counter configuration in the device configurator tool

Application

Figure 14 shows the parameters of the TCPWM counter clock configuration in the device configurator tool.

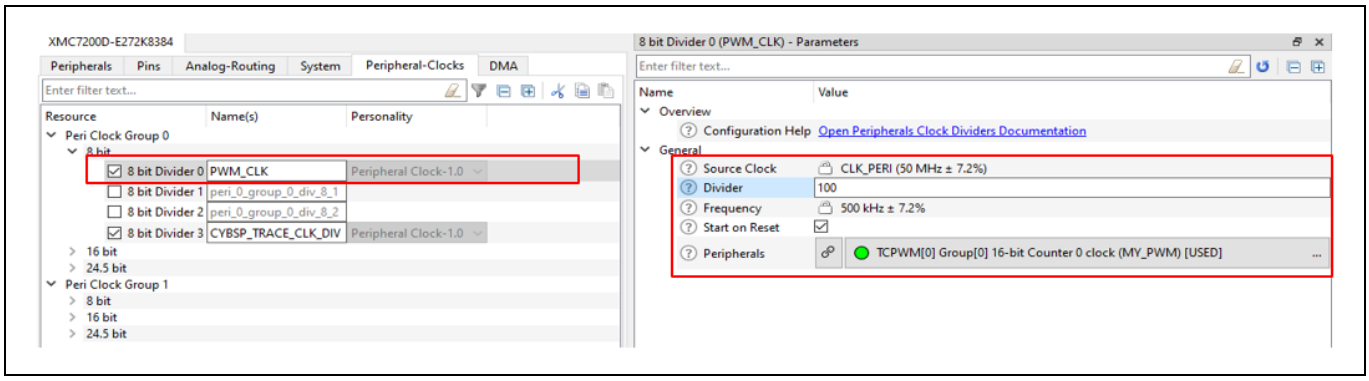

Figure 14 TCPWM counter clock configuration in the device configurator tool

Figure 15 shows the DMA parameters configuration in device configurator tool for TCPWM counter.

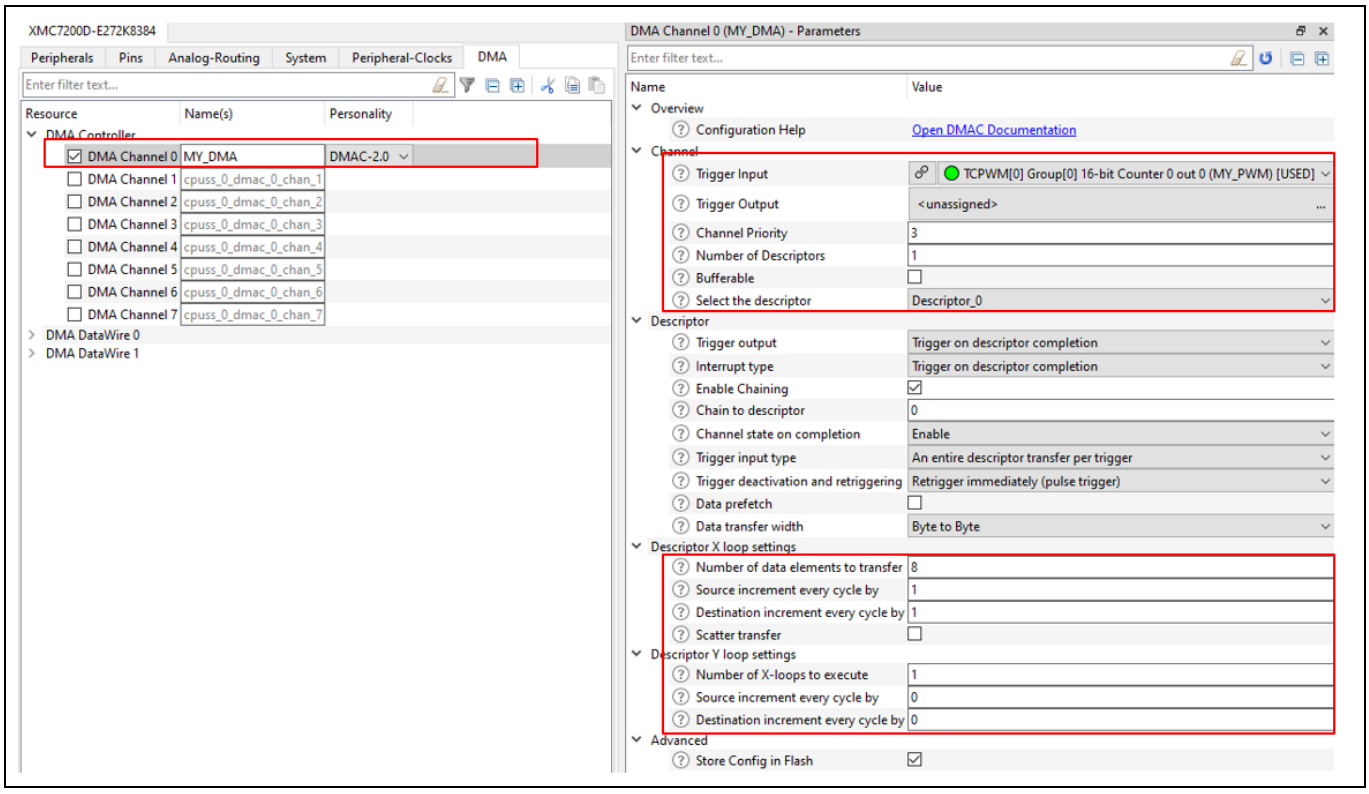

Figure 15 DMA configuration in device configurator tool

3.1.3 Example program for triggering P-DMA (DW) transfer by TCPWM timer

Code Listing 1 shows the example program for Figure 11. The code snippets in this application note are part of the Peripheral Driver Library (PDL). Contact Technical support for the PDL.

Code Listing 1 Example for triggering P-DMA (DW) transfer by TCPWM timer

```
/*******************************************************************************
* Macros
*******************************************************************************/
#define PWM_INTERRUPT_PRIORITY (7u)
#define DMA_INTERRUPT_PRIORITY (6u)
#define BUFFER_SIZE            (8u)

/*******************************************************************************
* Global Variables
*******************************************************************************/
/* PWM interrupt*/
cy_stc_sysint_t intrCfg =
{
    .intrSrc = ((NvicMux3_IRQn << 16) | MY_PWM_IRQ),          /* Interrupt source is tcpwm_0_interrupts_0_IRQn */
    .intrPriority = PWM_INTERRUPT_PRIORITY                    /* Interrupt priority is 7 */
};

/* DMA interrupt*/
cy_stc_sysint_t intrCfg1 =
{
    .intrSrc = ((NvicMux2_IRQn << 16) | MY_DMA_IRQ),          /* Interrupt source is cpuss_interrupts_dmac_0_IRQn */
    .intrPriority = DMA_INTERRUPT_PRIORITY                    /* Interrupt priority is 6 */
};

/*DMA transfer source buffer and destination buffer*/
uint8_t src[BUFFER_SIZE];
uint8_t dst[BUFFER_SIZE];

uint8_t dmaFlag;

/*******************************************************************************
* Function Prototypes
*******************************************************************************/
static void pwm_interrupt_handler();
static void dma_interrupt_handler();
static void arrayInit(void);

/*******************************************************************************
* Function Definitions
*******************************************************************************/

/*******************************************************************************
* Function Name: main
********************************************************************************
* Summary:
* This is the main function for PWM trigger DMA data transfer function.
*    1. initialize the BSP.
*    2. initialize the retarget_io and user led1 pins.
*    3. initialize the PWM and enable the PWM terminal count interrupt.
*    4. initialize the DMA and enable the DMA descriptor interrupt.
*    5. Start the PWM and DMA functions. PWM terminal output event will trigger the DMA start to transfer data.
*    6. PWM terminal interrupt will toggle user led1. UART terminal will output DMA transfer data(0~7).
*
* Parameters:
*  void
*
* Return:
*  int
*
*******************************************************************************/
int main(void)
{
    cy_rslt_t result;
    uint8_t cnt;

    /* Initialize the device and board peripherals */
    result = cybsp_init();

    /* Board init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Enable global interrupts */
```

Application

Code Listing 1 Example for triggering P-DMA (DW) transfer by TCPWM timer

```
    __enable_irq();

    arrayInit();

    /* Initialize the retarget-io to use the debug UART port */
       result = cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX, CY_RETARGET_IO_BAUDRATE);
       if (CY_RSLT_SUCCESS != result)
       {
           /* Halt the CPU while debugging */
           CY_ASSERT(false);
       }

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

    /* Initialize PWM using the config structure generated using device configurator*/
    if (CY_TCPWM_SUCCESS != Cy_TCPWM_PWM_Init(MY_PWM_HW, MY_PWM_NUM, &MY_PWM_config))
       {
           CY_ASSERT(0);
       }

    /* Enable the initialized PWM */
    Cy_TCPWM_PWM_Enable(MY_PWM_HW, MY_PWM_NUM);

    /* Configure and enable PWM terminal count interrupt */
    Cy_SysInt_Init(&intrCfg, pwm_interrupt_handler);
    NVIC_ClearPendingIRQ((IRQn_Type)intrCfg.intrSrc);
    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);

    /* Start the PWM */
    Cy_TCPWM_TriggerStart_Single(MY_PWM_HW, MY_PWM_NUM);

    /*Initialize DMA descriptor */
    if (CY_DMAC_SUCCESS != Cy_DMAC_Descriptor_Init(&MY_DMA_Descriptor_0, &MY_DMA_Descriptor_0_config))
       {
            CY_ASSERT(0);
       }

    /*Set the DMA source address and destination address*/
    Cy_DMAC_Descriptor_SetSrcAddress(&MY_DMA_Descriptor_0,src);
    Cy_DMAC_Descriptor_SetDstAddress(&MY_DMA_Descriptor_0, dst);

    /* Configure and enable DMA interrupt */
    Cy_SysInt_Init(&intrCfg1, dma_interrupt_handler);
    NVIC_ClearPendingIRQ((IRQn_Type)intrCfg1.intrSrc);
    NVIC_EnableIRQ((IRQn_Type) NvicMux2_IRQn);

    /*Initialize DMA channel*/
    if (CY_DMAC_SUCCESS != Cy_DMAC_Channel_Init(MY_DMA_HW, MY_DMA_CHANNEL, &MY_DMA_channelConfig))
        {
            CY_ASSERT(0);
        }
    /*Set the DMA descriptor*/
    Cy_DMAC_Channel_SetDescriptor(MY_DMA_HW, MY_DMA_CHANNEL, &MY_DMA_Descriptor_0);
    Cy_DMAC_Channel_SetPriority(MY_DMA_HW, MY_DMA_CHANNEL, 3UL);

    /*Set the DMA channel interrupt*/
    Cy_DMAC_Channel_SetInterruptMask (MY_DMA_HW, MY_DMA_CHANNEL, CY_DMAC_INTR_MASK);
    Cy_DMAC_Channel_SetInterrupt(MY_DMA_HW, MY_DMA_CHANNEL, CY_DMAC_INTR_COMPLETION);

    /*Enable DMA channel*/
    Cy_DMAC_Channel_Enable(MY_DMA_HW, MY_DMA_CHANNEL);

    /* Enable DMA hardware block */
    Cy_DMAC_Enable(MY_DMA_HW);

    for (;;)
    {
    /*DMA finish one descriptor transfer*/
       if(dmaFlag == 1u)
       {
           for(cnt = 0; cnt < BUFFER_SIZE; cnt++)
           {
            printf("%d ", dst[cnt]);
           }
           memset(&dst,0,8);
           dmaFlag = 0u;
           printf("\r\n");
       }
    }
}

/*******************************************************************************
```

Application

Code Listing 1 Example for triggering P-DMA (DW) transfer by TCPWM timer

```
* Function Name: pwm_Interrupt_handler_pdl
********************************************************************************
*
*  Summary:
*  pwm TC interrupt handler for the PDL example.
*
*  Parameters:
*  None
*
*  Return:
*  None
*
*******************************************************************************/
static void pwm_interrupt_handler()
{
    /*toggle user LED1*/
    Cy_GPIO_Inv(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_PIN);

    /*clear the pwm terminal count interrupt*/
    Cy_TCPWM_ClearInterrupt(MY_PWM_HW, MY_PWM_NUM, CY_TCPWM_INT_ON_TC);
}

/*******************************************************************************
* Function Name: dma_interrupt_handler
********************************************************************************
*
*  Summary:
*  DMA one descriptor transfer finish interrupt handler.
*
*  Parameters:
*  None
*
*  Return:
*  None
*
*******************************************************************************/
void dma_interrupt_handler()
{
    dmaFlag = 1u;
    /*clear the DMA channel interrupt*/
    Cy_DMAC_Channel_ClearInterrupt(MY_DMA_HW, MY_DMA_CHANNEL, CY_DMAC_INTR_COMPLETION);
}

/*******************************************************************************
* Function Name: arrayInit
********************************************************************************
* Summary:
* This function creates an source array for DMA transfer function.
*
* Parameters:
*  void
*
* Return:
*  void
*
*******************************************************************************/
void arrayInit(void)
{
    uint8_t i;

    for(i=0; i < BUFFER_SIZE; i++)
    {
        src[i] = i;
    }
}
```

Note: The highlighted sections of the code snippet are not explained in this application note. For details, see the architecture reference manual [2].

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 the 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 16 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 7. 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 17 demonstrates the operation. For more details on TCPWM and SAR ADC, see the “TCPWM” and “SAR ADC” chapters of the architecture reference manual [2].

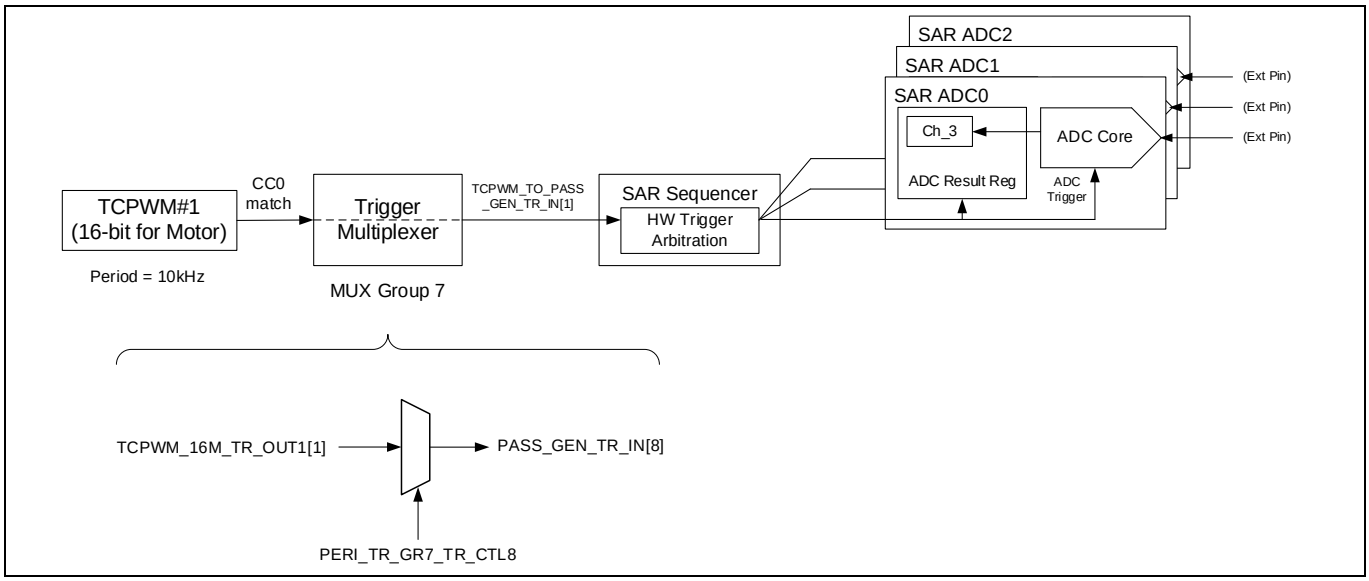

Figure 16 Example of group trigger and ADC generic trigger input in XMC7200

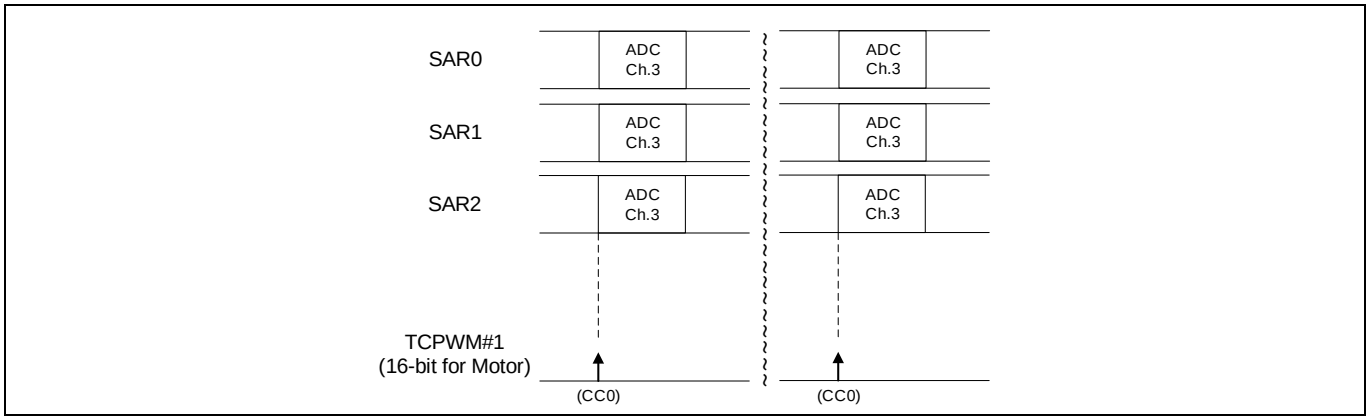

Figure 17 Example operation: Simultaneous ADC conversion on three SARs by single TCPWM timer

Application

Triggers must be set to implement this application. Figure 18 demonstrates the procedure for setting the trigger.

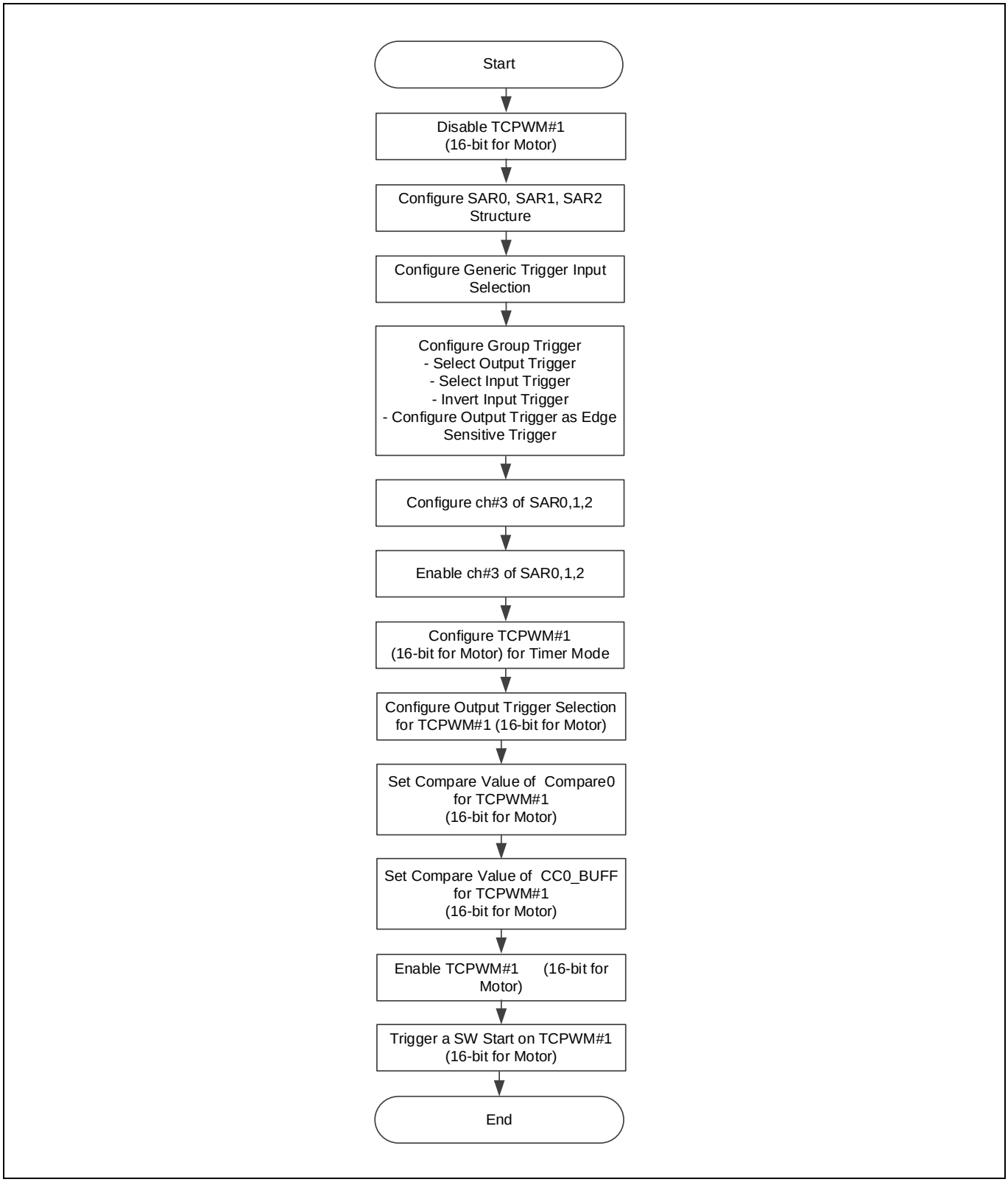

Figure 18 Setting for ADC generic trigger input and group trigger in XMC7200

Application

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 lists the values to be set. These tables are extracted from the XMC7200 datasheet that provides trigger group, input trigger, and output trigger.

Table 12 Trigger outputs of MUX Group 7 of XMC7200

Output (“k”)	Trigger label	Description
MUX Group 7: PASS trigger multiplexer		
0:11	PASS_GEN_TR_IN[0:11]	Triggers to PASS SAR ADCs

Table 13 Trigger inputs of MUX Group 7 of XMC7200

Input (“j”)	Trigger label	Description
MUX Group 7: PASS trigger multiplexer		
1:31	PDMA0_TR_OUT[0:31]	General purpose P-DMA0 triggers
32:44	TCPWM1_16M_TR_OUT0[0:11]	16-bit Motor enhanced TCPWM1 counters
45:57	TCPWM1_32_TR_OUT0[0:12]	32-bit TCPWM1 counters
58:65	HSIOM_IO_INPUT[0:7]	I/O Inputs
66:68	EVTGEN_TR_OUT[12:14]	Event generator triggers

For Trigger Group Inputs and Trigger Group Outputs tables of each series, see the device datasheets [1].

Figure 19 shows the input trigger and output trigger structure.

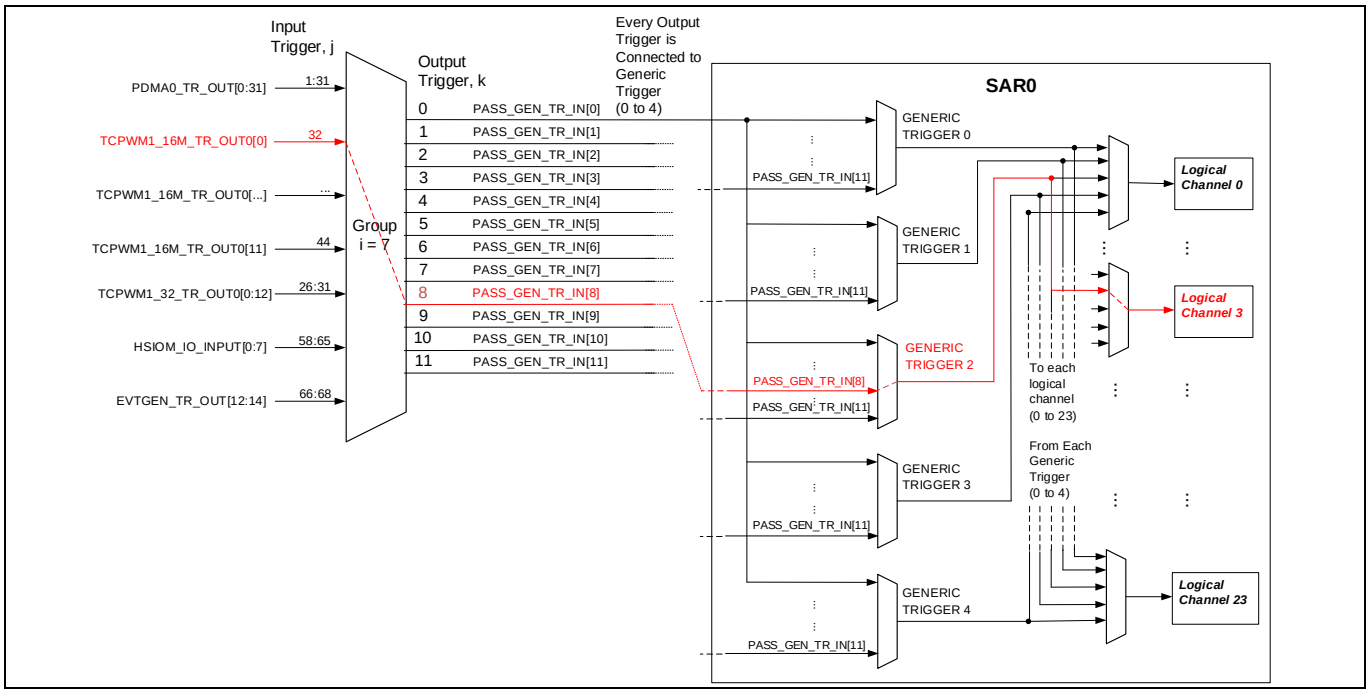

Figure 19 Trigger routing between TCPWM 16-bit motor and generic trigger 2 in SAR0 unit of XMC7200

Application

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 7 (i=7)
   Output trigger, generic trigger 8 or `PASS_GEN_TR_IN[8] (k=8)`
   Therefore, the register is specified as `PERI_TR_GR7_TR_CTL8`.
2. Input trigger selection:
   Input trigger `TCPWM_16M_TR_OUT1[0] (j=32)`
   The configuration will be `PERI_TR_GR6_TR_CTL2.TR_SEL = 32`.
3. Generic trigger 2 is forwarded to SAR generic trigger input 2, IN2 of the corresponding SAR unit:

- For SAR0, the configuration will be
  - `SAR_TR_IN_SEL0.IN2_SEL = 8`
- For SAR1, the configuration will be
  - `SAR_TR_IN_SEL1.IN2_SEL = 8`
- For SAR2, the configuration will be
  - `SAR_TR_IN_SEL2.IN2_SEL = 8`

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

- For SAR0, the configuration for SAR trigger select will be
  - `PASS0_SAR0_CH3_TR_CTL.SEL = 4` (This value represents SAR Generic trigger input 2)
- For SAR1, the configuration for SAR trigger selection will be
  - `PASS0_SAR1_CH3_TR_CTL.SEL = 4`
- For SAR2, the configuration for SAR trigger selection will be
  - `PASS0_SAR2_CH3_TR_CTL.SEL = 4`

To find the trigger setting of the group trigger, see the device datasheets [1].

3.2.2 Configuration

Figure 20 shows the parameters of the configuration part in the device configurator tool for SAR.

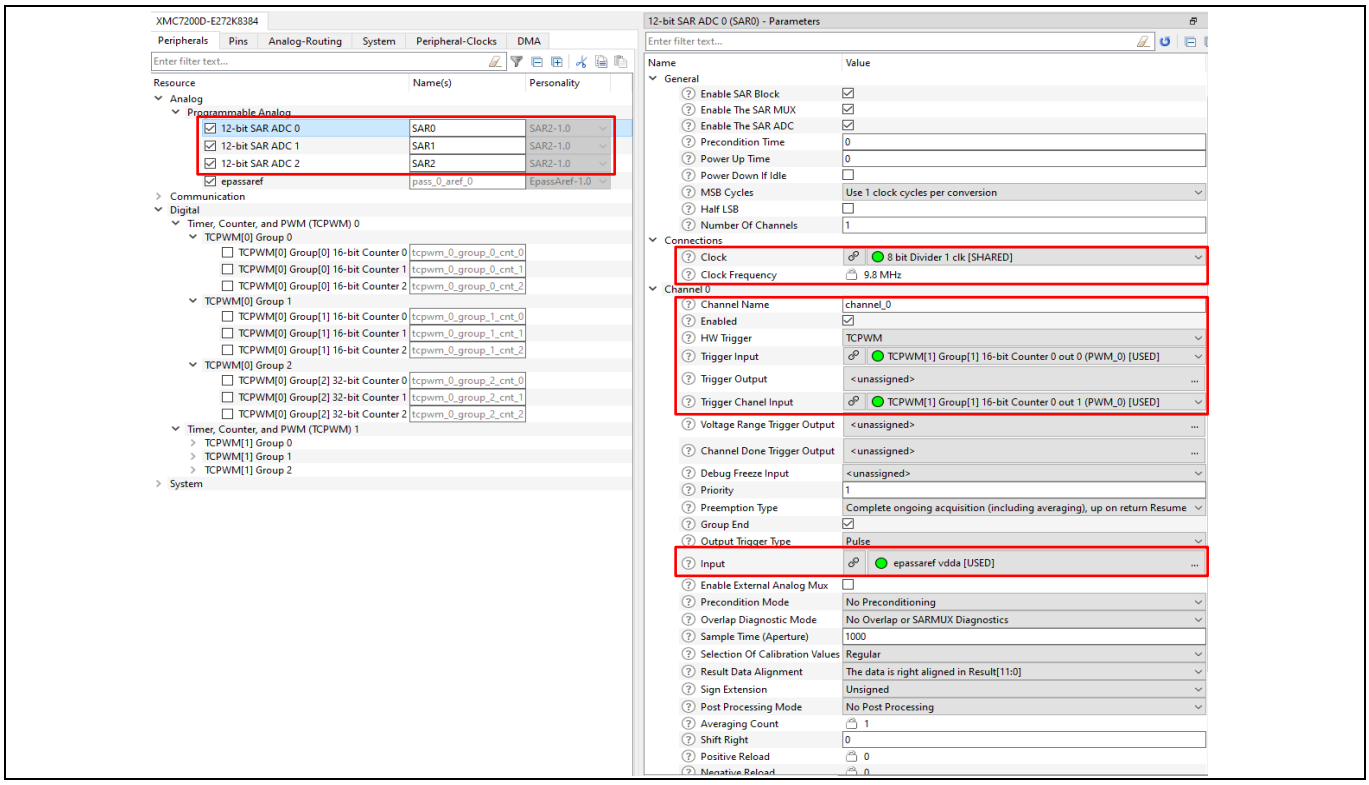

Figure 20 Three SAR unit configuration in the device configurator tool

Figure 21 shows the parameters of TCPWM configuration in the device configurator tool.

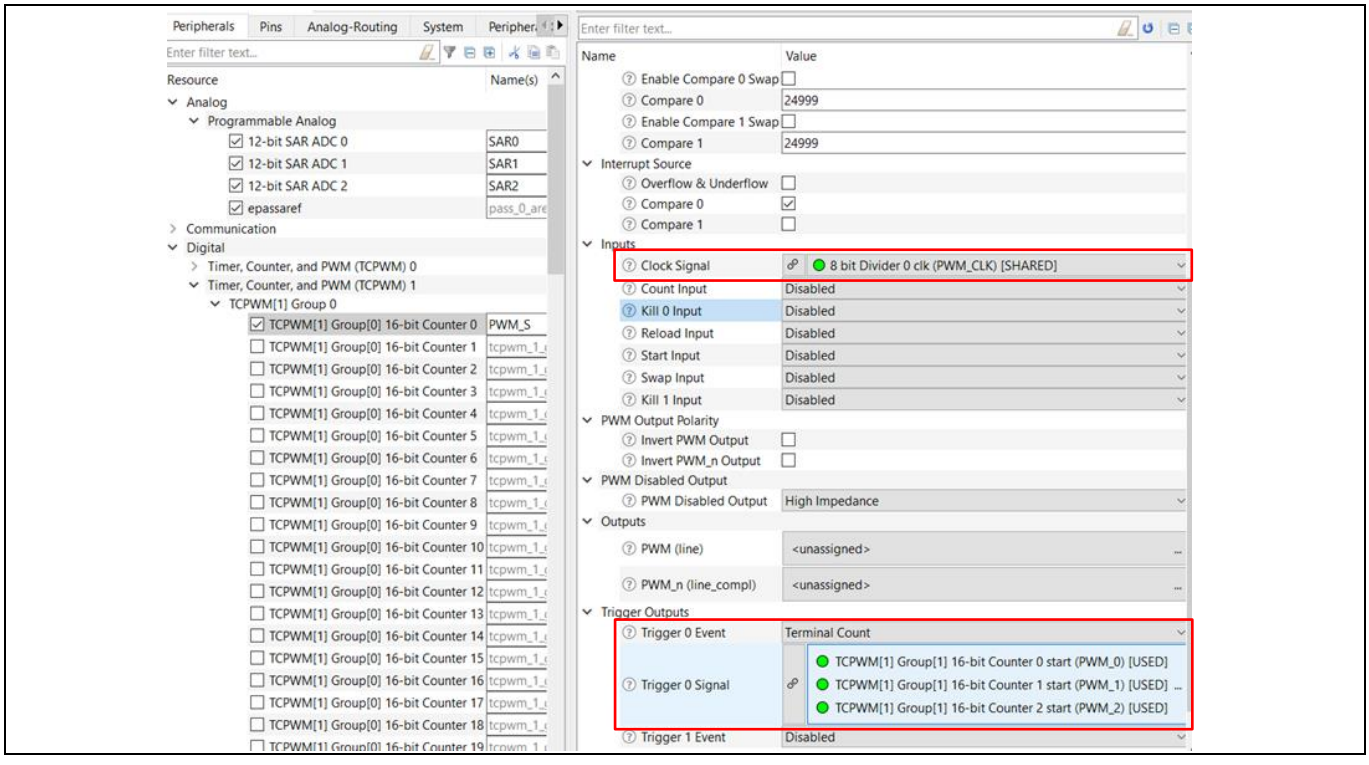

Figure 21 TCPWM configuration in the device configurator tool

Application

Figure 22 shows the parameters of three units TCPWM configuration in the device configurator tool.

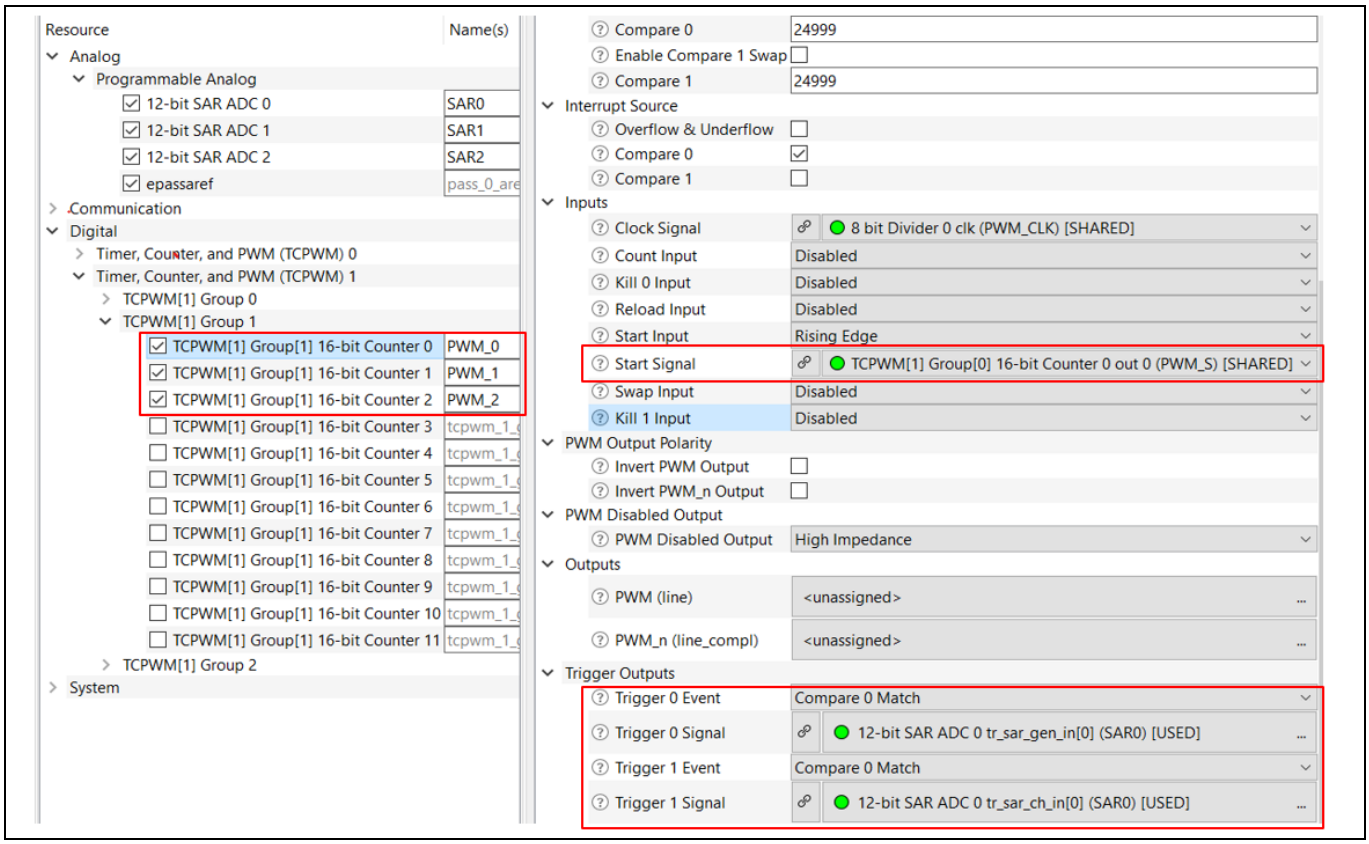

Figure 22 Three units TCPWM configuration in the device configurator tool

3.2.3 Example program for simultaneous ADC conversion on three SARs by single TCPWM timer

Code Listing 2 shows the example program for Figure 18.

Code Listing 2 Example of simultaneous ADC conversion on three SARs by single TCPWM timer

```
int main(void)
{
    cy_rslt_t result;
    uint16_t sar0Result = 0u;
    uint16_t sar1Result = 0u;
        uint16_t sar2Result = 0u;

    /* Initialize the device and board peripherals */
    result = cybsp_init();

    /* Board init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Enable global interrupts */
    __enable_irq();

    /* Initialize the retarget-io to use the debug UART port */
    result = cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX, CY_RETARGET_IO_BAUDRATE);
    if (CY_RSLT_SUCCESS != result)
       {
           /* Halt the CPU while debugging */
           CY_ASSERT(false);
       }

    /*Analog resource initialize*/
    init_analog_resources();
```

Application

Code Listing 2 Example of simultaneous ADC conversion on three SARs by single TCPWM timer

```
    /*pwm resource initialize*/
    init_pwm_resources();

    /* Initialize PWM_S*/
    if (CY_TCPWM_SUCCESS != Cy_TCPWM_PWM_Init(PWM_S_HW, PWM_S_NUM, &PWM_S_config))
       {
           CY_ASSERT(0);
       }

    /* Enable the PWM_S */
    Cy_TCPWM_PWM_Enable(PWM_S_HW, PWM_S_NUM);

    /* Configure and enable PWM_S CC0 match interrupt */
    Cy_SysInt_Init(&PWM_CC0_IRQ_cfg, pwm_interrupt_handler);
    NVIC_ClearPendingIRQ((IRQn_Type)PWM_CC0_IRQ_cfg.intrSrc);
    NVIC_EnableIRQ((IRQn_Type) NvicMux2_IRQn);

    /* Start the PWM_S to trigger three PWMs and start three ADCs(one-to-one triggers) */
    Cy_TCPWM_TriggerStart_Single(PWM_S_HW, PWM_S_NUM);

    for (;;)
    {
    /*get SAR0, SAR1, SAR2 channels results and showed on terminal*/
   if(sar0_isr_set)
      {
       sar0Result = Cy_SAR2_Channel_GetResult(SAR0_HW, 0u, NULL);
       sar0_isr_set = false;
       printf("sar0Result = %d \r\n",sar0Result );
      }

      if(sar1_isr_set)
        {
          sar1Result = Cy_SAR2_Channel_GetResult(SAR1_HW, 0u, NULL);
          sar1_isr_set = false;
          printf("sar1Result = %d \r\n",sar1Result );
        }

      if(sar2_isr_set)
        {
          sar2Result = Cy_SAR2_Channel_GetResult(SAR2_HW, 0u, NULL);
          sar2_isr_set = false;
          printf("sar2Result = %d \r\n",sar2Result );
        }
    }
}

/*******************************************************************************
* Function Name: sar0_interrupt
********************************************************************************
* Summary:
* This function is the handler for SAR0 interrupt
*
* Parameters:
*  None
*
* Return:
*  None
*
*******************************************************************************/
static void sar0_interrupt_handler(void)
{
    /* Check if End-Of-Scan trigger has occurred. If yes, set sar0_isr_set flag to true  */
    if (Cy_SAR2_Channel_GetInterruptStatus(SAR0_HW,0u) == CY_SAR2_INT_GRP_DONE)
    {
        sar0_isr_set = true;
    }

    /* Clear the interrupts */
    Cy_SAR2_Channel_ClearInterrupt(SAR0_HW, 0u ,CY_SAR2_INT_GRP_DONE);
}

/*******************************************************************************
* Function Name: sar1_interrupt
********************************************************************************
* Summary:
* This function is the handler for SAR1 interrupt
*
* Parameters:
*  None
*
* Return:
*  None
```

Application

Code Listing 2 Example of simultaneous ADC conversion on three SARs by single TCPWM timer

```
*
*******************************************************************************/
static void sar1_interrupt_handler(void)
{
    /* Check if End-Of-Scan trigger has occurred. If yes, set sar1_isr_set flag to true  */
    if (Cy_SAR2_Channel_GetInterruptStatus(SAR1_HW, 0u) == CY_SAR2_INT_GRP_DONE)
    {
        sar1_isr_set = true;
    }

    /* Clear the interrupts */
    Cy_SAR2_Channel_ClearInterrupt(SAR1_HW, 0u,CY_SAR2_INT_GRP_DONE);
}

/*******************************************************************************
* Function Name: sar2_interrupt
********************************************************************************
* Summary:
* This function is the handler for SAR1 interrupt
*
* Parameters:
*  None
*
* Return:
*  None
*
*******************************************************************************/
static void sar2_interrupt_handler(void)
{
    /* Check if End-Of-Scan trigger has occurred. If yes, set sar1_isr_set flag to true  */
    if (Cy_SAR2_Channel_GetInterruptStatus(SAR2_HW, 0u) == CY_SAR2_INT_GRP_DONE)
    {
        sar2_isr_set = true;
    }

    /* Clear the interrupts */
    Cy_SAR2_Channel_ClearInterrupt(SAR2_HW, 0u,CY_SAR2_INT_GRP_DONE);
}

/*******************************************************************************
* Function Name: pwm_interrupt
********************************************************************************
* Summary:
* This function is the handler for PWM CC0 interrupt
*
* Parameters:
*  None
*
* Return:
*  None
*
*******************************************************************************/
static void pwm_interrupt_handler()
{
    /*toggle user LED1*/
    Cy_GPIO_Inv(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_PIN);

    /*clear the PWM_S CC0 match interrupt*/
    Cy_TCPWM_ClearInterrupt(PWM_S_HW, PWM_S_NUM, CY_TCPWM_INT_ON_CC0);
}

/*******************************************************************************
* Function Name: init_analog_resources
********************************************************************************
* Summary:
* This function initializes analog components - CTBM, SAR ADC and CTDAC
*
* Parameters:
*  void
*
* Return:
*  void
*
*******************************************************************************/
static void init_analog_resources(void)
{
    /* Variable to capture return value of functions */
    cy_en_sar2_status_t status;

    /* Initialize SAR0 */
    status = Cy_SAR2_Init(SAR0_HW, &SAR0_config);
    if (CY_SAR2_SUCCESS != status)
    {
```

Application

Code Listing 2 Example of simultaneous ADC conversion on three SARs by single TCPWM timer

```
        CY_ASSERT(0);
    }

    /* Initialize SAR1 */
    status = Cy_SAR2_Init(SAR1_HW, &SAR1_config);
    if (CY_SAR2_SUCCESS != status)
    {
        CY_ASSERT(0);
    }

    /* Initialize SAR2 */
    status = Cy_SAR2_Init(SAR2_HW, &SAR2_config);
    if (CY_SAR2_SUCCESS != status)
    {
        CY_ASSERT(0);
    }

    /*Enable SAR0, SAR1 and SAR2*/
    Cy_SAR2_Enable(SAR0_HW);
    Cy_SAR2_Enable(SAR1_HW);
    Cy_SAR2_Enable(SAR2_HW);

    /*Configure and register SAR0, SAR1 and SAR2 channels interrupts*/
    Cy_SysInt_Init(&SAR0_IRQ_cfg, sar0_interrupt_handler);
    NVIC_ClearPendingIRQ((IRQn_Type)SAR0_IRQ_cfg.intrSrc);

    Cy_SysInt_Init(&SAR1_IRQ_cfg, sar1_interrupt_handler);
    NVIC_ClearPendingIRQ((IRQn_Type)SAR1_IRQ_cfg.intrSrc);

    Cy_SysInt_Init(&SAR2_IRQ_cfg, sar2_interrupt_handler);
    NVIC_ClearPendingIRQ((IRQn_Type)SAR2_IRQ_cfg.intrSrc);
    NVIC_EnableIRQ((IRQn_Type) NvicMux4_IRQn);


    /*Initialize SAR0, SAR1 and SAR2 channels*/
    status = Cy_SAR2_Channel_Init(SAR0_HW, 0u, &SAR0_channel_0_config);
    if (CY_SAR2_SUCCESS != status)
    {
        CY_ASSERT(0);
    }

    status = Cy_SAR2_Channel_Init(SAR1_HW, 0u, &SAR1_channel_0_config);
    if (CY_SAR2_SUCCESS != status)
    {
       CY_ASSERT(0);
    }

    status = Cy_SAR2_Channel_Init(SAR2_HW, 0u, &SAR2_channel_0_config);
    if (CY_SAR2_SUCCESS != status)
    {
       CY_ASSERT(0);
    }

    /*Set SAR0, SAR1, SAR2 channels interrupt mask*/
    Cy_SAR2_Channel_SetInterruptMask(SAR0_HW, 0u,CY_SAR2_INT_GRP_DONE);
    Cy_SAR2_Channel_SetInterruptMask(SAR1_HW, 0u,CY_SAR2_INT_GRP_DONE);
    Cy_SAR2_Channel_SetInterruptMask(SAR2_HW, 0u,CY_SAR2_INT_GRP_DONE);

    /*Enable SAR0, SAR1 and SAR2 channels*/
    Cy_SAR2_Channel_Enable(SAR0_HW, 0u);
    Cy_SAR2_Channel_Enable(SAR1_HW, 0u);
    Cy_SAR2_Channel_Enable(SAR2_HW, 0u);
}

void init_pwm_resources(void)
{
    /* Initialize and enable PWM_0*/
    if (CY_TCPWM_SUCCESS != Cy_TCPWM_PWM_Init(PWM_0_HW, PWM_0_NUM, &PWM_0_config))
       {
           CY_ASSERT(0);
       }
    Cy_TCPWM_PWM_Enable(PWM_0_HW, PWM_0_NUM);

    /* Initialize and enable PWM_1*/
     if (CY_TCPWM_SUCCESS != Cy_TCPWM_PWM_Init(PWM_1_HW, PWM_1_NUM, &PWM_1_config))
        {
              CY_ASSERT(0);
        }
     Cy_TCPWM_PWM_Enable(PWM_1_HW, PWM_1_NUM);

     /* Initialize and enable PWM_2*/
      if (CY_TCPWM_SUCCESS != Cy_TCPWM_PWM_Init(PWM_2_HW, PWM_2_NUM, &PWM_2_config))
            {
```

Code Listing 2 Example of simultaneous ADC conversion on three SARs by single TCPWM timer

```
                CY_ASSERT(0);
            }
        Cy_TCPWM_PWM_Enable(PWM_2_HW, PWM_2_NUM);
}
```

Note: The highlighted sections of the code snippet are not explained in this application note. For details, see architecture reference manual [2].

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 23 shows an example application of a one-to-one trigger in XMC7200. ADC conversion of SAR0 is triggered by TCPWM 16-bit counter on one-to-one trigger Multiplexer Group 7. Compare match of TCPWM counter channel 0 and channel 1 trigger the conversion on channels 4 and 5 of SAR0, respectively. Figure 24 demonstrates the operation. For more details on TCPWM and SAR ADC, see the TCPWM and SAR ADC chapters of the architecture reference manual [2].

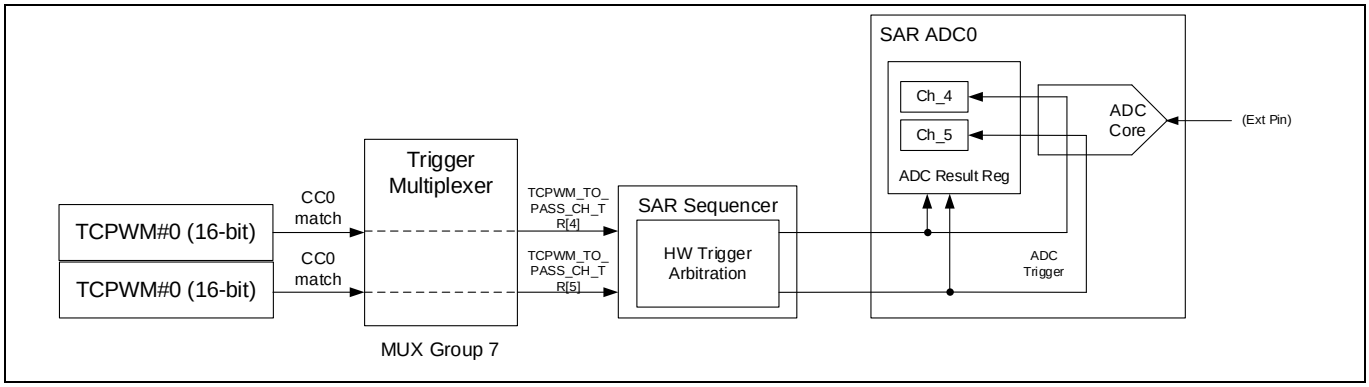

Figure 23 Example of one-to-one trigger in XMC7200

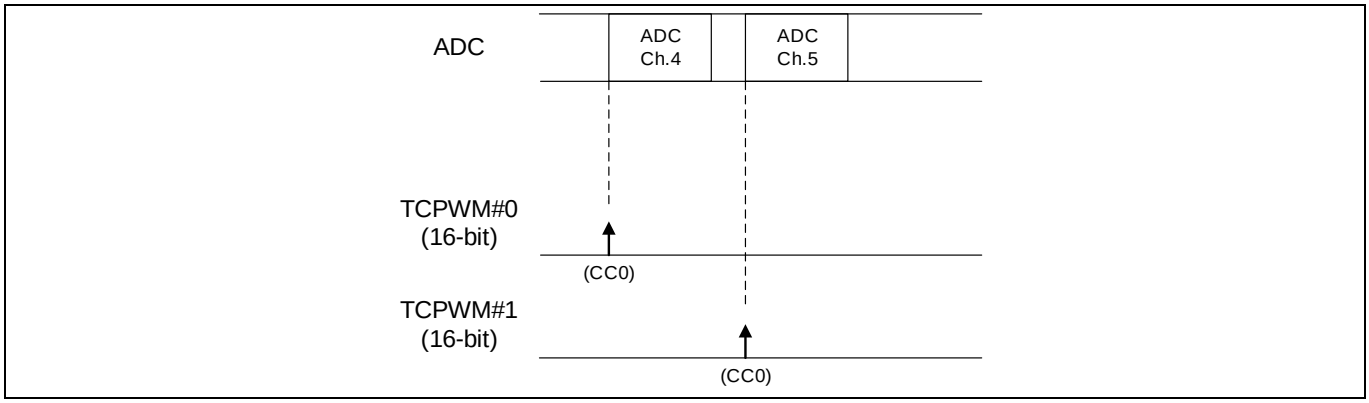

Figure 24 Example operation: Triggering ADC conversion by TCPWM timer

Triggers must be set to implement this application. Figure 25 describes the procedure for setting the trigger. It also provides the setting for TCPWM counter and SAR0 structure.

Application

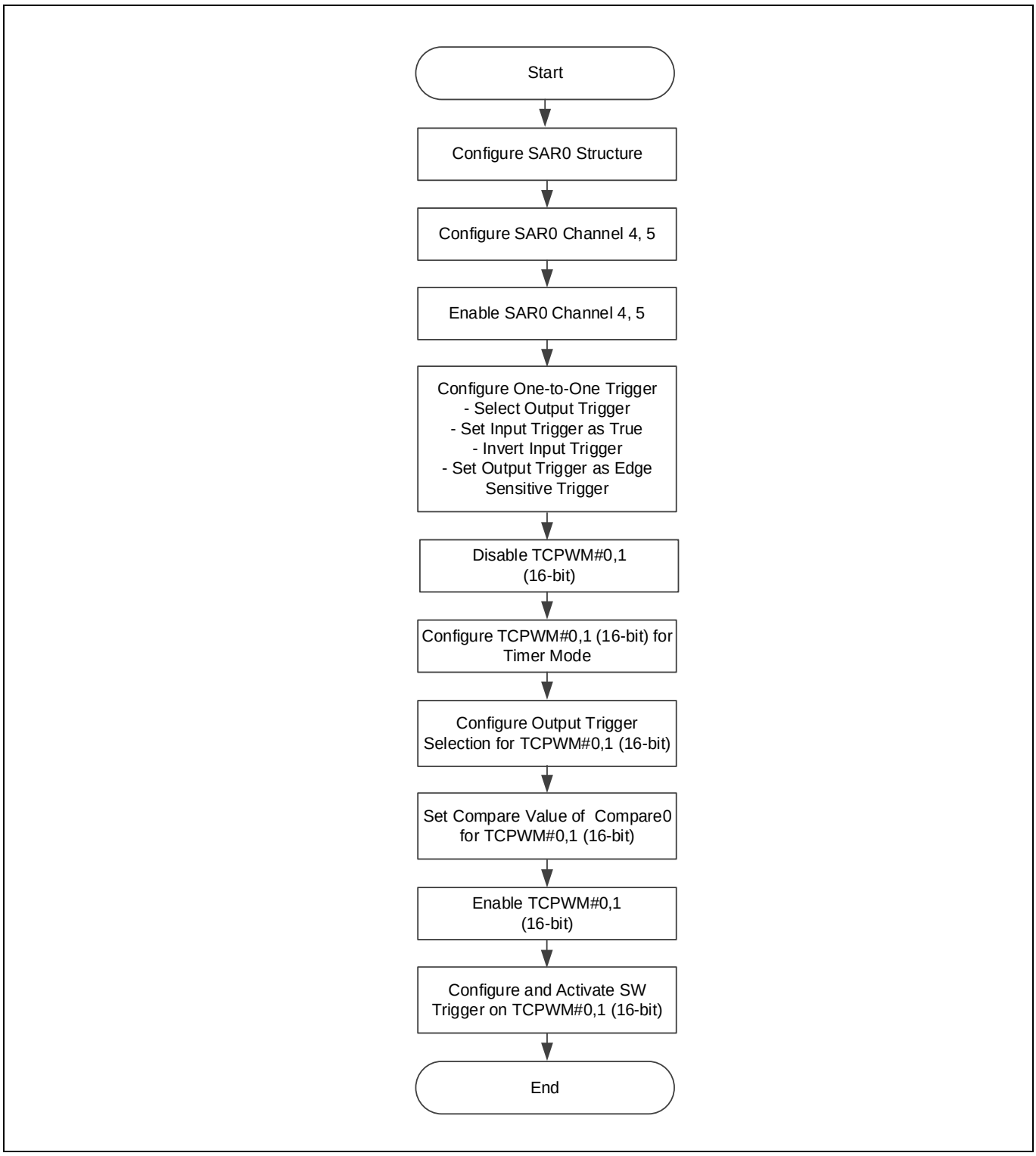

Figure 25 Setting for one-to-one trigger in XMC7200

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

Application

Trigger Multiplexer setting

Table 14 lists the values to be set. These tables are extracted from the XMC7200 datasheet that provides the trigger group and input trigger.

Table 14 Trigger one-to-one MUX Group 7 of XMC7200

Input trigger k	Trigger inputs	Trigger outputs
MUX Group 7: TCPWM1 to PASS direct connect		
0	TCPWM1_16M_TR_OUT1[0]	PASS0_CH_TR_IN[0]
1	TCPWM1_16M_TR_OUT1[3]	PASS0_CH_TR_IN[1]
2	TCPWM1_16M_TR_OUT1[6]	PASS0_CH_TR_IN[2]
3	TCPWM1_16M_TR_OUT1[9]	PASS0_CH_TR_IN[3]
4:31	TCPWM1_16_TR_OUT1[0:27]	PASS0_CH_TR_IN[4:31]
32	TCPWM1_16M_TR_OUT1[1]	PASS0_CH_TR_IN[32]
33	TCPWM1_16M_TR_OUT1[4]	PASS0_CH_TR_IN[33]
34	TCPWM1_16M_TR_OUT1[7]	PASS0_CH_TR_IN[34]
35	TCPWM1_16M_TR_OUT1[10]	PASS0_CH_TR_IN[35]
36:63	TCPWM1_16_TR_OUT1[28:55]	PASS0_CH_TR_IN[36:63]
64	TCPWM1_16M_TR_OUT1[2]	PASS0_CH_TR_IN[64]
65	TCPWM1_16M_TR_OUT1[5]	PASS0_CH_TR_IN[65]
66	TCPWM1_16M_TR_OUT1[8]	PASS0_CH_TR_IN[66]
67	TCPWM1_16M_TR_OUT1[11]	PASS0_CH_TR_IN[67S]
68:95	TCPWM1_16_TR_OUT1[56:83]	PASS0_CH_TR_IN[68:95]

The following describes how to connect SAR0 channel 4 and channel 5 to 16-bit TCPWM channel 1.

Note: 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 7 (i=7)
   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_GR7_TR_CTL4`, and `PERI_TR_1TO1_GR7_TR_CTL5`, respectively.
2. Input trigger configuration:
   Typically, the input trigger of the 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 [1].

3.3.2 Configuration

Figure 26 shows the parameters and values for SAR2 global configurator in the device configurator tool.

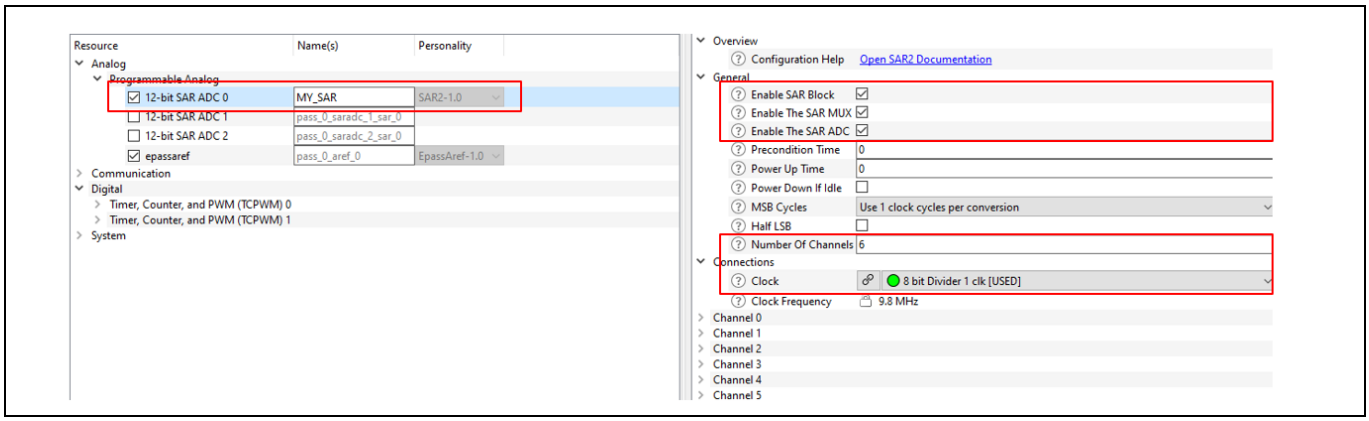

Figure 26 SAR2 global configuration in device configurator tool

Figure 27 shows the parameters and values for SAR2 channel_4 and channel_5 configutation in the device configurator tool.

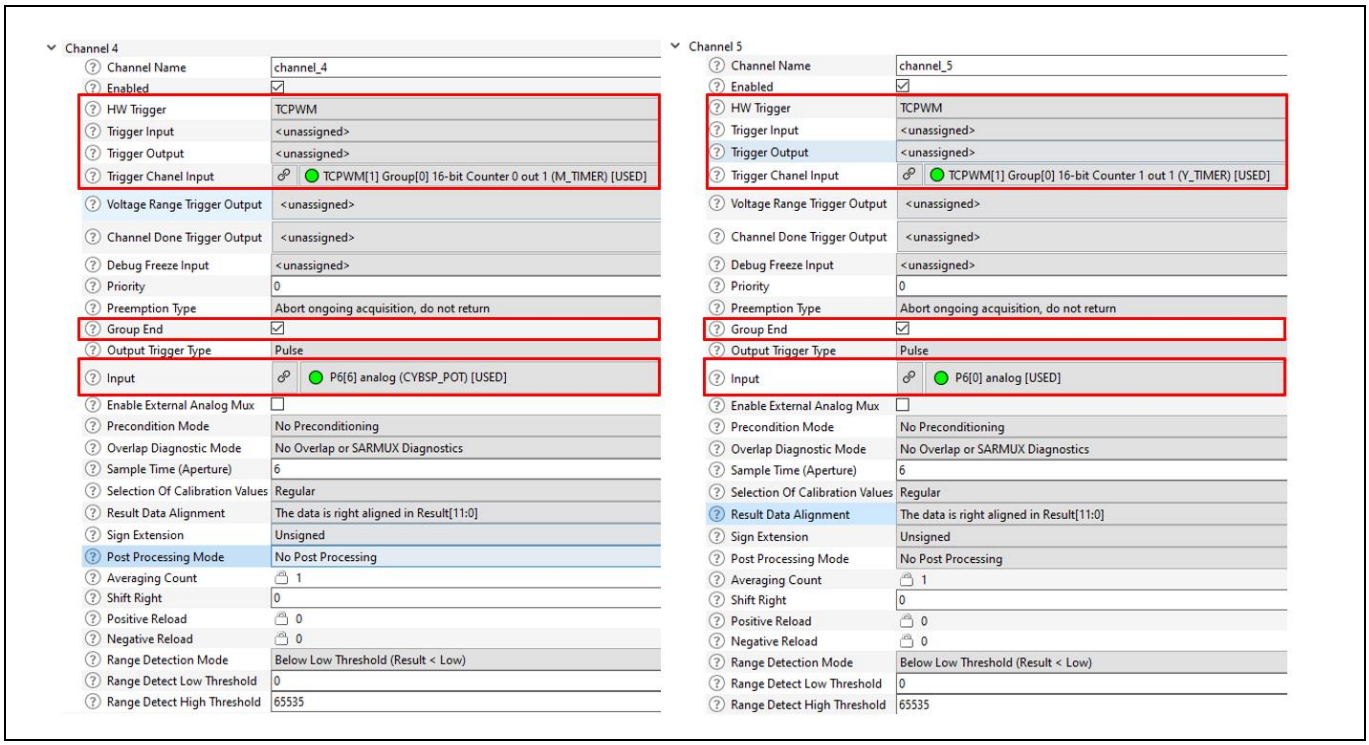

Figure 27 SAR2 channel_4 and channel_5 configuration in device configurator tool

Application

Figure 28 shows the parameters and values of TCPWM timer configuration for SAR channel_4 in the device configurator tool.

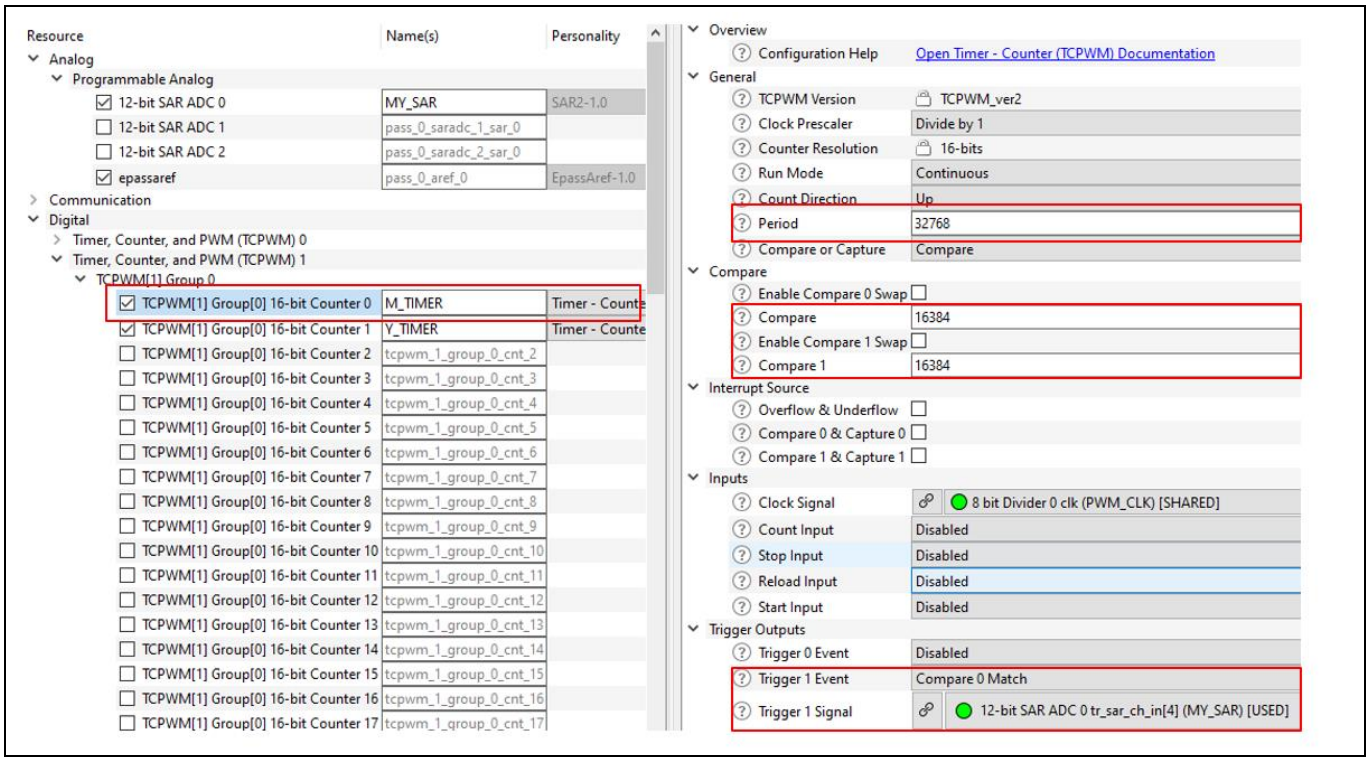

Figure 28 TCPWM Timer configuration for SAR channel_4 in device configurator tool

Figure 29 shows the parameters and values of TCPWM Timer configuration for SAR channel_5 in the device configurator tool.

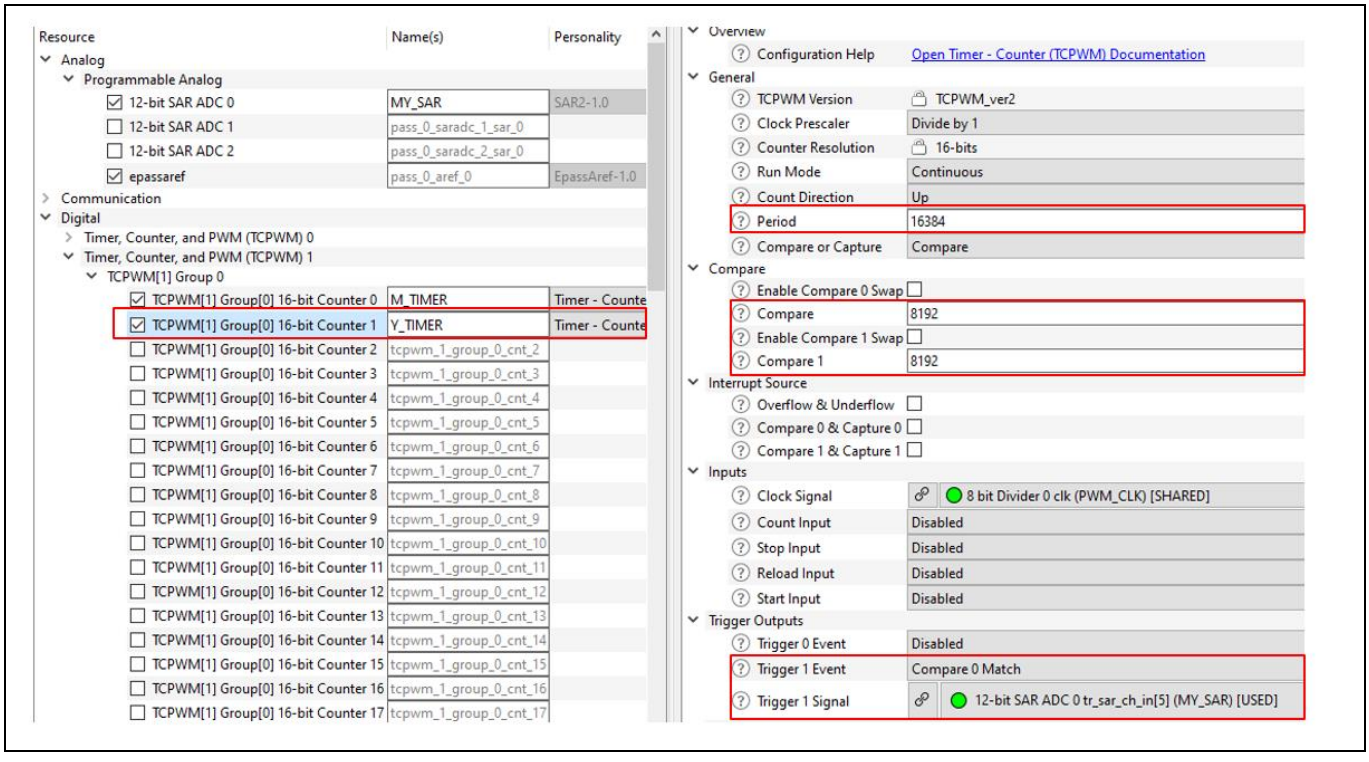

Figure 29 TCPWM Timer configuration for SAR channel_5 in device configurator tool

3.3.3 Example program for triggering ADC conversion by TCPWM timer

Code Listing 3 demonstrates the example program for Figure 25.

Code Listing 3 Example of triggering ADC conversion by TCPWM timer

```
/* SAR0 interrupt configuration structure */
const cy_stc_sysint_t SAR0_IRQ_cfg = {
    .intrSrc = ((NvicMux3_IRQn << 16) | MY_SAR_CH4_IRQ),
    .intrPriority = 7u
};

/* SAR1 interrupt configuration structure */
const cy_stc_sysint_t SAR1_IRQ_cfg = {
    .intrSrc = ((NvicMux3_IRQn << 16) | MY_SAR_CH5_IRQ),
    .intrPriority = 7u
};

static bool sar0_isr_set = false;

static bool sar1_isr_set = false;



int main(void)
{
    cy_rslt_t result;
    uint16_t sar0Result4 = 0u;
    uint16_t sar0Result5 = 0u;

    /* Initialize the device and board peripherals */
    result = cybsp_init();

    /* Board init failed. Stop program execution */
    if (result != CY_RSLT_SUCCESS)
    {
        CY_ASSERT(0);
    }

    /* Enable global interrupts */
    __enable_irq();

    /* Initialize retarget-io to use the debug UART port */
         result = cy_retarget_io_init(CYBSP_DEBUG_UART_TX, CYBSP_DEBUG_UART_RX, CY_RETARGET_IO_BAUDRATE);
         if (result != CY_RSLT_SUCCESS)
         {
                 CY_ASSERT(0);
         }

         /* \x1b[2J\x1b[;H - ANSI ESC sequence for clear screen */
    printf("\x1b[2J\x1b[;H");
         printf("*******************************************************************************\r\n");
         printf("XMC7000 MCU: TCPWM Timer trigger two ADC channels conversion\r\n");
         printf("*******************************************************************************\r\n");

    /*Analog resource initialize*/
    init_analog_resources();

    /* Initialize Timer using the config structure generated using device configurator*/
    Cy_TCPWM_Counter_Init(Y_TIMER_HW, Y_TIMER_NUM, &Y_TIMER_config);
    Cy_TCPWM_Counter_Init(M_TIMER_HW, M_TIMER_NUM, &M_TIMER_config);

    /* Enable the initialized Timer */
    Cy_TCPWM_Counter_Enable(M_TIMER_HW, M_TIMER_NUM);
    Cy_TCPWM_Counter_Enable(Y_TIMER_HW, Y_TIMER_NUM);

    /* Then start the counter */
         Cy_TCPWM_TriggerStart_Single(M_TIMER_HW, M_TIMER_NUM);
         Cy_TCPWM_TriggerStart_Single(Y_TIMER_HW, Y_TIMER_NUM);

    for (;;)
    {
         if(sar0_isr_set)
            {
               sar0Result4 = Cy_SAR2_Channel_GetResult(MY_SAR_HW, 4u, NULL);
               printf("sar0Result4 = %d \r\n",sar0Result4 );
                           sar0_isr_set = false;
            }
         if(sar1_isr_set)
                    {
                           sar0Result5 = Cy_SAR2_Channel_GetResult(MY_SAR_HW, 5u, NULL);
                           printf("sar0Result5 = %d \r\n",sar0Result5 );
                           sar1_isr_set = false;
```

Application

Code Listing 3 Example of triggering ADC conversion by TCPWM timer

```
                    }
    }

}

static void sar0_interrupt_handler(void)
{
    /* Check if End-Of-Scan trigger has occurred. If yes, set sar0_isr_set flag to true  */
    if (Cy_SAR2_Channel_GetInterruptStatus(MY_SAR_HW,4u) == CY_SAR2_INT_GRP_DONE)
    {
        sar0_isr_set = true;
    }

    /* Clear the interrupts */
    Cy_SAR2_Channel_ClearInterrupt(MY_SAR_HW, 4u ,CY_SAR2_INT_GRP_DONE);
}

static void sar1_interrupt_handler(void)
{
    /* Check if End-Of-Scan trigger has occurred. If yes, set sar0_isr_set flag to true  */
    if (Cy_SAR2_Channel_GetInterruptStatus(MY_SAR_HW,5u) == CY_SAR2_INT_GRP_DONE)
    {
        sar1_isr_set = true;
    }

    /* Clear the interrupts */
    Cy_SAR2_Channel_ClearInterrupt(MY_SAR_HW, 5u ,CY_SAR2_INT_GRP_DONE);
}
/*******************************************************************************
* Function Name: init_analog_resources
********************************************************************************
* Summary:
* This function initializes analog components - CTBM, SAR ADC and CTDAC
*
* Parameters:
*  void
*
* Return:
*  void
*
*******************************************************************************/
static void init_analog_resources(void)
{
    /* Variable to capture return value of functions */
    cy_en_sar2_status_t status;

    /* Initialize SAR0 */
    status = Cy_SAR2_Init(MY_SAR_HW, &MY_SAR_config);
    if (CY_SAR2_SUCCESS != status)
    {
        CY_ASSERT(0);
    }

    /*Enable SAR0*/
    Cy_SAR2_Enable(MY_SAR_HW);

    /*Configure and register SAR0 channel4 interrupts*/
    Cy_SysInt_Init(&SAR0_IRQ_cfg, sar0_interrupt_handler);
    NVIC_ClearPendingIRQ((IRQn_Type)SAR0_IRQ_cfg.intrSrc);

    /*Configure and register SAR0 channel5 interrupts*/
    Cy_SysInt_Init(&SAR1_IRQ_cfg, sar1_interrupt_handler);

        NVIC_ClearPendingIRQ((IRQn_Type)SAR1_IRQ_cfg.intrSrc);

    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);

    /*Initialize SAR0 channel 4*/
    Cy_SAR2_Channel_Init(MY_SAR_HW, 4u, &MY_SAR_channel_4_config);
    /*Initialize SAR0 channel 5*/
    Cy_SAR2_Channel_Init(MY_SAR_HW, 5u, &MY_SAR_channel_5_config);

    /*Set SAR0 channel 4 interrupt mask*/
    Cy_SAR2_Channel_SetInterruptMask(MY_SAR_HW, 4u,CY_SAR2_INT_GRP_DONE);
    /*Set SAR0 channel 5 interrupt mask*/
    Cy_SAR2_Channel_SetInterruptMask(MY_SAR_HW, 5u,CY_SAR2_INT_GRP_DONE);

}
```

Note: The highlighted sections of the code snippet are not explained in this application note. For details, see the architecture reference manual [2].

3.4 Simultaneous starting of TCPWM timer by SW trigger

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

3.4.1 Use case description of simultaneous starting of TCPWM timer by SW trigger

Figure 30 shows an example application of SW trigger in XMC7200. In 120-degree commutation control, three TCPWM 16-bit Motor Control counter channels 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 31 demonstrates the operation. The connection between trigger MUX Group 4 and TCPWM 16-bit motor control counter is illustrated in Figure 32. For more details on TCPWM, see the TCPWM chapter of the architecture reference manual [2].

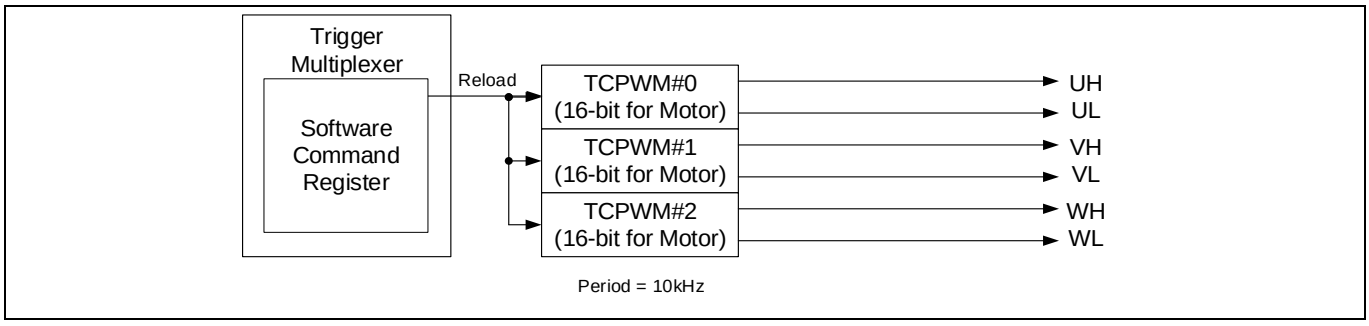

Figure 30 Example of SW trigger in XMC7200

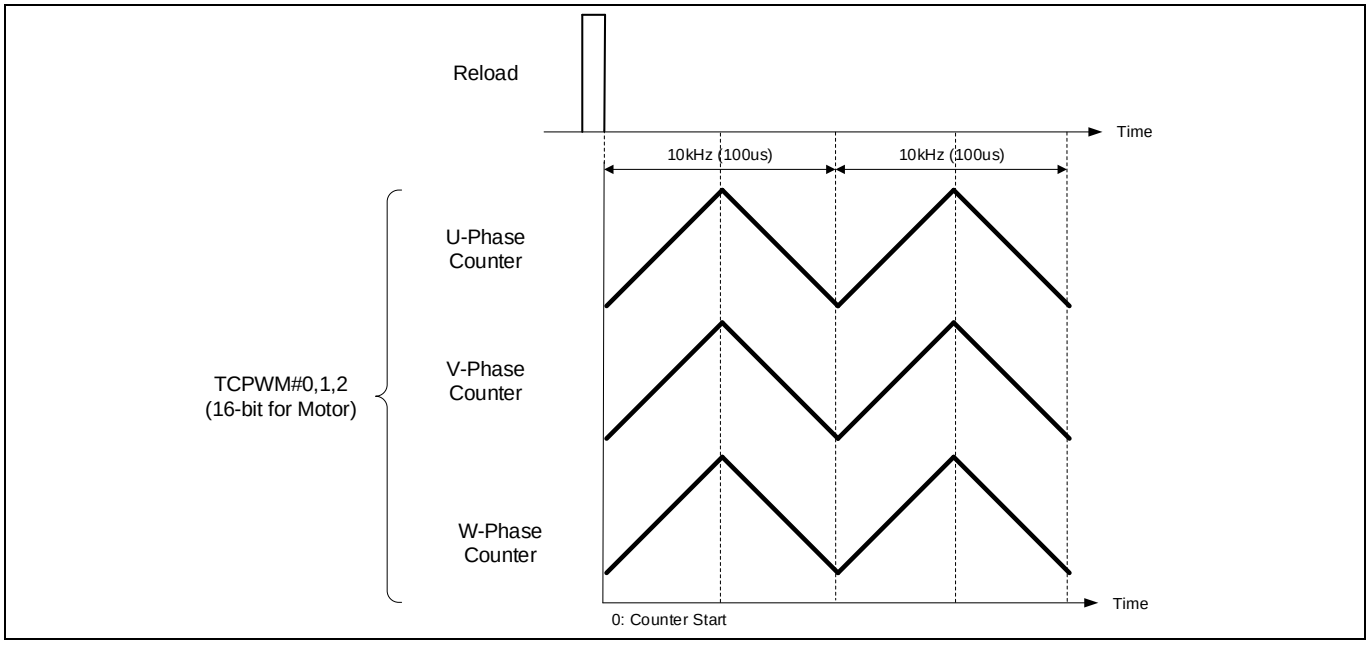

Figure 31 Example operation: Simultaneous starting of TCPWM timer by SW trigger

Application

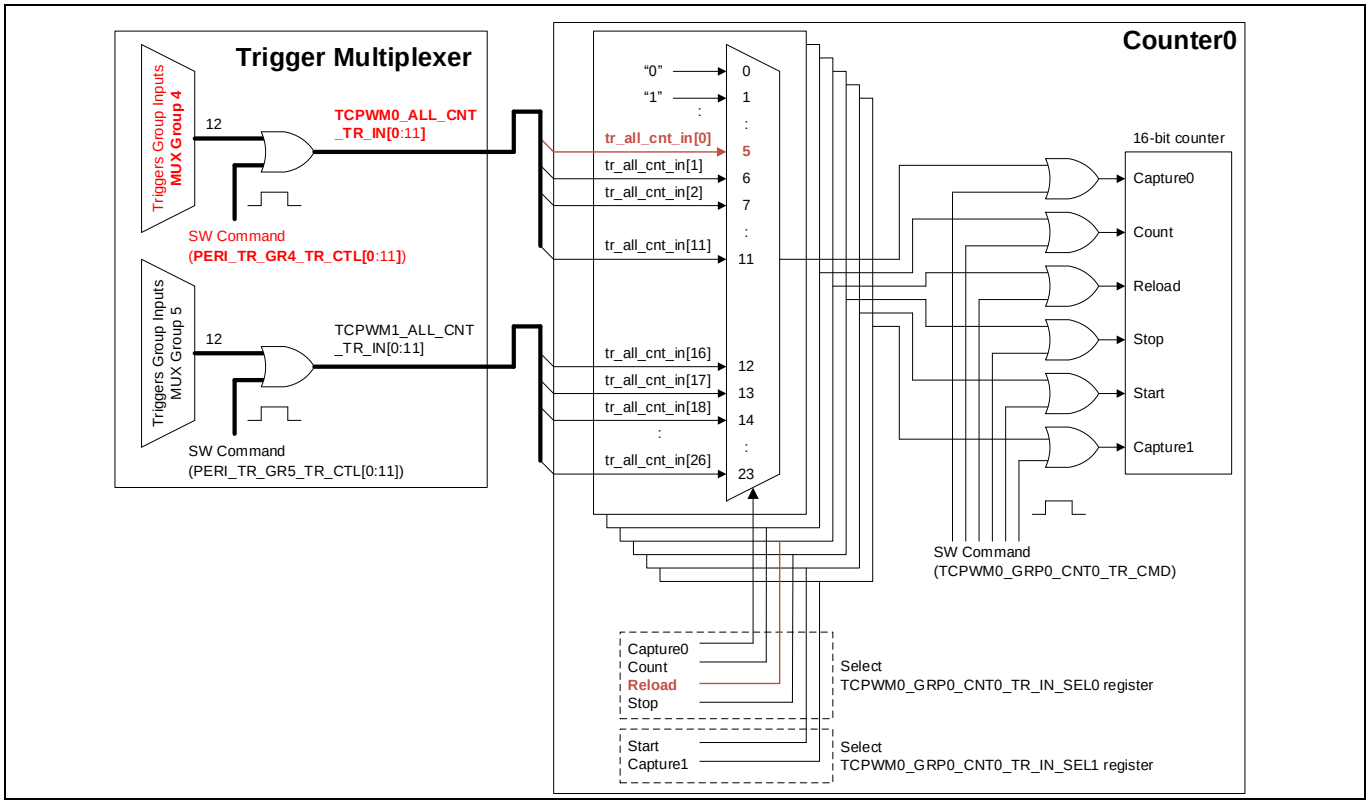

Figure 32 Connection between MUX Group 4 and TCPWM 16-bit motor control counter

Application

Triggers must be set to implement this application. Figure 33 lists the procedure for setting the trigger.

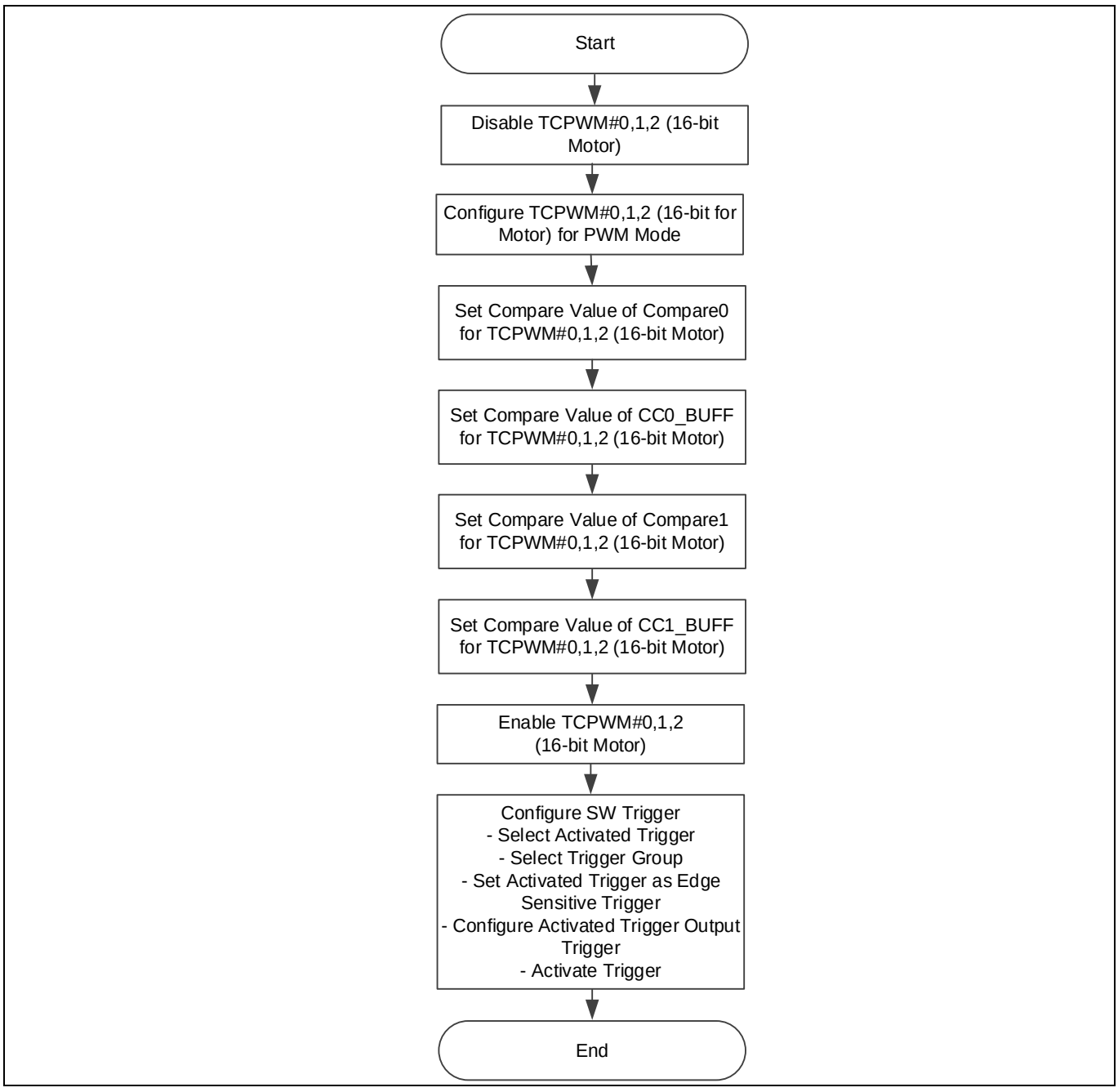

Figure 33 Setting for one-to-one trigger in XMC7200

Application

The configuration for the SW trigger in Step (8) is 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). This table is extracted from the XMC7200 datasheet that provides the trigger group and output trigger.

Table 15 Trigger outputs of MUX Group 4 of XMC7200

Output (“k”)	Trigger label	Description
MUX Group 4: TCPWM_OUT (TCPWM0 to P-DMA0 trigger multiplexer)		
0:11	TCPWM_ALL_CNT_TR_IN[0:11]	All counters trigger input

For the trigger group outputs table of each series, see the device datasheets [1].

Figure 34 shows SW triggering the TCPWM counter.

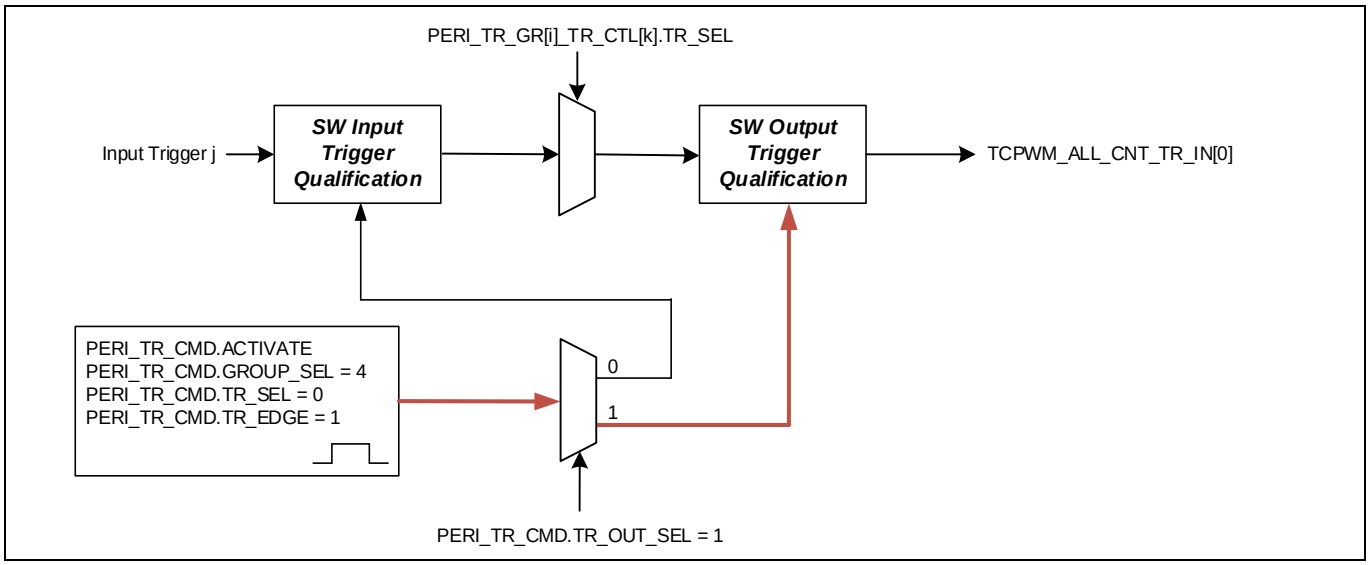

Figure 34 SW triggering TCPWM counter

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 Configuration

Figure 35 shows the parameters and values for TCPWM configuration in the device configurator tool.

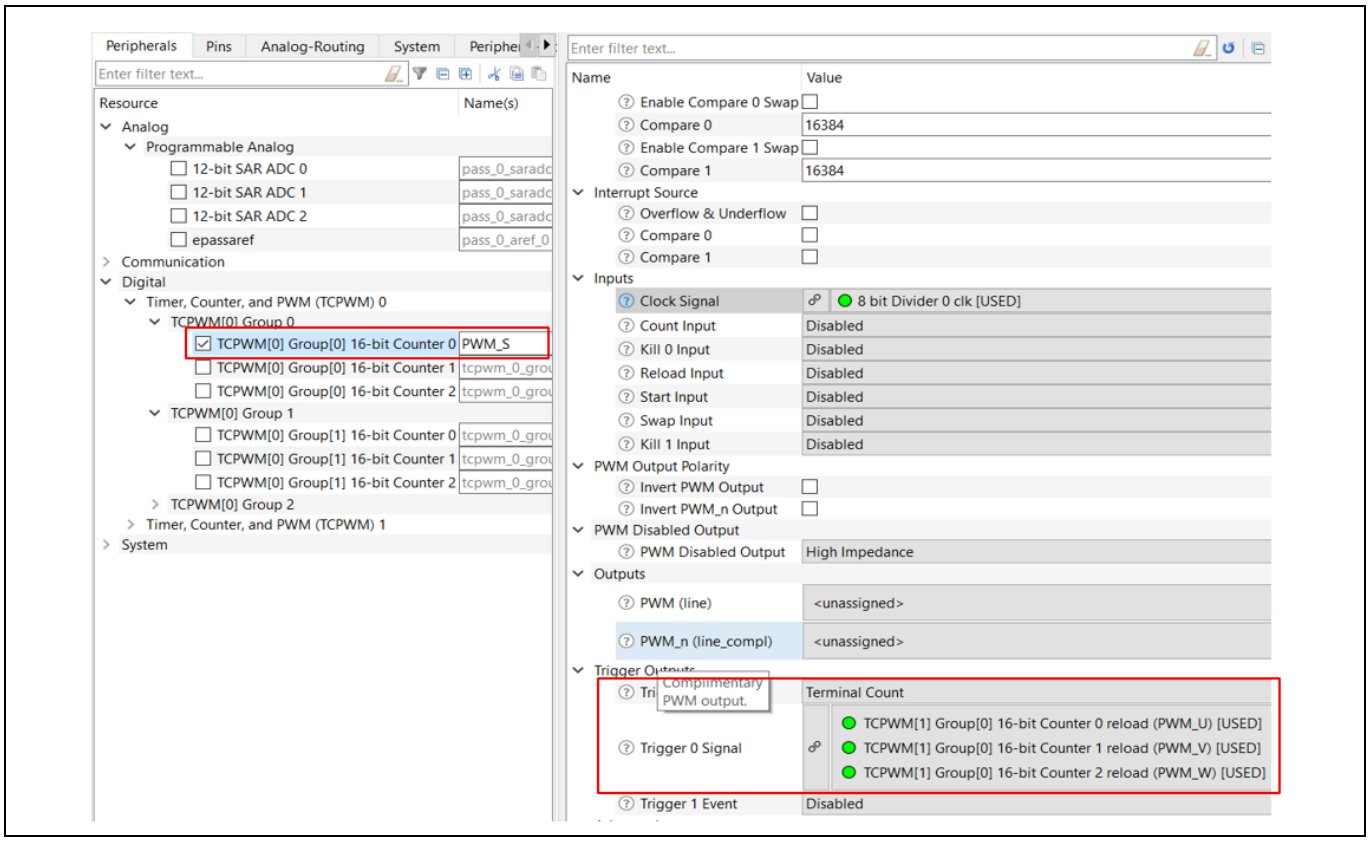

Figure 35 TCPWM configuration in the device configurator tool

Figure 36 shows the parameters and values for three units TCPWM configuration in the device configurator tool.

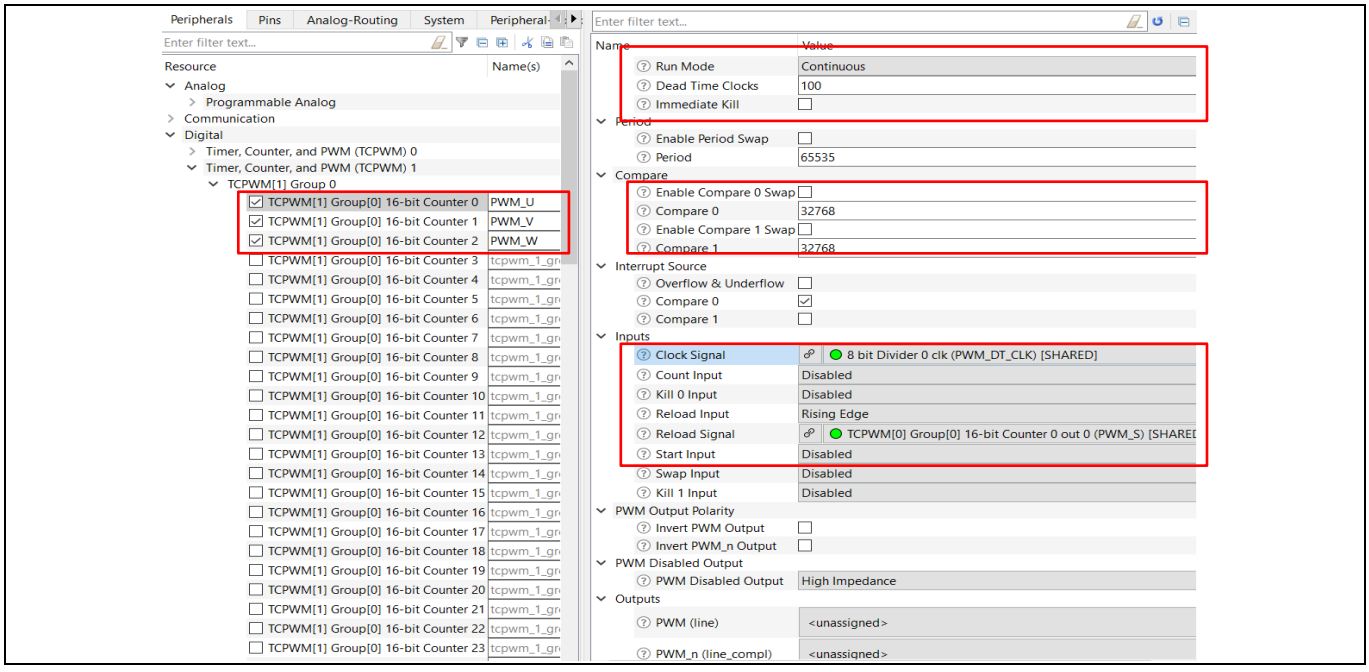

Figure 36 Three units TCPWM configuration in the device configurator tool

3.4.3 Example program for simultaneous TCPWM timer start by SW trigger

Code Listing 4 shows the example program for Figure 33.

Code Listing 4 Example of starting TCPWM timer simultaneous by SW trigger

```
/* PWM_U, PWM_V, PWM_W CC0 interrupts configuration structure */
const cy_stc_sysint_t PWM_U_cfg = {
    .intrSrc = ((NvicMux3_IRQn << 16) | tcpwm_1_interrupts_0_IRQn),
    .intrPriority = 7u
};

const cy_stc_sysint_t PWM_V_cfg = {
    .intrSrc = ((NvicMux3_IRQn << 16) | tcpwm_1_interrupts_1_IRQn),
    .intrPriority = 7u
};

const cy_stc_sysint_t PWM_W_cfg = {
    .intrSrc = ((NvicMux3_IRQn << 16) | tcpwm_1_interrupts_2_IRQn),
    .intrPriority = 7u
};

/*******************************************************************************
* Function Definitions
*******************************************************************************/
int main(void)
{
          cy_rslt_t result;

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

          /*Enable global interrupt*/
          __enable_irq();

          /*Initial the BSP user LED1, LED2, LED3*/
          Cy_GPIO_Pin_Init(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_PIN, &CYBSP_USER_LED1_config );
          Cy_GPIO_Pin_Init(CYBSP_USER_LED2_PORT, CYBSP_USER_LED2_PIN, &CYBSP_USER_LED2_config );
          Cy_GPIO_Pin_Init(CYBSP_USER_LED3_PORT, CYBSP_USER_LED3_PIN, &CYBSP_USER_LED3_config );

          /*PWM_U Dead Time Mode initial*/
          if (CY_TCPWM_SUCCESS != Cy_TCPWM_PWM_Init(PWM_U_HW, PWM_U_NUM, &PWM_U_config))
                {
                   CY_ASSERT(0);
                }
         /* Enable the PWM_U */
         Cy_TCPWM_PWM_Enable(PWM_U_HW, PWM_U_NUM);

          /*PWM_W Dead Time Mode initial*/
    if (CY_TCPWM_SUCCESS != Cy_TCPWM_PWM_Init(PWM_W_HW, PWM_W_NUM, &PWM_W_config))
                    {
                       CY_ASSERT(0);
                    }
   /* Enable the PWM_W */
    Cy_TCPWM_PWM_Enable(PWM_W_HW, PWM_W_NUM);

    /*PWM_V Dead Time Mode initial*/
    if (CY_TCPWM_SUCCESS != Cy_TCPWM_PWM_Init(PWM_V_HW, PWM_V_NUM, &PWM_V_config))
            {
                    CY_ASSERT(0);
            }
    /* Enable the PWM_W */
    Cy_TCPWM_PWM_Enable(PWM_V_HW, PWM_V_NUM);

    /*Register and enable PWM_U, PWM_V,PWM_W interrupts*/
    pwms_interrupt_initial();

    /*PWM_S PWM Mode initial*/
    if (CY_TCPWM_SUCCESS != Cy_TCPWM_PWM_Init(PWM_S_HW, PWM_S_NUM, &PWM_S_config))
          {
              CY_ASSERT(0);
          }
    /* Enable the PWM_S */
    Cy_TCPWM_PWM_Enable(PWM_S_HW, PWM_S_NUM);

    /* Then start the PWM_S to synchronous PWM_U, PWM_V and PWM_W */
    Cy_TCPWM_TriggerReloadOrIndex_Single(PWM_S_HW, PWM_S_NUM);

    for(;;)
```

Application

Code Listing 4 Example of starting TCPWM timer simultaneous by SW trigger

```
    {

    }
}

static void pwms_interrupt_initial(void)
{
        /*Configure and register pwm_u interrupt*/
        Cy_SysInt_Init(&PWM_U_cfg, pwm_u_interrupt_handler);
        NVIC_ClearPendingIRQ((IRQn_Type)PWM_U_cfg.intrSrc);

        /*Configure and register pwm_v interrupt*/
    Cy_SysInt_Init(&PWM_V_cfg, pwm_v_interrupt_handler);
        NVIC_ClearPendingIRQ((IRQn_Type)PWM_V_cfg.intrSrc);

        /*Configure and register pwm_w interrupt*/
        Cy_SysInt_Init(&PWM_W_cfg, pwm_w_interrupt_handler);
        NVIC_ClearPendingIRQ((IRQn_Type)PWM_W_cfg.intrSrc);
        NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);

}

static void pwm_u_interrupt_handler(void)
{
        /*toggle user LED1*/
        Cy_GPIO_Inv(CYBSP_USER_LED1_PORT, CYBSP_USER_LED1_PIN);

        /*clear the counter CC0 compare interrupt*/
        Cy_TCPWM_ClearInterrupt(PWM_U_HW, PWM_U_NUM, CY_TCPWM_INT_ON_CC0);
}

static void pwm_v_interrupt_handler(void)
{
        /*toggle user LED1*/
        Cy_GPIO_Inv(CYBSP_USER_LED2_PORT, CYBSP_USER_LED2_PIN);

        /*clear the counter CC0 compare interrupt*/
        Cy_TCPWM_ClearInterrupt(PWM_V_HW, PWM_V_NUM, CY_TCPWM_INT_ON_CC0);
}

static void pwm_w_interrupt_handler(void)
{
        /*toggle user LED1*/
        Cy_GPIO_Inv(CYBSP_USER_LED3_PORT, CYBSP_USER_LED3_PIN);

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

The highlighted sections of the code snippet are not explained in this application note. For details, see the architecture reference manual [2].

Glossary

Table 16 Glossary

Terms	Description
MUX	Multiplexer
OV	Overflow
P-DMA (DW)	Peripheral direct memory access
SAR ADC	Successive approximation register analog-to-digital converter
SCB	Serial communications block
SW	Software
TCPWM	Timer, Counter, and Pulse Width Modulator
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

References

Contact Technical Support to obtain XMC7000 family reference documents.

[1] Device datasheets

- 002-33896; XMC7100 series 32-bit Arm® Cortex®-M7 microcontroller datasheet
- 002-33522; XMC7200 series 32-bit Arm® Cortex®-M7 microcontroller datasheet

[2] Reference manuals

- 002-33816; XMC7000 MCU family architecture technical reference manual
- 002-33817; XMC7100 register technical reference manual
- 002-33812; XMC7200 register technical reference manual

[3] Application notes

- 002-34802; AN234802 - Secure system configuration in XMC7000 family
- 002-34334; AN234334 - Getting started with XMC7000 MCU on ModusToolbox™ software
- 002-34254; AN234254 - Multi-core handling guide in XMC7000
- 002-34119; AN234119 - Timer, Counter, and PWM _TCPWM_ usage in XMC7000 family
- AN234282 - Using a SAR ADC in XMC7000 Family
- AN234225 XMC7000_ Using direct memory access _DMA_ controllers

Revision history

Document revision	Date	Description of changes
**	2021-11-29	Initial release
*A	2022-04-25	Updated code snippet follow new version PDL(V3.0) Updated Figure 11, Figure 18, Figure 25, Figure 33
*B	2024-03-13	Updated all of the document device configurator pictures. Updated all of the document examples code. Deleted PDL driver source code.

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

Edition 2024-03-13

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-34197 Rev *B

Important notice

The information contained in this application note is given as a hint for the implementation of the product only and shall in no event be regarded as a description or warranty of a certain functionality, condition or quality of the product. Before implementation of the product, the recipient of this application note must verify any function and other technical information given herein in the real application. 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) with respect to any and all information given in this application note.

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.

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.