Using a SAR ADC in XMC7000 family

About this document

Scope and purpose

AN234282 demonstrates how to configure and use a SAR ADC in XMC7000 family with a software trigger, a hardware trigger, group processing, averaging, range detection, pulse detection, diagnosis, and calibration. This document is based on using the Device Configurator to configure the ADC parameters.

Associated Part Family

XMC7000 family of XMC™ industrial microcontrollers.

Table of contents

Table of contents

1 Introduction

This application note describes how to use a SAR ADC for Infineon XMC7000 family. The SAR ADC converts analog input voltages into digital values. The analog channels can be individual or grouped. Each channel can be triggered by software or hardware. The SAR ADC features averaging, range detection, pulse detection, diagnosis, and calibration.

The Device Configurator is part of a collection of tools included with the ModusToolbox™. Use the Device Configurator to enable and configure device peripherals, such as clocks and pins, as well as standard MCU peripherals that do not require their own tool. To understand the functionality described and terminology used in this application note, see the “SAR ADC” chapter of the architecture reference manual.

2 Software trigger procedure

Figure 1 shows an example application that converts voltage values given to pin ADC[0]_0 of the MCU to digital values. Analog-to-digital conversion is repeated in the interrupt service routine (ISR) by using a software trigger.

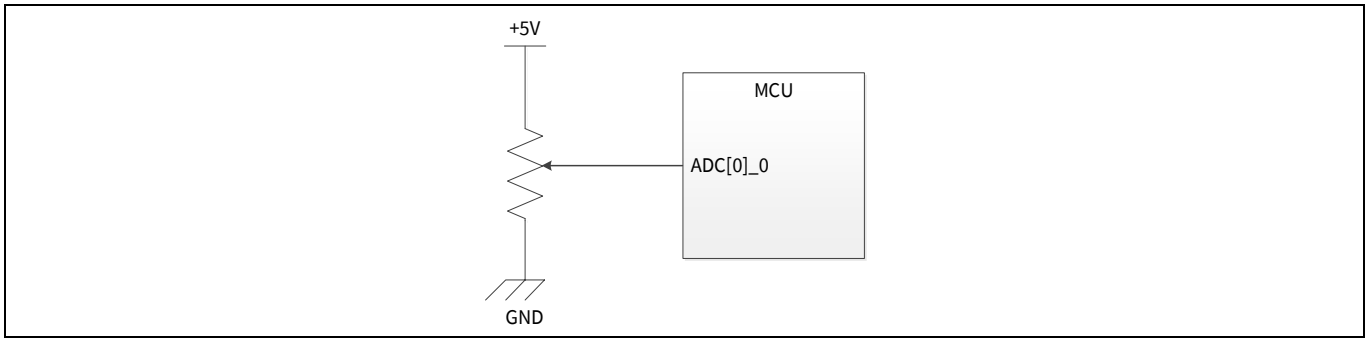

Figure 1 Example of analog-to-digital conversion connection

To implement this application, use the following sections that describe the procedure for setting the ADC channel. These sections also provide an example for using a software trigger.

2.1 Basic ADC global configuration

This section describes how to configure the ADC based on a use case using the Peripheral Driver Library (PDL). The code snippets in this application note are part of PDL. See References.

The driver part configures each register based on the parameter values in the configuration part.

2.2 ADC global configuration

Configuring the common ADC settings in each channel involves the following:

- Enabling the ADC and configuring auto idle power down with the SARn_CTL register
- Configuring debug freeze with the PASS_PASS_CTL register

Figure 2 shows an example of the ADC global settings.

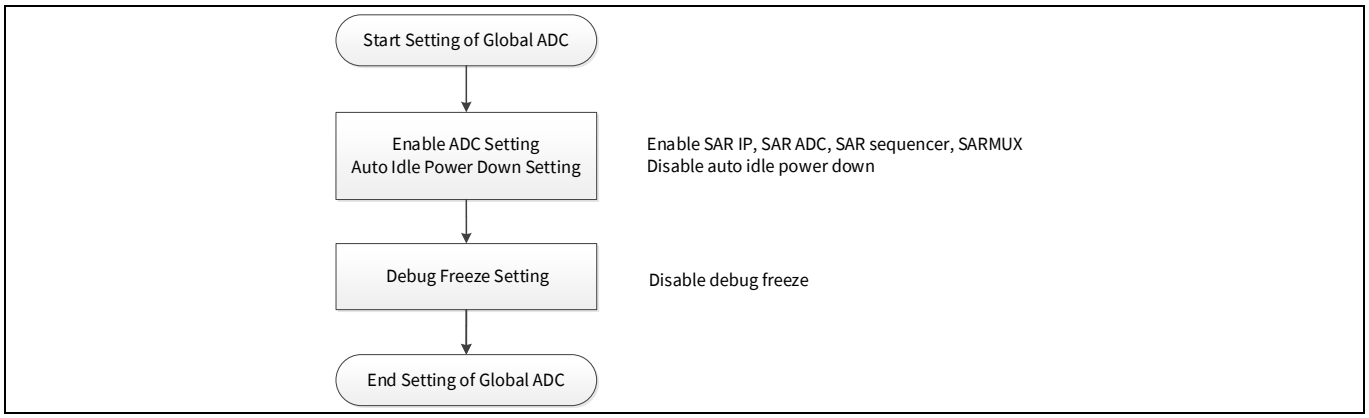

Figure 2 Example of ADC global configuration

2.2.1 Use case

The following use case is an example of ADC global setting:

- ADC logical channel: 0
- ADC operation frequency: 13 MHz
- Auto idle power down: 0 (Disable)
- Power-up time: 0 (Disable)
- SAR IP: 1 (Enable)
- SAR ADC and SARSEQ: 1(Enable)
- SARMUX: 1 (Enable)
- VDDA Voltage: 3.3 V

2.2.2 Configuration

Figure 3 shows the parameters of the configuration part in Device Configurator tool for SAR2 global setting.

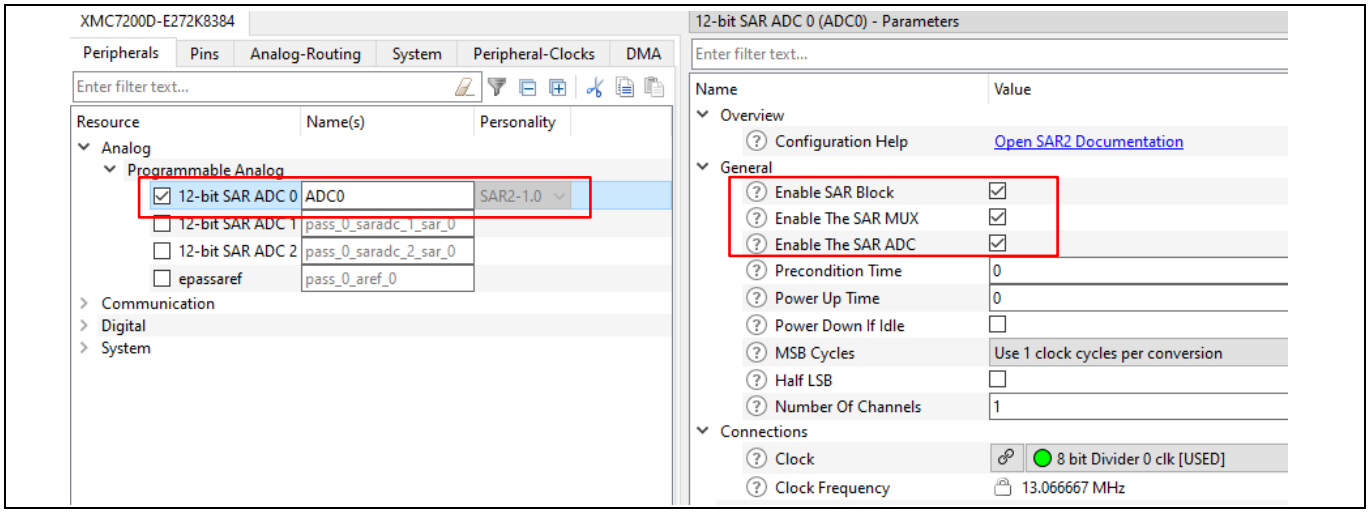

Figure 3 SAR2 global configuration in Device Configurator

Table 1 Functions for ADC global configuration

Functions	Description	Value
`Cy_SAR2_Init(PASS_SAR,ADC Configure)`	Initializes the ADC module	–

2.2.3 Sample code

This section demonstrates a sample code for the initial configuration of ADC global settings. The following description will help you understand the register notation of the driver part of the PDL:

- base->unENABLE.stcField.u1CHAN_EN is the PASS0_SAR0_CH0_ENABLE.CHAN_EN setting mentioned in the registers reference manual. Other registers are also described in the same manner.
- Performance improvement measures
- To improve the performance of register setting, the PDL writes a complete 32-bit data to the register. Each bit field is generated in advance in a bit-writable buffer and written to the register as the final 32-bit data.

Code Listing 1 General configuration of ADC global settings

```
#define ADC0_HW PASS0_SAR0
#define ADC0_CH0_HW PASS0_SAR0_CH0
#define ADC0_channel_0_HW ADC0_CH0_HW
#define ADC0_CH0_IRQ pass_0_interrupts_sar_0_IRQn
#define ADC0_channel_0_IRQ ADC0_CH0_IRQ
#define ADC0_VDDA_RANGE CY_SAR2_VDDA_2_7V_TO_4_5V
#ifndef SARMUX0_CH0_PORT_ADDR
#define SARMUX0_CH0_PORT_ADDR 0
#endif
#define ADC0_channel_0_IDX (0UL)

const cy_stc_sar2_config_t ADC0_config =
{
.preconditionTime = 0U,
.powerupTime = 0U,
.enableIdlePowerDown = false,
.msbStretchMode = CY_SAR2_MSB_STRETCH_MODE_1CYCLE,
.enableHalfLsbConv = false,
.sarMuxEnable = true,
.adcEnable = true,
.sarIpEnable = true,
.channelConfig = {(cy_stc_sar2_channel_config_t *)&ADC0_channel_0_config,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL},
};

int main(void)
{
:
```

Software trigger procedure

```
    /* 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 ADC */
    {
       /* Initialize the SAR2 module */
       Cy_SAR2_Init(ADC0_HW, &ADC0_config);
       /* Set ePASS MMIO reference buffer mode for bangap voltage */
       Cy_SAR2_SetReferenceBufferMode(PASS0_EPASS_MMIO,
CY_SAR2_REF_BUF_MODE_OFF);:
   }
:
    for(;;)
    {
    }
}
```

2.3 Logical channel configuration with software trigger

Each logical channel has one A/D conversion result register. A logical channel can be assigned as any analog input (`ADC[n]_i`) by configuring the `SARn_CHx_SAMPLE_CTL` register.

Figure 4 shows the example of logical channel configuration with a software trigger. In this example, the minimum value of the sample time is used. To find the proper sample time for your system, see the datasheet mentioned in References.

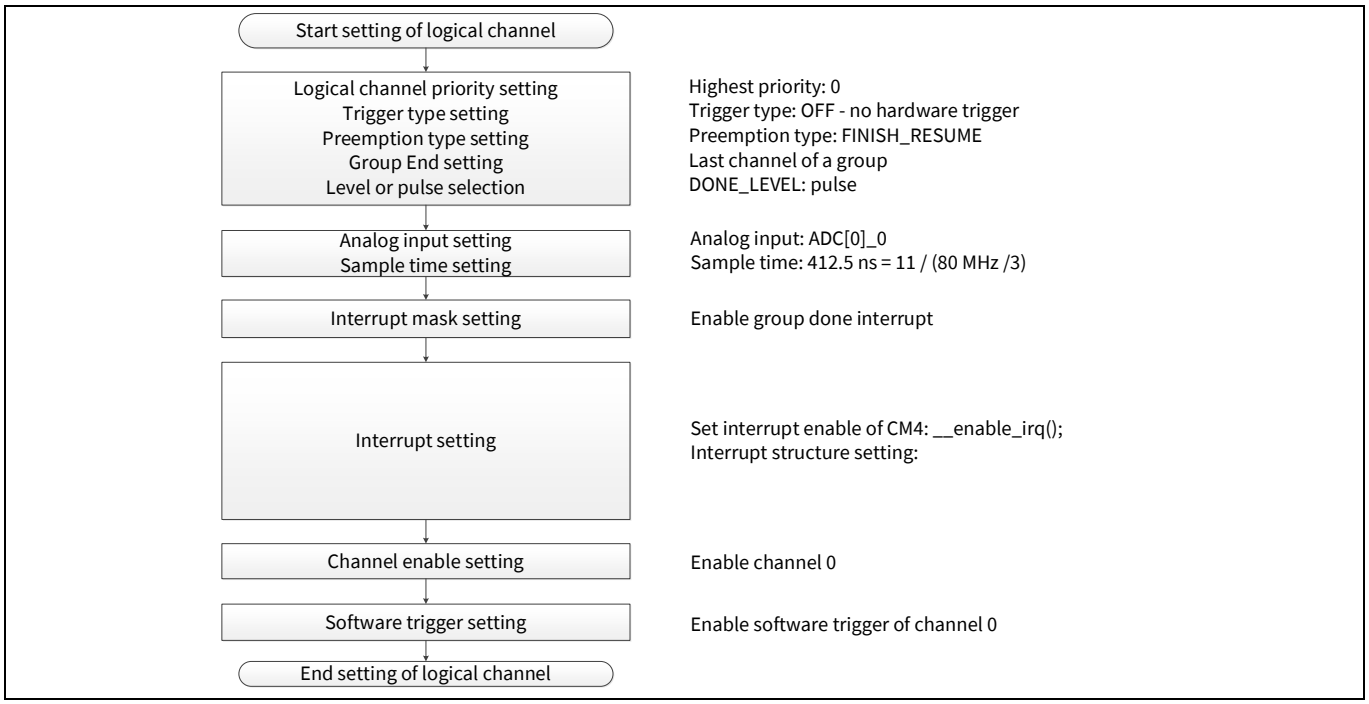

Figure 4 Example of logical channel setting with software trigger

Logical channel 0 is used in this example. Any logical channel will work in this application if proper analog input is selected.

2.3.1 Use case

The following use case is an example of configuring the logical channel with software trigger.

- Analog input: ADC[0]_0
- Sample time: 846 ns=11/13 MHz
- ADC trigger selection: OFF
- Channel priority: 0 (highest)
- Channel pre-emption type: Finish resume
- Channel group: The last channel
- ADC done level: Pulse
- ADC pin/port address: ADC[0]_0
- External analog MUX: 0, Enable
- Pre-conditioning mode: OFF
- Overlap diagnostics mode: OFF

Software trigger procedure

- Calibration value select: Regular
- Post processing mode: None
- Result alignment: Right
- Sign extension: Unsigned
- Averaging count: 0
- Shift right: 0
- Mask group: Done/Not cancelled/Not overflow
- Mask channel: Not range/Not pulse/Not overflow

2.3.2 Configuration

Channel configuration in Device Configurator and Table 2 lists the functions of the configuration part of in the PDL for configuring a logical channel with software trigger.

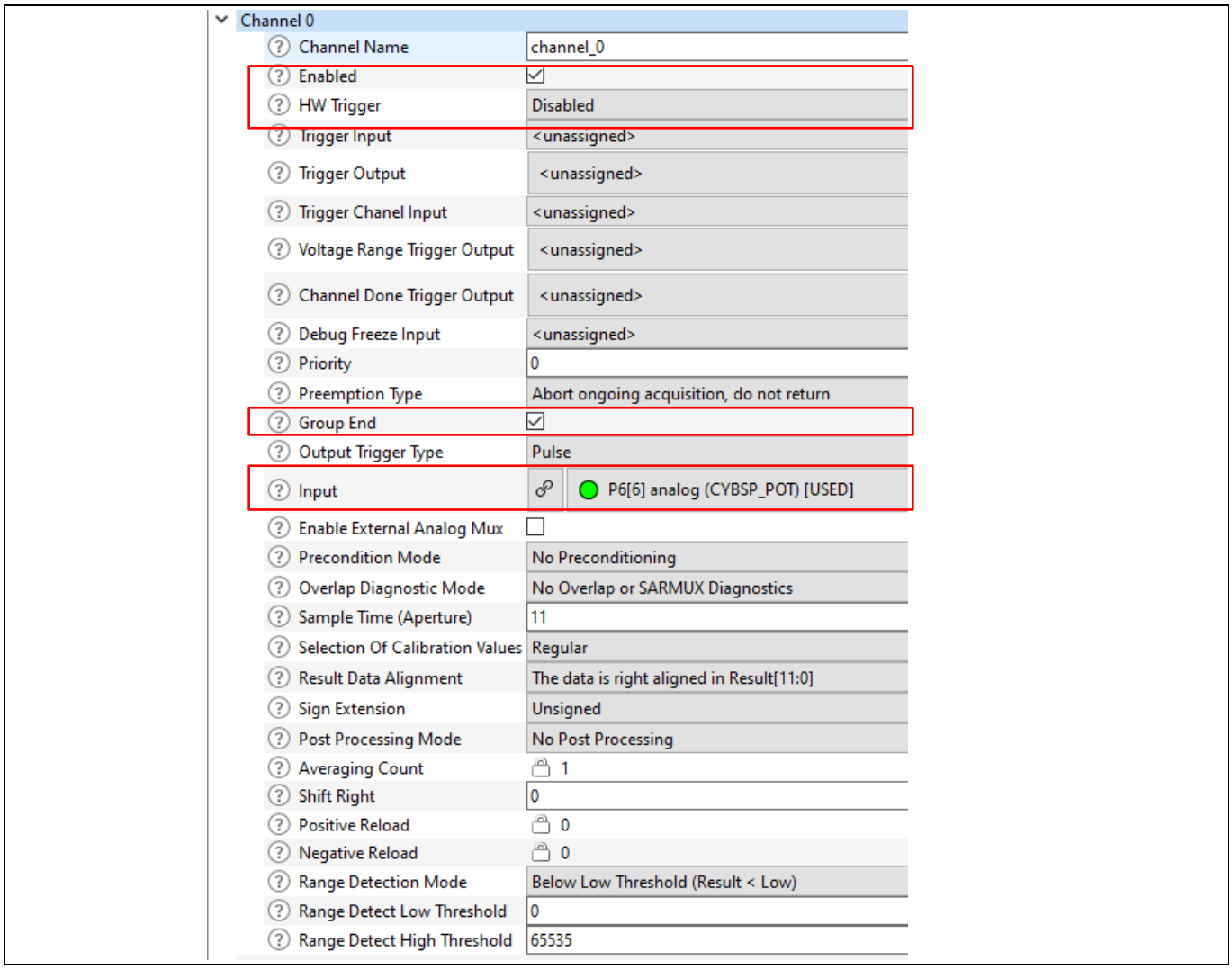

Figure 5 TCPWM timer configuration in Device Configurator

Software trigger procedure

Table 2 Functions for configuring a logical channel with software trigger

Functions	Description	Value
`Cy_SAR2_Channel_Init(PASS SARchannel,ADCchannel Configure)`	Initializes the ADC channel	–
`Cy_SAR2_Channel_Enable(SAR channel)`	Enables the corresponding channel	–
`Cy_SAR2_Channel_SoftwareTrigger(PASS SARchannel)`	Issues a software start trigger	–

2.3.3 Sample code

See Code Listing 2 for sample code for initial configuration of a logical channel with a software trigger.

Code Listing 2 General ADC global configuration

```
#define ADC0_HW PASS0_SAR0
#define ADC0_CH0_HW PASS0_SAR0_CH0
#define ADC0_channel_0_HW ADC0_CH0_HW
#define ADC0_CH0_IRQ pass_0_interrupts_sar_0_IRQn
#define ADC0_channel_0_IRQ ADC0_CH0_IRQ
#define ADC0_VDDA_RANGE CY_SAR2_VDDA_2_7V_TO_4_5V
#ifndef SARMUX0_CH0_PORT_ADDR
#define SARMUX0_CH0_PORT_ADDR 0
#endif
#define ADC0_channel_0_IDX (0UL)

const cy_stc_sar2_channel_config_t ADC0_channel_0_config =
{
.channelHwEnable = true,
.triggerSelection = CY_SAR2_TRIGGER_OFF,
.channelPriority = 0U,
.preenptionType = CY_SAR2_PREEMPTION_ABORT_CANCEL,
.doneLevel = CY_SAR2_DONE_LEVEL_PULSE,
.isGroupEnd = true,
.pinAddress = SARMUX0_CH0_PIN_ADDR,
.portAddress = SARMUX0_CH0_PORT_ADDR,
.extMuxEnable = false,
.extMuxSelect = 0U,
.preconditionMode = CY_SAR2_PRECONDITION_MODE_OFF,
.overlapDiagMode = CY_SAR2_OVERLAP_DIAG_MODE_OFF,
.sampleTime = 11U,
.calibrationValueSelect = CY_SAR2_CALIBRATION_VALUE_REGULAR,
```

Software trigger procedure

Code Listing 2 General ADC global configuration

```
.postProcessingMode = CY_SAR2_POST_PROCESSING_MODE_NONE,
.resultAlignment = CY_SAR2_RESULT_ALIGNMENT_RIGHT,
.signExtention = CY_SAR2_SIGN_EXTENTION_UNSIGNED,
.averageCount = 1U,
.rightShift = 0U,
.positiveReload = 0U,
.negativeReload = 0U,
.rangeDetectionMode = CY_SAR2_RANGE_DETECTION_MODE_BELOW_LO,
.rangeDetectionLoThreshold = 0U,
.rangeDetectionHiThreshold = 65535U,
};
int main(void)
{
    cy_rslt_t result;
    uint16_t               resultBuff;
    uint32_t adc_status = 0;


    /* 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);
      }
```

Software trigger procedure

Code Listing 2 General ADC global configuration

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

    /* Initialize ADC */

    /* Initialize the SAR2 module */
    Cy_SAR2_Init(ADC0_HW, &ADC0_config);
    /* Set ePASS MMIO reference buffer mode for bangap voltage */
    Cy_SAR2_SetReferenceBufferMode(PASS0_EPASS_MMIO,
CY_SAR2_REF_BUF_MODE_OFF);

    /* Initialize the SAR2 Channel */
    Cy_SAR2_Channel_Init(ADC0_HW,0, &ADC0_channel_0_config);

    /* Issue software start trigger */
    Cy_SAR2_Channel_SoftwareTrigger(ADC0_HW, 0);

    for (;;)
    {
        /* Get conversion results in counts */
      resultBuff = Cy_SAR2_Channel_GetResult(ADC0_HW, 0, &adc_status);
      if(CY_SAR2_STATUS_VALID == (adc_status & CY_SAR2_STATUS_VALID))
            {
            /* Print out the ADC conversion result by UART */
                  printf("ADC conversion complete, result: %d\r\n",
resultBuff);
                  resultBuff = 0;

                  /* Trigger next conversion */
                  Cy_SAR2_Channel_SoftwareTrigger(ADC0_HW, 0);
            }
    }
}
```

2.4 A/D conversion ISR with software trigger

Figure 6 shows the example of ADC ISR with a software trigger. For details on CPU interrupt handling, see the architecture reference manual mentioned in References.

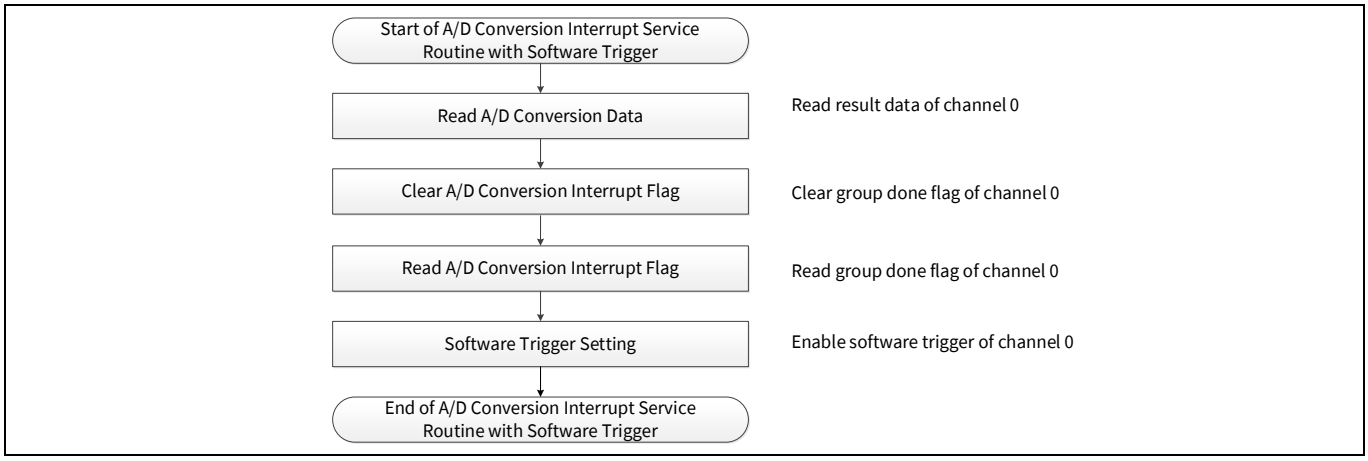

Figure 6 Example of ADC ISR with software trigger

2.4.1 Use case

See Section 2.3.1.

2.4.2 Configuration

Table 3 lists the functions of the configuration part of in the PDL for global ADC configuration.

Table 3 Functions for configuring the ADC ISR with software trigger

Functions	Description	Value
`Cy_SAR2_Channel_GetResult(SAR Channel,Status)`	Gets the conversion result and status	–
`Cy_SAR2_Channel_ClearInterruptStatus(SAR Channel,Source)`	Clears the corresponding channel interrupt status	–
`Cy_SAR2_Channel_SoftwareTrigger(SAR Channel)`	Issues the software start trigger	–

2.4.3 Sample code

See Code Listing 3 for sample code for initial configuration of the ADC ISR with software trigger.

Code Listing 3 Configuration of the ADC ISR with software trigger

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

    /* Initialize the device and board peripherals */
```

Software trigger procedure

Code Listing 3 Configuration of the ADC ISR with software trigger

```
    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: ISR with software trigger ADC conversion\r\n");
      printf("**************************************************************
******************\r\n");

    /* Initialize ADC */

    /* Initialize the SAR2 module */
    Cy_SAR2_Init(ADC0_HW, &ADC0_config);
    /* Set ePASS MMIO reference buffer mode for bangap voltage */
    Cy_SAR2_SetReferenceBufferMode(PASS0_EPASS_MMIO,
CY_SAR2_REF_BUF_MODE_OFF);

    /* Initialize the SAR2 Channel */
    Cy_SAR2_Channel_Init(ADC0_HW,0, &ADC0_channel_0_config);
```

Software trigger procedure

Code Listing 3 Configuration of the ADC ISR with software trigger

```
    /* Set ADC group done interrupt */
    Cy_SAR2_Channel_SetInterruptMask(ADC0_HW, ADC_LOGICAL_CHANNEL,
CY_SAR2_INT_GRP_DONE);

    /* Register ADC interrupt handler and enable interrupt */
       Cy_SysInt_Init(&irq_cfg_sar, &adc_int_handler);
       NVIC_ClearPendingIRQ(ADC_IRQ_NUM);
       NVIC_EnableIRQ((IRQn_Type)ADC_IRQ_NUM);



    /* Issue software start trigger */
    Cy_SAR2_Channel_SoftwareTrigger(ADC0_HW, 0);



    for (;;)
    {
       /* Check ADC conversion done flag */
              if(adc_done_flag != 0)
              {
                     adc_done_flag = 0;
                     /* Print out the ADC conversion result by UART */
                     printf("ADC conversion complete, result: %d\r\n",
adc_result);

              }
    }
}

void adc_int_handler (void)
{
       uint32_t adc_status = 0;

       /* Get interrupt status */
       uint32_t intrSource =
Cy_SAR2_Channel_GetInterruptStatusMasked(ADC0_HW, ADC_LOGICAL_CHANNEL);
       if(CY_SAR2_INT_GRP_DONE == (intrSource & CY_SAR2_INT_GRP_DONE))
       {
              /* Get the ADC conversion result and status */
```

Software trigger procedure

Code Listing 3 Configuration of the ADC ISR with software trigger

```
            adc_result = Cy_SAR2_Channel_GetResult(ADC0_HW,
ADC_LOGICAL_CHANNEL, &adc_status);
            if(CY_SAR2_STATUS_VALID == (adc_status &
CY_SAR2_STATUS_VALID))
            {
                  adc_done_flag = 1;
            }
            /* Clear interrupt source */
            Cy_SAR2_Channel_ClearInterrupt(ADC0_HW, ADC_LOGICAL_CHANNEL,
CY_SAR2_INT_GRP_DONE);


            /* Trigger next conversion */
            Cy_SAR2_Channel_SoftwareTrigger(ADC0_HW, 0);


      }
}
```

3 Hardware trigger procedure

The SAR ADC can be triggered by related hardware functions such as TCPWM, GPIO, and event generator.

This section explains the example application that converts voltage values given to the ADC[0]_6 pin of the MCU to digital values. The analog-to-digital conversion is repeated at regular intervals with a hardware trigger of a corresponding TCPWM.

The physical setting of this application is the same as shown in Figure 1.

Ensure that you have set up the parameters as mentioned in Basic ADC global .

To implement this application, follow the procedure for configuring the ADC channel and its example when using a hardware trigger. The example in this section uses a corresponding TCPWM for the trigger.

3.1 Configuring a logical channel with hardware trigger

Figure 7 shows the example of configuring a logical channel with a hardware trigger in XMC7000. In this example, the minimum value of the sample time is used. To find the proper sample time for your system, see the datasheet mentioned in References.

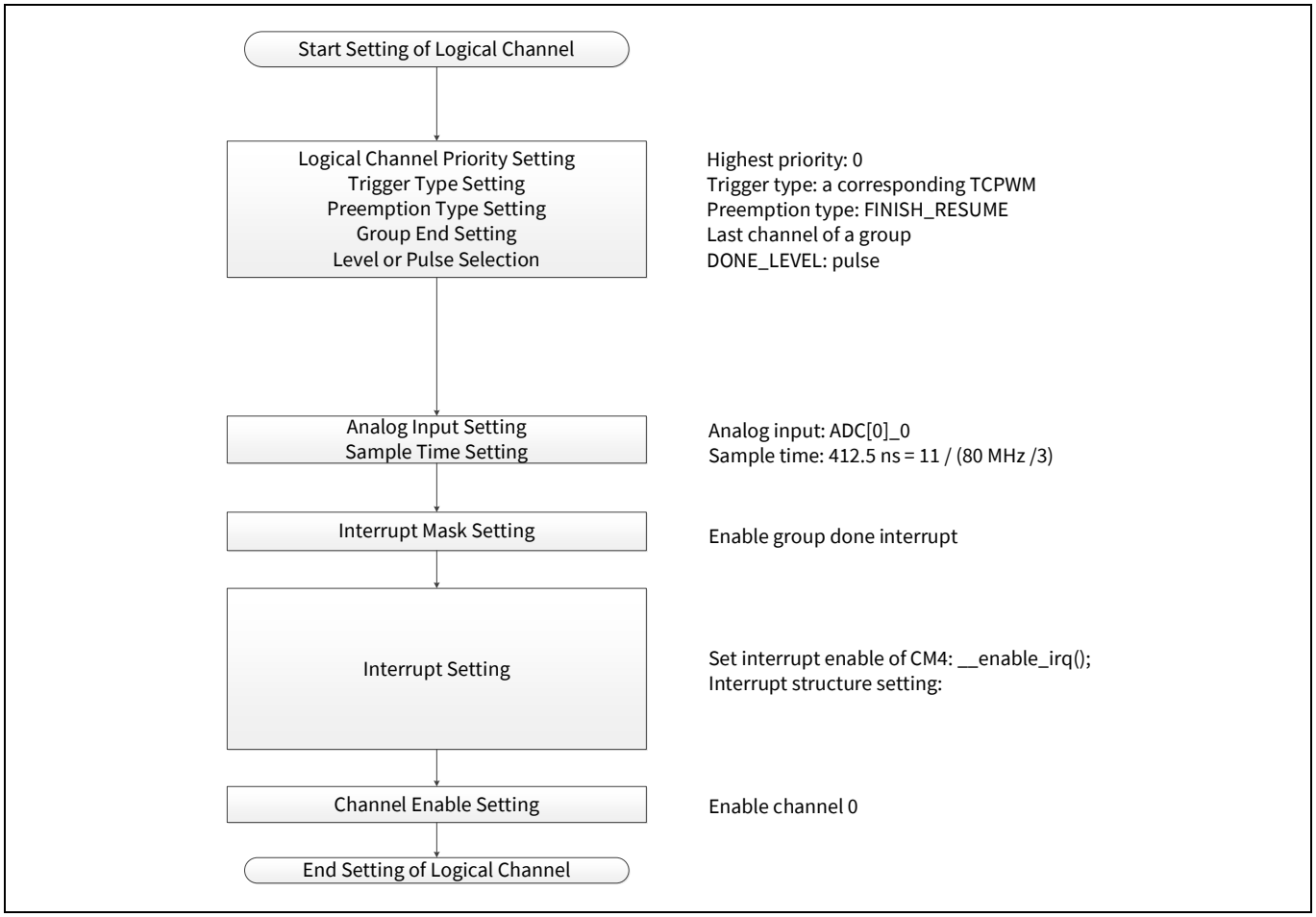

Figure 7 Example configuration of a logical channel with hardware trigger in XMC7000

Logical channel 0 is used in this example. Any logical channel will work in this application if proper analog input is selected. Also, TCPWM and the trigger multiplexer should be configured.

3.1.1 Use case

The following use case is an example of configuring a logical channel with hardware trigger:

- Analog input: ADC[0]_6
- Sample time: 846 ns=11/13 MHz
- Trigger select: TCPWM
- Pre-emption type: FINISH_RESUME
- GROUP_END: Last channel of a group
- External analog MUX: 0, Enable
- Pre-conditioning mode: OFF
- Overlap diagnostics mode: OFF
- Calibration value select: Regular
- Post processing mode: None
- Result alignment: Right
- Sign extension: Unsigned
- Averaging count: 0
- Shift right: 0
- Mask group: Done/Not cancelled/Not overflow
- Mask channel: Not range/Not pulse/Not overflow

Hardware trigger procedure

-

3.1.2 Configuration

Table 4 lists the functions of the configuration part in the PDL configuring a logical channel with hardware trigger in XMC7000.

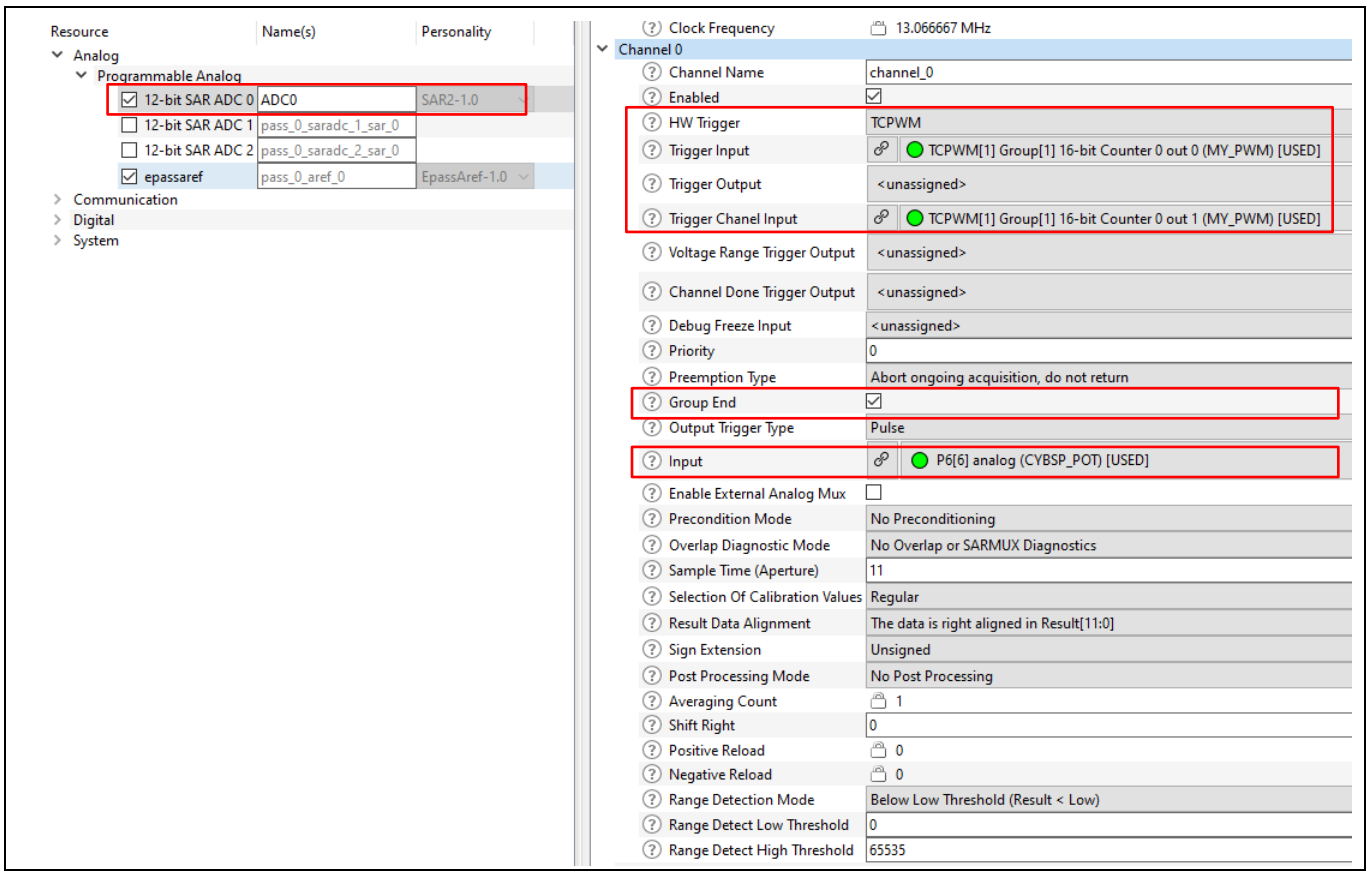

Figure 8 XMC7200: SAR channel configuration in Device Configurator

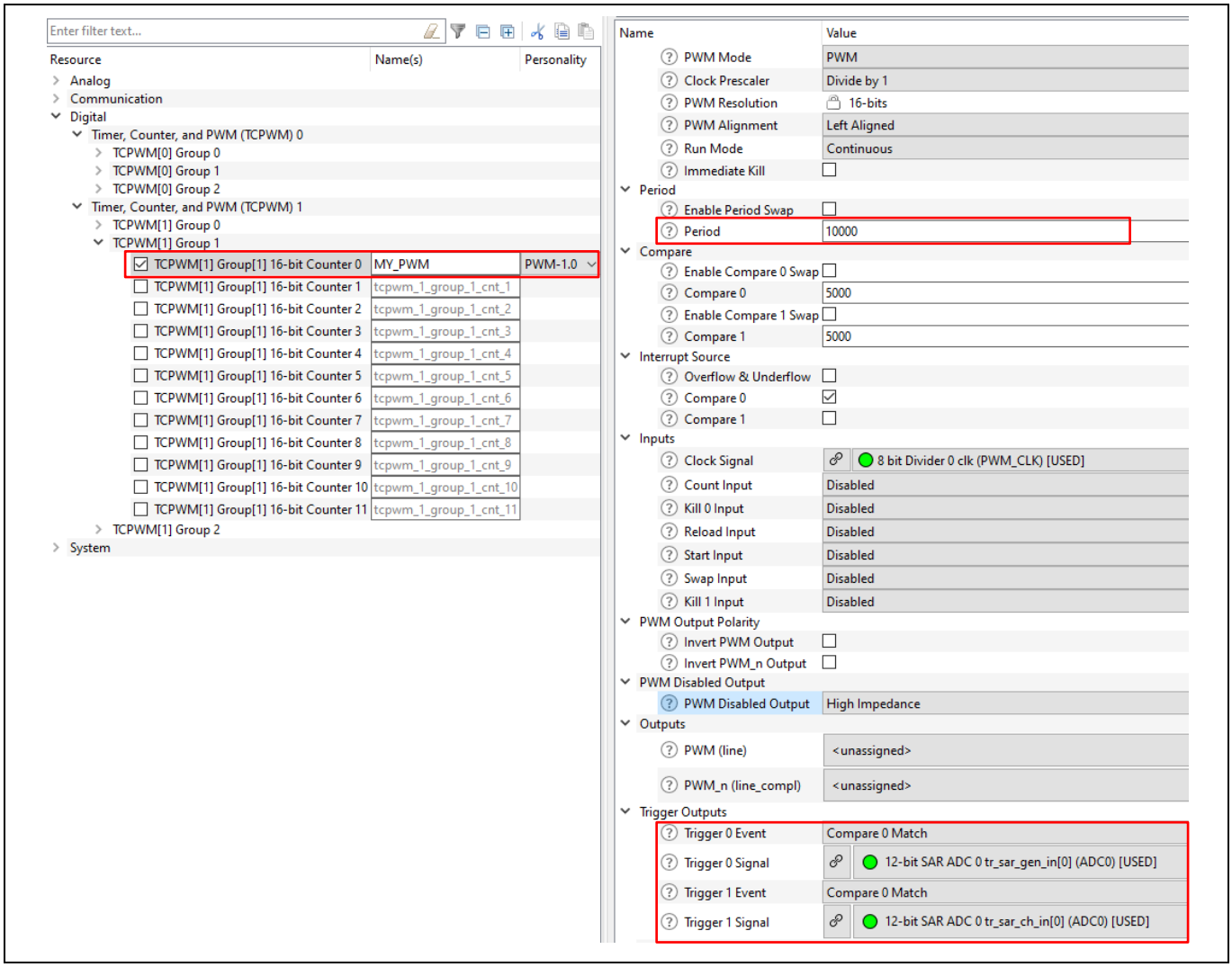

Figure 9 XMC7200: TCPWM configuration in Device Configurator

Table 4 Functions for configuring a logical channel with hardware trigger in XMC7000

Functions	Description	Value
`Cy_SAR2_Channel_GetInterruptMaskedStatus(PASS SARchannel,INTR Source)`	Returns the interrupt status	–
`Cy_Tcpwm_Pwm_Init(Group Counter, TCPWM PWM config)`	Initializes the TCPWM for PWM operation	–
`Cy_Tcpwm_Pwm_Enable(Group Counter)`	De-initializes the TCPWM block	–
`Cy_Tcpwm_TriggerStart(Group Counter)`	Triggers a software start on selected TCPWMs	–

3.1.3 Sample code

See Code Listing 4 and Code Listing 5 for sample code for initial configuration of a logical channel with hardware trigger in XMC7000.

Hardware trigger procedure

Code Listing 4 Configuring a logical channel with hardware trigger in XMC7000

```
#define MY_PWM_HW TCPWM1
#define MY_PWM_NUM 256UL
#define MY_PWM_IRQ tcpwm_1_interrupts_256_IRQn
#define MY_PWM_INPUT_DISABLED 0x7U

const cy_stc_tcpwm_pwm_config_t MY_PWM_config =
{
.pwmMode = CY_TCPWM_PWM_MODE_PWM,
.clockPrescaler = CY_TCPWM_PWM_PRESCALER_DIVBY_1,
.pwmAlignment = CY_TCPWM_PWM_LEFT_ALIGN,
.deadTimeClocks = 0,
.runMode = CY_TCPWM_PWM_CONTINUOUS,
.period0 = 10000,
.period1 = 32768,
.enablePeriodSwap = false,
.compare0 = 5000,
.compare1 = 16384,
.enableCompareSwap = false,
.interruptSources = (CY_TCPWM_INT_ON_TC & 0U) | (CY_TCPWM_INT_ON_CC0 ) |
(CY_TCPWM_INT_ON_CC1 & 0U),
.invertPWMOut = CY_TCPWM_PWM_INVERT_DISABLE,
.invertPWMOutN = CY_TCPWM_PWM_INVERT_DISABLE,
.killMode = CY_TCPWM_PWM_STOP_ON_KILL,
.swapInputMode = MY_PWM_INPUT_DISABLED & 0x3U,
.swapInput = CY_TCPWM_INPUT_0,
.reloadInputMode = MY_PWM_INPUT_DISABLED & 0x3U,
.reloadInput = CY_TCPWM_INPUT_0,
.startInputMode = MY_PWM_INPUT_DISABLED & 0x3U,
.startInput = CY_TCPWM_INPUT_0,
.killInputMode = MY_PWM_INPUT_DISABLED & 0x3U,
.killInput = CY_TCPWM_INPUT_0,
.countInputMode = MY_PWM_INPUT_DISABLED & 0x3U,
.countInput = CY_TCPWM_INPUT_1,
.swapOverflowUnderflow = false,
.immediateKill = false,
.tapsEnabled = 45,
.compare2 = 5000,
.compare3 = 16384,
```

Hardware trigger procedure

Code Listing 4 Configuring a logical channel with hardware trigger in XMC7000

```
.enableCompare1Swap = false,
.compare0MatchUp = true,
.compare0MatchDown = false,
.compare1MatchUp = true,
.compare1MatchDown = false,
.kill1InputMode = MY_PWM_INPUT_DISABLED & 0x3U,
.kill1Input = CY_TCPWM_INPUT_0,
.pwmOnDisable = CY_TCPWM_PWM_OUTPUT_HIGHZ,
.trigger0Event = CY_TCPWM_CNT_TRIGGER_ON_CC0_MATCH,
.trigger1Event = CY_TCPWM_CNT_TRIGGER_ON_CC0_MATCH,
};

#define ADC0_HW PASS0_SAR0
#define ADC0_CH0_HW PASS0_SAR0_CH0
#define ADC0_channel_0_HW ADC0_CH0_HW
#define ADC0_CH0_IRQ pass_0_interrupts_sar_0_IRQn
#define ADC0_channel_0_IRQ ADC0_CH0_IRQ
#define ADC0_VDDA_RANGE CY_SAR2_VDDA_2_7V_TO_4_5V
#ifndef SARMUX0_CH0_PORT_ADDR
#define SARMUX0_CH0_PORT_ADDR 0
#endif
#define ADC0_channel_0_IDX (0UL)

const cy_stc_sar2_channel_config_t ADC0_channel_0_config =
{
.channelHwEnable = true,
.triggerSelection = CY_SAR2_TRIGGER_TCPWM,
.channelPriority = 0U,
.preenptionType = CY_SAR2_PREEMPTION_ABORT_CANCEL,
.doneLevel = CY_SAR2_DONE_LEVEL_PULSE,
.isGroupEnd = true,
.pinAddress = SARMUX0_CH0_PIN_ADDR,
.portAddress = SARMUX0_CH0_PORT_ADDR,
.extMuxEnable = false,
.extMuxSelect = 0U,
.preconditionMode = CY_SAR2_PRECONDITION_MODE_OFF,
.overlapDiagMode = CY_SAR2_OVERLAP_DIAG_MODE_OFF,
.sampleTime = 11U,
```

Hardware trigger procedure

Code Listing 4 Configuring a logical channel with hardware trigger in XMC7000

```
.calibrationValueSelect = CY_SAR2_CALIBRATION_VALUE_REGULAR,
.postProcessingMode = CY_SAR2_POST_PROCESSING_MODE_NONE,
.resultAlignment = CY_SAR2_RESULT_ALIGNMENT_RIGHT,
.signExtention = CY_SAR2_SIGN_EXTENTION_UNSIGNED,
.averageCount = 1U,
.rightShift = 0U,
.positiveReload = 0U,
.negativeReload = 0U,
.rangeDetectionMode = CY_SAR2_RANGE_DETECTION_MODE_BELOW_LO,
.rangeDetectionLoThreshold = 0U,
.rangeDetectionHiThreshold = 65535U,
};
const cy_stc_sar2_config_t ADC0_config =
{
.preconditionTime = 0U,
.powerupTime = 0U,
.enableIdlePowerDown = false,
.msbStretchMode = CY_SAR2_MSB_STRETCH_MODE_1CYCLE,
.enableHalfLsbConv = false,
.sarMuxEnable = true,
.adcEnable = true,
.sarIpEnable = true,
.channelConfig = {(cy_stc_sar2_channel_config_t *)&ADC0_channel_0_config,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL},
};
```

Code Listing 5 TCPWM Hardware trigger ADC conversion function

```
int main(void)
{
    cy_rslt_t result;
    uint16_t               resultBuff;
      uint32_t adc_status = 0;

    /* Initialize the device and board peripherals */
```

Hardware trigger procedure

Code Listing 5 TCPWM Hardware trigger ADC conversion function

```
    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: HW trigger ADC conversion\r\n");
      printf("**************************************************************
******************\r\n");

    /* Initialize the SAR2 module */
    Cy_SAR2_Init(ADC0_HW, &ADC0_config);
    /* Set ePASS MMIO reference buffer mode for bangap voltage */
    Cy_SAR2_SetReferenceBufferMode(PASS0_EPASS_MMIO,
CY_SAR2_REF_BUF_MODE_OFF);

    /* Initialize the SAR2 Channel */
    Cy_SAR2_Channel_Init(ADC0_HW,0, &ADC0_channel_0_config);

    /* Initialize PWM using the config structure generated using device
configurator*/
```

Code Listing 5 TCPWM Hardware trigger ADC conversion function

```
    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);

    /* Then start the PWM */
    Cy_TCPWM_TriggerReloadOrIndex_Single(MY_PWM_HW, MY_PWM_NUM);

    for (;;)
    {
        /* Get conversion results in counts */
      resultBuff = Cy_SAR2_Channel_GetResult(ADC0_HW, 0, &adc_status);
      if(CY_SAR2_STATUS_VALID == (adc_status & CY_SAR2_STATUS_VALID))
            {
            /* Print out the ADC conversion result by UART */
                  printf("ADC conversion complete, result: %d\r\n",
resultBuff);
                  resultBuff = 0;
            }
    }
}
```

3.2 A/D conversion ISR with hardware trigger

Figure 10 shows the example of the ADC ISR with a hardware trigger. For details on CPU interrupt handing, see the architecture reference manual mentioned in References.

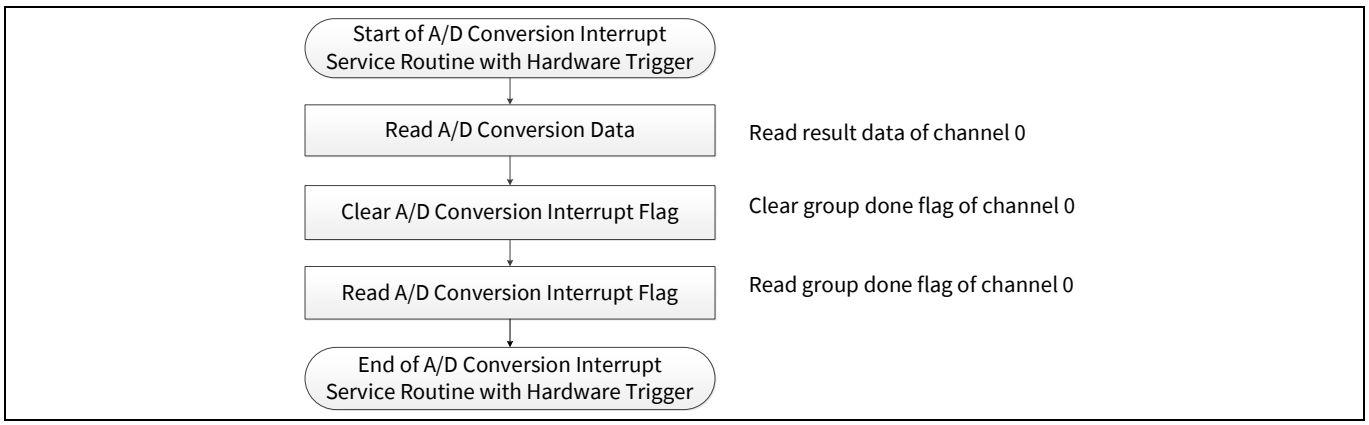

Figure 10 Example of ADC ISR with hardware trigger

3.2.1 Use case

See Section 3.1.1.

3.2.2 Configuration

Table 5 lists the functions of the configuration part of in the PDL for the ADC ISR with hardware trigger.

Table 5 Functions for configuring the ADC ISR with hardware trigger

Functions	Description	Value
`Cy_SAR2_Channel_GetResult(SAR Channel,Status)`	Gets the conversion result and status	–
`Cy_SAR2_Channel_ClearInterruptStatus(SAR Channel,Source)`	Clears the corresponding channel interrupt status	–

3.2.3 Sample code

See Code Listing 6 for sample code for the initial configuration of the ADC ISR with software trigger.

Code Listing 6 Configuring the ADC ISR with software trigger

```
/* ADC channel number definition */
#define ADC_LOGICAL_CHANNEL     (0u)
#define ADC_IRQ_NUM             (NvicMux2_IRQn)
#define ADC_INTR_NUM            (((ADC_IRQ_NUM << 16u) | ADC0_CH0_IRQ))
#define ADC_INTR_PRIORITY       (7u)

/*******************************************************************************
*****
* Global Variables
*******************************************************************************
****/
```

Hardware trigger procedure

Code Listing 6 Configuring the ADC ISR with software trigger

```
/* ADC interrupt configuration */
const cy_stc_sysint_t irq_cfg_sar =
{
    .intrSrc  = ADC_INTR_NUM,
    .intrPriority  = ADC_INTR_PRIORITY,
};

/* ADC conversion complete flag */
volatile uint8_t adc_done_flag = 0;

/* ADC conversion result */
uint16_t adc_result;


int main(void)
{
    cy_rslt_t result;

    /* 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);
      }
```

Code Listing 6 Configuring the ADC ISR with software trigger

```
       /* \x1b[2J\x1b[;H - ANSI ESC sequence for clear screen */
       printf("\x1b[2J\x1b[;H");
       printf("*************************************************************
******************\r\n");
       printf("XMC7000 MCU: HW trigger ADC conversion with interrupt\r\n");
       printf("*************************************************************
******************\r\n");


    /* Initialize the SAR2 module */
    Cy_SAR2_Init(ADC0_HW, &ADC0_config);
    /* Set ePASS MMIO reference buffer mode for bangap voltage */
    Cy_SAR2_SetReferenceBufferMode(PASS0_EPASS_MMIO,
CY_SAR2_REF_BUF_MODE_OFF);
    /* Initialize the SAR2 Channel */
    Cy_SAR2_Channel_Init(ADC0_HW,0, &ADC0_channel_0_config);


    /* Set ADC group done interrupt */
    Cy_SAR2_Channel_SetInterruptMask(ADC0_HW, ADC_LOGICAL_CHANNEL,
CY_SAR2_INT_GRP_DONE);


    /* Register ADC interrupt handler and enable interrupt */
       Cy_SysInt_Init(&irq_cfg_sar, &adc_int_handler);
       NVIC_ClearPendingIRQ(ADC_IRQ_NUM);
       NVIC_EnableIRQ((IRQn_Type)ADC_IRQ_NUM);


       /* Initialize TCPWM0_GRPx_CNTx_PWM_PR as PWM Mode & Enable */
       Cy_TCPWM_PWM_Init(PWM_SAR2_HW, PWM_SAR2_NUM,&PWM_SAR2_config);


       /* Enable the MY_COUNTER0 */
       Cy_TCPWM_PWM_Enable(PWM_SAR2_HW, PWM_SAR2_NUM);


       //Cy_TCPWM_TriggerReloadOrIndex_Single(PWM_SAR2_HW, PWM_SAR2_NUM);
       Cy_TCPWM_TriggerStart_Single(PWM_SAR2_HW, PWM_SAR2_NUM);


    for (;;)
    {
       /* Check ADC conversion done flag */
             if(adc_done_flag != 0)
             {
```

Hardware trigger procedure

Code Listing 6 Configuring the ADC ISR with software trigger

```
                    adc_done_flag = 0;
                    /* Print out the ADC conversion result by UART */
                    printf("ADC conversion complete, result: %d\r\n",
adc_result);
             }
    }
}

void adc_int_handler (void)
{
       uint32_t adc_status = 0;

       /* Get interrupt status */
       uint32_t intrSource =
Cy_SAR2_Channel_GetInterruptStatusMasked(ADC0_HW, ADC_LOGICAL_CHANNEL);
       if(CY_SAR2_INT_GRP_DONE == (intrSource & CY_SAR2_INT_GRP_DONE))
       {
             /* Get the ADC conversion result and status */
             adc_result = Cy_SAR2_Channel_GetResult(ADC0_HW,
ADC_LOGICAL_CHANNEL, &adc_status);
             if(CY_SAR2_STATUS_VALID == (adc_status &
CY_SAR2_STATUS_VALID))
             {
                    adc_done_flag = 1;
             }
             /* Clear interrupt source */
             Cy_SAR2_Channel_ClearInterrupt(ADC0_HW, ADC_LOGICAL_CHANNEL,
CY_SAR2_INT_GRP_DONE);
       }
}
```

4 Group procedure

The SAR ADC has a function for consecutive conversions using multiple pins with one trigger. The pins and the order of conversions can be selected from any ports that can be configured as an analog input. ADC logical channels that are selected for consecutive conversion with one trigger are called ‘groups’.

A group trigger is defined by the first channel of the group, which has the trigger type set to ‘OFF’ (no hardware trigger), ‘TCPWM’, ‘Generic 0 to 4’, or ‘Continuous’.

If the “continue group with next channel” option is not set by the GROUP_END bit of the SARn_CHxTR_CTL register, the group will consist of only one channel. Otherwise, the group continues until the last channel in a row with its bit set to ‘last channel of a group’. After the first channel of the group is triggered and converted, the second channel is automatically triggered, and so on until the whole group is converted.

Figure 11 shows an example application that converts the voltage consecutively in the order of ADC[0]_10, ADC[0]_12 and ADC[0]_15 with one software trigger.

Ensure that you have set up the parameters as mentioned in Basic ADC global .

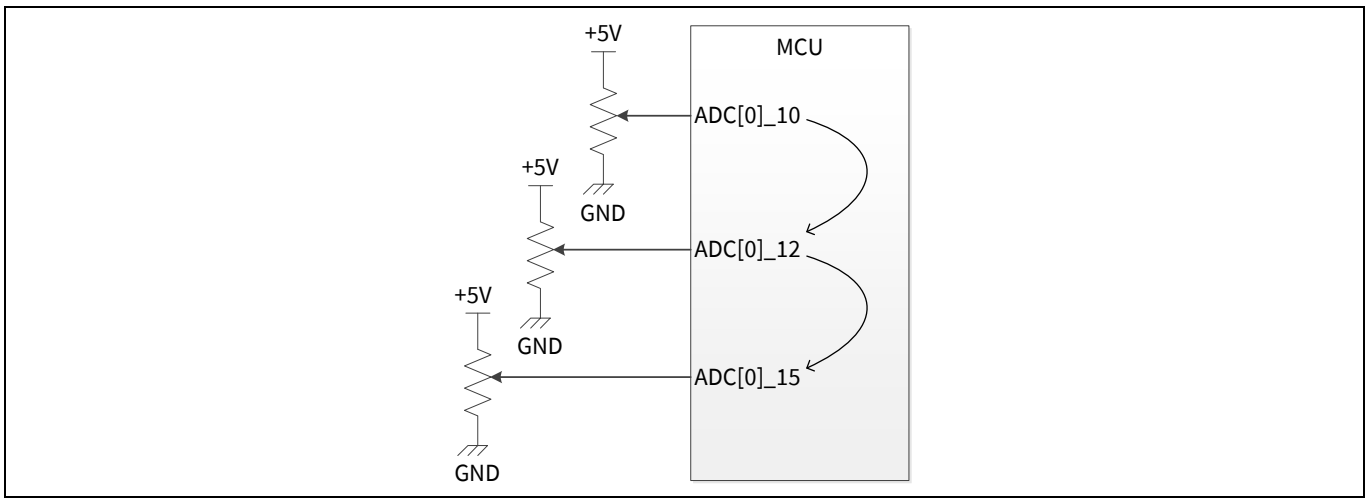

Figure 11 Example of group conversion connection

Use the information in the following sections and the example to implement this application.

4.1 Configuring a logical channel for group procedure

Figure 12 shows the example of configuring a logical channel for the group procedure in XMC7000. In this example, the minimum value of the sample time is used. To find the proper sample time for your system, see the datasheet mentioned in References.

Group procedure

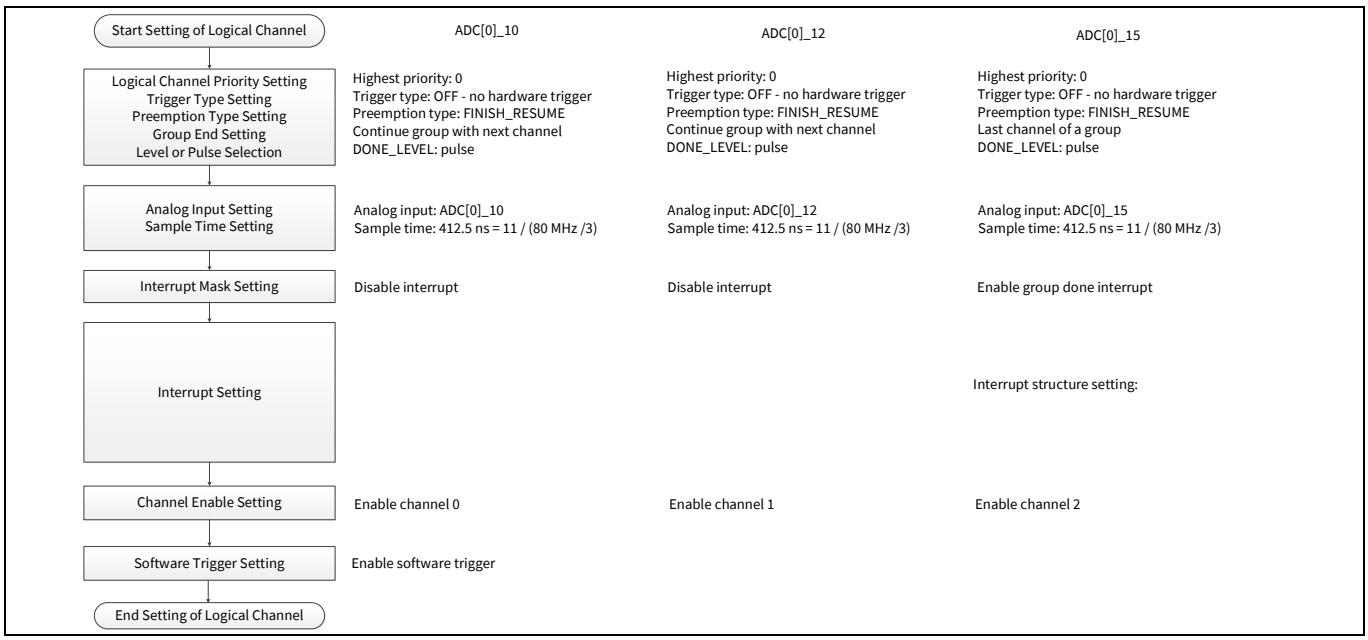

Figure 12 Example of configuring a logical channel for group conversion in XMC7000

Although this example uses logical channels 0, 1, and 2, you can use any channel if those numbers are consecutive.

4.1.1 Use case

The following use case is an example of configuring a logical channel for group conversion.

- ADC trigger selection: OFF
- Channel priority: 0 (highest)
- Channel pre-emption type: Finish resume
- Analog input, channel group and interrupt

Channel	Analog input	Channel group	Interrupt
CH0	ADC[0]_ 0	Continue group with next channel	Disable
CH1	ADC[0]_1	Continue group with next channel	Disable
CH2	ADC[0]_2	Last channel of a group	Enable group done interrupt

- ADC done level: Pulse
- External analog MUX: 0, Enable
- Pre-conditioning Mode: OFF
- Overlap diagnostics mode: OFF
- Calibration value select: Regular
- Post processing mode: None
- Result alignment: Right
- Sign extension: Unsigned
- Averaging count: 0
- Shift right: 0
- Mask group: Not done/Not cancelled/Not overflow

Group procedure

- Mask channel: Not range/Not pulse/Not overflow

4.1.2 Configuration

Table 6 lists the functions for configuring a logical channel for group conversion in XMC7000.

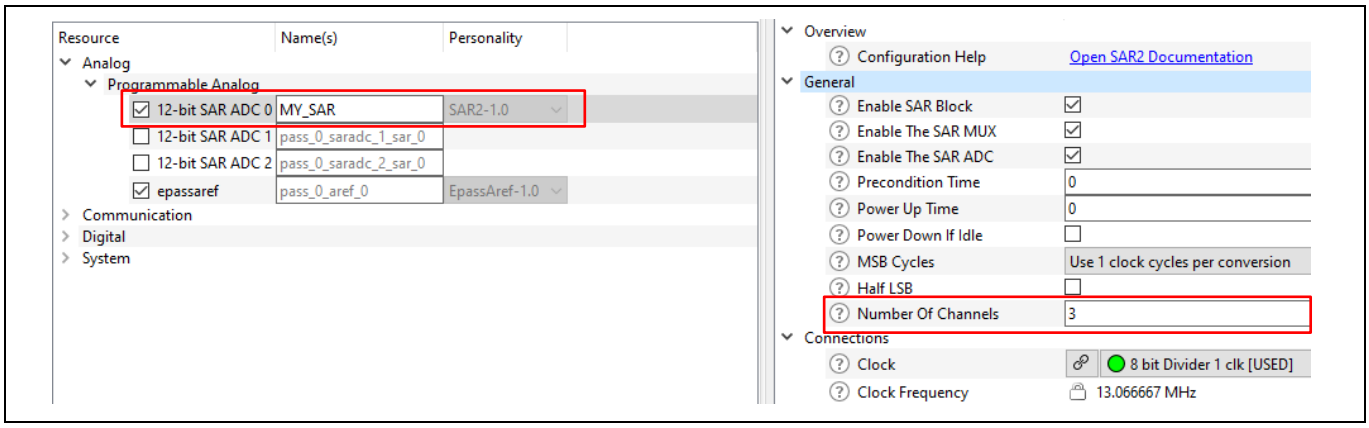

Figure 13 SAR2 global configuration for group in Device Configurator

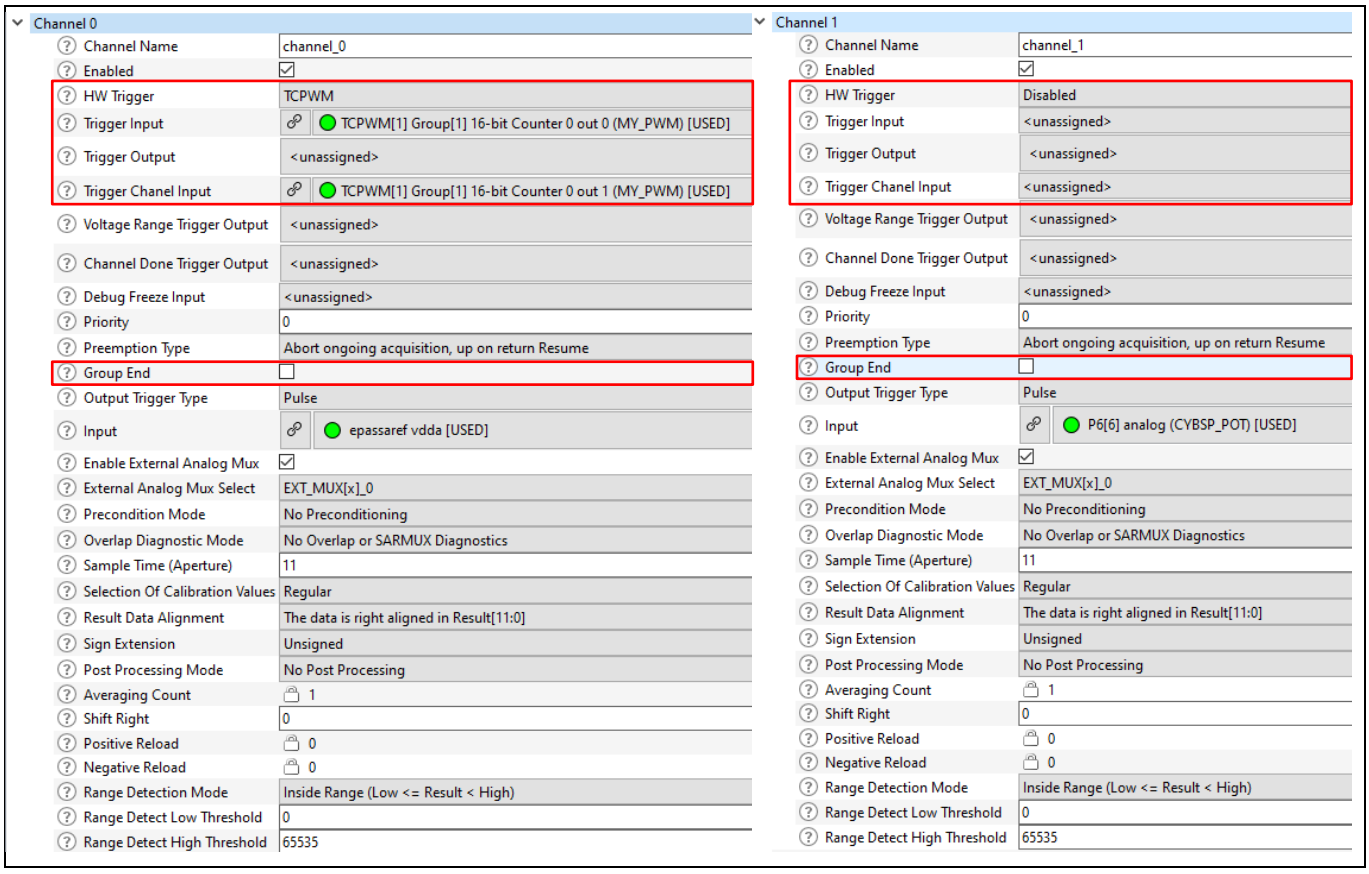

Figure 14 SAR2 channel_0 and channel_1 configuration for group in Device Configurator

Group procedure

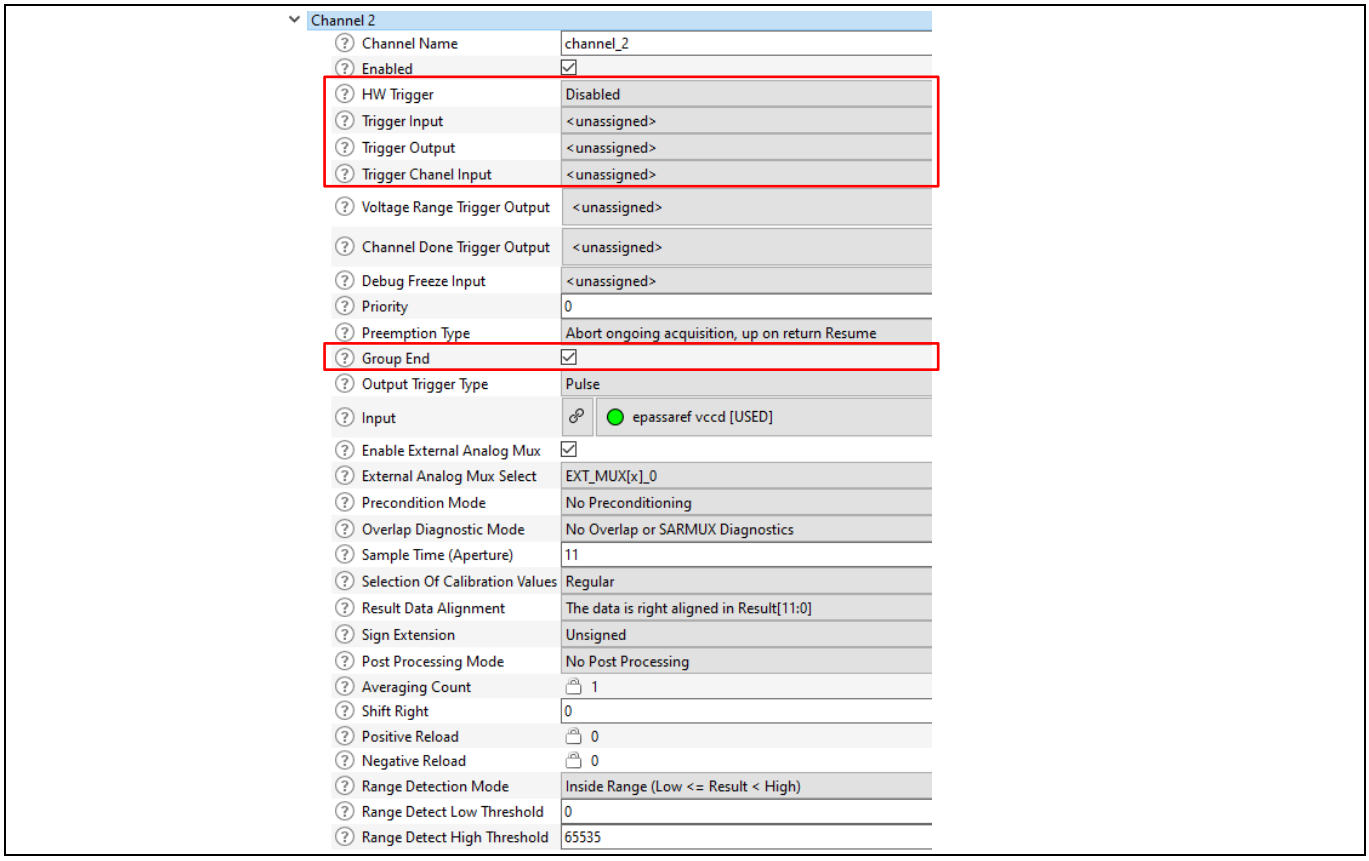

Figure 15 SAR2 channel_2 configuration for group in Device Configurator

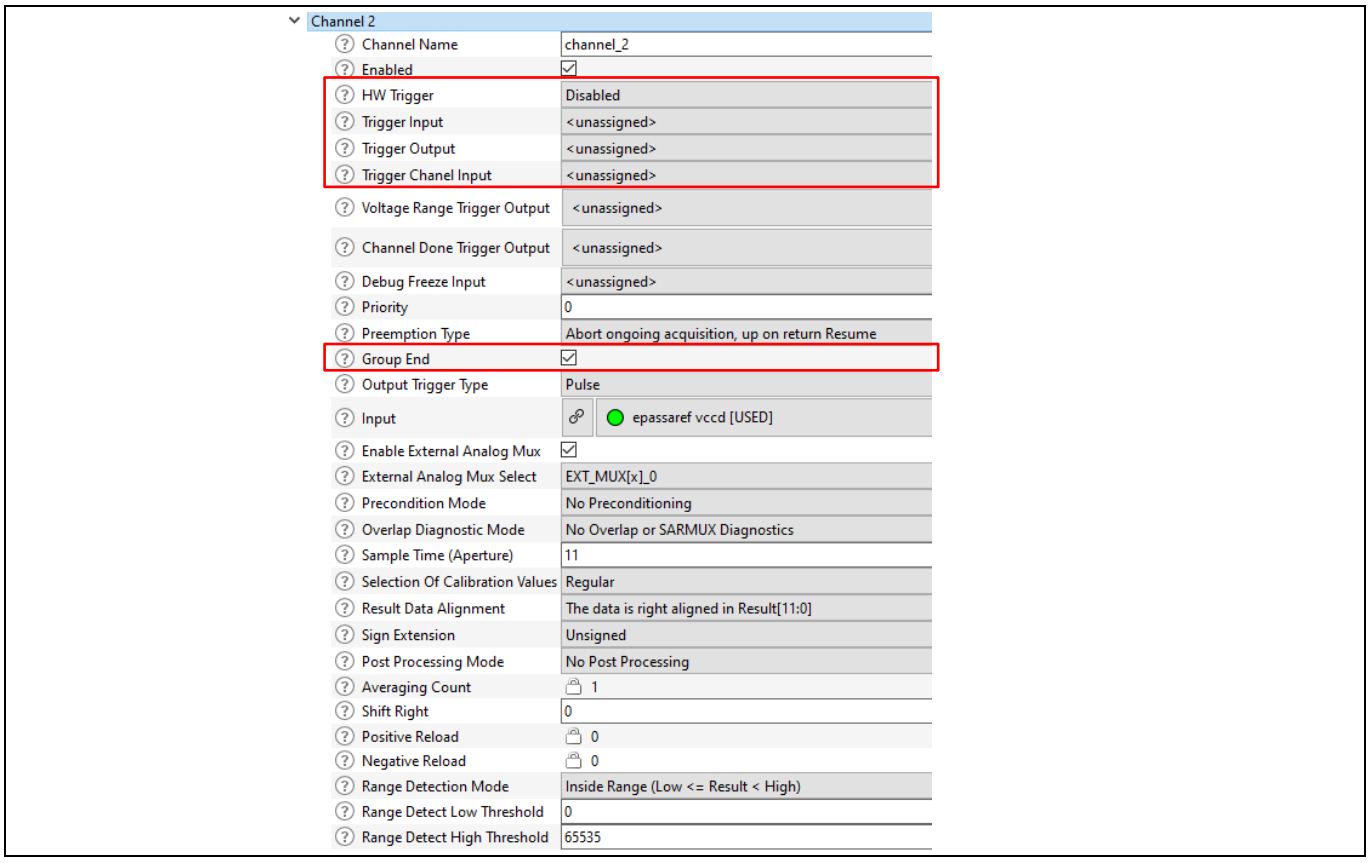

Figure 16 SAR2 channel_2 configuration for group in Device Configurator

Group procedure

Table 6 Functions for configuring logical channels for group in XMC7000

Functions	Description	Value
`Cy_SAR2_Channel_Init(PASS SARchannel,ADCchannel Configure)`	Initializes the ADC channel	–
`Cy_SysInt_InitIRQ(Config)`	Initializes the referenced system interrupt by setting the interrupt vector	–
`Cy_SAR2_Channel_Enable(SARchannel)`	Enables the corresponding channel	–
`Cy_SAR2_Channel_SoftwareTrigger(PASS SARchannel)`	Issues a software start trigger	–

4.1.3 Sample code

See Code Listing 7 for sample code for initial configuration of a logical channel for group procedure.

Code Listing 7 Configuring a logical channel for group procedure

```
/* SAR0 interrupt configuration structure */
const cy_stc_sysint_t SAR0_IRQ_cfg = {
    .intrSrc = ((NvicMux3_IRQn << 16) | MY_SAR_CH2_IRQ),
    .intrPriority = 7u
};

static bool sar0_isr_set = false;

int main(void)
{
    cy_rslt_t result;
    uint16_t sar0Result0 = 0u;
    uint16_t sar0Result1 = 0u;
    uint16_t sar0Result2 = 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();
```

Group procedure

Code Listing 7 Configuring a logical channel for group procedure

```
    /* 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 trigger mutil-channel ADC conversion
(Group)\r\n");
       printf("************************************************************
******************\r\n");

    /*Analog resource initialize*/
    init_analog_resources();

    /* 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);

    /* Then start the PWM */
    Cy_TCPWM_TriggerReloadOrIndex_Single(MY_PWM_HW, MY_PWM_NUM);

    for (;;)
    {
       if(sar0_isr_set)
          {
```

Group procedure

Code Listing 7 Configuring a logical channel for group procedure

```
            sar0Result0 = Cy_SAR2_Channel_GetResult(MY_SAR_HW, 0u, NULL);
            printf("sar0Result0 = %d \r\n",sar0Result0 );

            sar0Result1 = Cy_SAR2_Channel_GetResult(MY_SAR_HW, 1u, NULL);
            printf("sar0Result1 = %d \r\n",sar0Result1 );

            sar0Result2 = Cy_SAR2_Channel_GetResult(MY_SAR_HW, 2u, NULL);
                    printf("sar0Result2 = %d \r\n",sar0Result2 );

            sar0_isr_set = false;
        }
    }
}

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);
    }

    /*Configure and register SAR0 channel interrupts*/
    Cy_SysInt_Init(&SAR0_IRQ_cfg, sar0_interrupt_handler);
    NVIC_ClearPendingIRQ((IRQn_Type)SAR0_IRQ_cfg.intrSrc);
    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);

    /*Initialize SAR0 channel 0*/
    Cy_SAR2_Channel_Init(MY_SAR_HW, 0u, &MY_SAR_channel_0_config);
    Cy_SAR2_Channel_Init(MY_SAR_HW, 1u, &MY_SAR_channel_1_config);
    Cy_SAR2_Channel_Init(MY_SAR_HW, 2u, &MY_SAR_channel_2_config);

    /*Set SAR0 channel 0 interrupt mask*/
    Cy_SAR2_Channel_SetInterruptMask(MY_SAR_HW, 2u,CY_SAR2_INT_GRP_DONE);
```

Code Listing 7 Configuring a logical channel for group procedure

```
}
```

4.2 A/D conversion ISR for group

Figure 17 shows the example of an ADC ISR for the group. For details on CPU interrupt handling, see the architecture reference manual mentioned in the References.

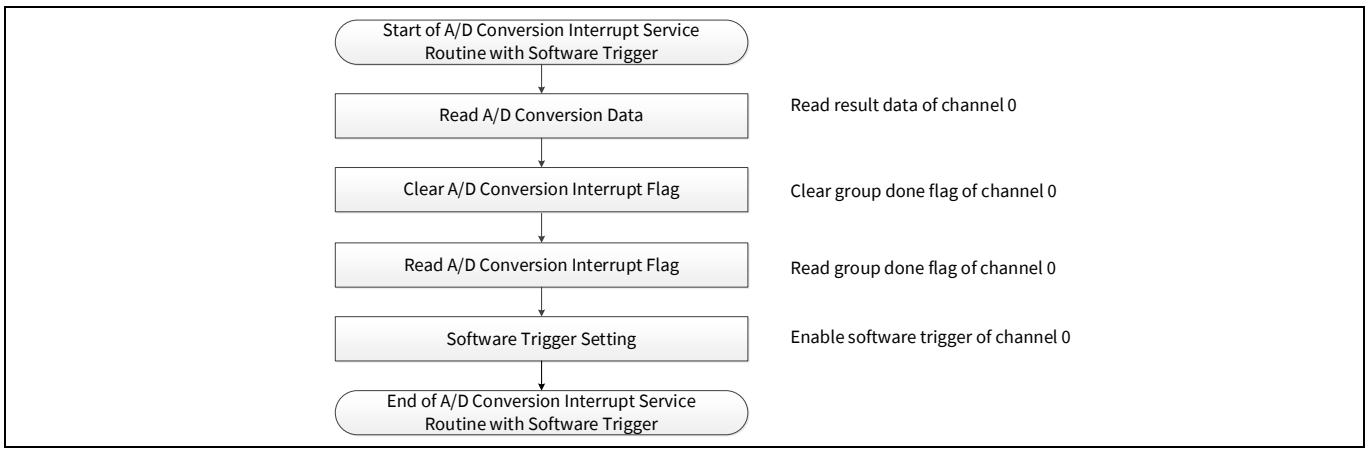

Figure 17 Example of ADC ISR for group

4.2.1 Use case

See Section 4.1.1.

4.2.2 Configuration

Table 7 lists the functions of the configuration part of in the PDL for an ADC ISR for group.

Table 7 Functions for configuring the ADC ISR for group procedure

Functions	Description	Value
`Cy_SAR2_Channel_GetResult(SAR Channel,Status)`	Gets the conversion result and status	–
`Cy_SAR2_Channel_ClearInterruptStatus(SAR Channel,Source)`	Clears the corresponding channel interrupt status	–
`Cy_SAR2_Channel_SoftwareTrigger(SAR Channel)`	Issues a software start trigger	–

4.2.3 Sample code

See Code Listing 8 for sample code for the initial configuration of ADC ISR for group procedure.

Code Listing 8 Configuring the ADC ISR for group procedure

```
static void sar0_interrupt_handler(void)
{
```

Group procedure

Code Listing 8 Configuring the ADC ISR for group procedure

```
    /* 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,2u) ==
CY_SAR2_INT_GRP_DONE)
    {
        sar0_isr_set = true;
    }

    /* Clear the interrupts */
    Cy_SAR2_Channel_ClearInterrupt(MY_SAR_HW, 2u ,CY_SAR2_INT_GRP_DONE);
}
```

5 Averaging procedure

Averaging is fully configured per channel by the SARn_CHx_POST_CTL register. The number of samples averaged is up to 256.

This section shows an example application that converts the voltage values given to the ADC[0]_0 at the MCU to digital averaging values. The physical setting of this application is the same as shown in Figure 1.

Ensure that you have configured the settings as described in the Basic ADC global configuration, Use case, and A/D conversion ISR with software trigger sections. Use the information in the following sections and the example to implement this application.

5.1 Configuring averaging

Figure 18 shows the example of configuring averaging. The ADC result of specified averaging count is accumulated, and it is shifted right by specifying the right shift bit. Then, it is stored in the result register. For true averaging, the averaging count must be a power of 2 and the right shift must be set to the corresponding value. For non-power of 2 the averaging counts, the right shift can only approximate the required divide. In this case, if a true averaging result is required, the software must perform a divide.

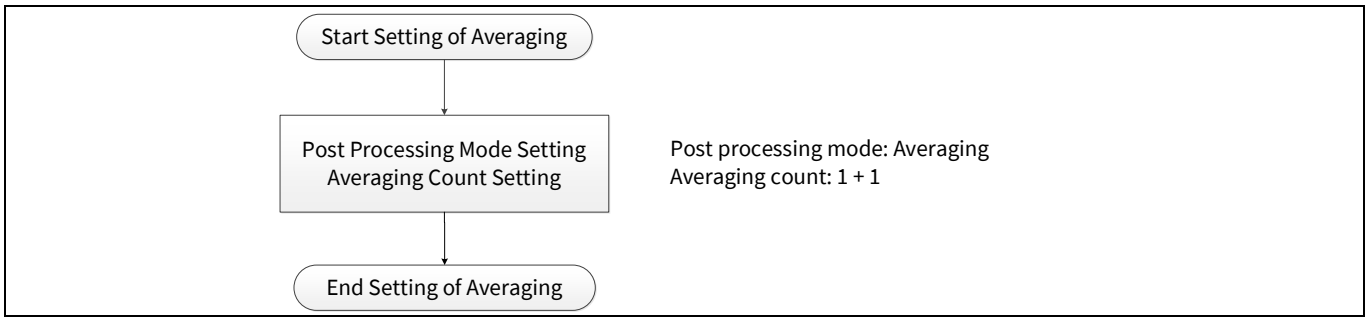

Figure 18 Example of configuring averaging

5.1.1 Use case

The following use case is an example of configuring averaging.

- Post processing mode: Averaging
- Averaging count: 1 + 1 times
- Right shift: 1
- Other use case items are same as Section 2.3.1.

5.1.2 Configuration

Figure 19 lists the parameters and Table 8 lists the functions of the configuration part in the PDL for configuring averaging.

Averaging procedure

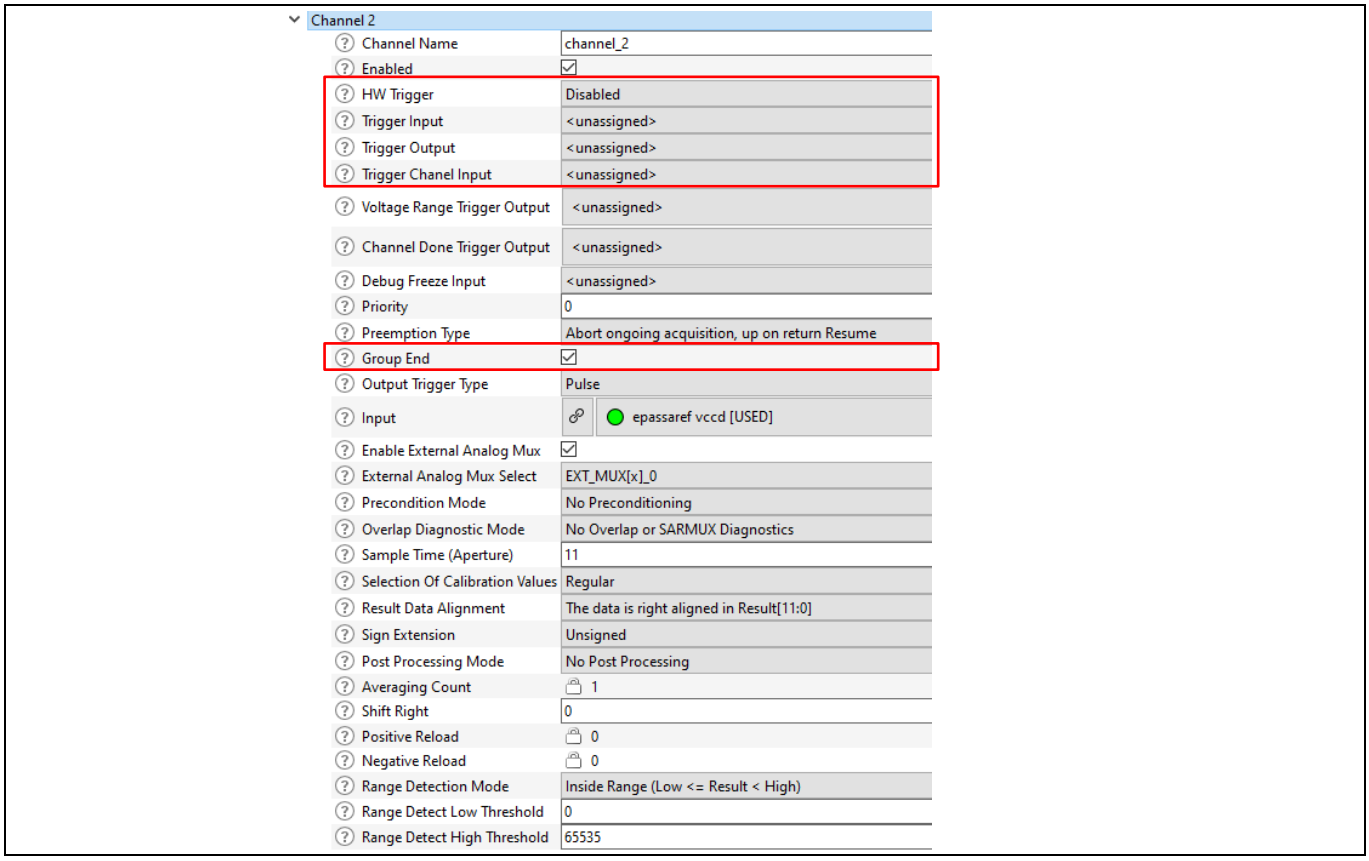

Figure 19 SAR2 channels configuration for averaging in Device Configurator

Table 8 Functions for configuring averaging

Functions	Description	Value
`Cy_SAR2_Channel_Init(PASS SARchannel,ADCchannel Configure)`	Initializes the ADC channel	–

5.1.3 Sample code

See Code Listing 9 for sample code for the initial configuration of averaging.

Code Listing 9 Configuring averaging

```
const cy_stc_sar2_channel_config_t MY_SAR_channel_0_config =
{
.channelHwEnable = true,
.triggerSelection = CY_SAR2_TRIGGER_TCPWM,
.channelPriority = 0U,
.preenptionType = CY_SAR2_PREEMPTION_ABORT_CANCEL,
.doneLevel = CY_SAR2_DONE_LEVEL_PULSE,
.isGroupEnd = true,
.pinAddress = SARMUX0_CH0_PIN_ADDR,
.portAddress = SARMUX0_CH0_PORT_ADDR,
```

Averaging procedure

Code Listing 9 Configuring averaging

```
.extMuxEnable = false,
.extMuxSelect = 0U,
.preconditionMode = CY_SAR2_PRECONDITION_MODE_OFF,
.overlapDiagMode = CY_SAR2_OVERLAP_DIAG_MODE_OFF,
.sampleTime = 20U,
.calibrationValueSelect = CY_SAR2_CALIBRATION_VALUE_REGULAR,
.postProcessingMode = CY_SAR2_POST_PROCESSING_MODE_AVG,
.resultAlignment = CY_SAR2_RESULT_ALIGNMENT_RIGHT,
.signExtention = CY_SAR2_SIGN_EXTENTION_UNSIGNED,
.averageCount = 8U,
.rightShift = 3U,
.positiveReload = 0U,
.negativeReload = 0U,
.rangeDetectionMode = CY_SAR2_RANGE_DETECTION_MODE_BELOW_LO,
.rangeDetectionLoThreshold = 0U,
.rangeDetectionHiThreshold = 65535U,
};

/* SAR0 interrupt configuration structure */
const cy_stc_sysint_t SAR0_IRQ_cfg = {
    .intrSrc = ((NvicMux3_IRQn << 16) | MY_SAR_CH0_IRQ),
    .intrPriority = 7u
};

static bool sar0_isr_set = false;

int main(void)
{
    cy_rslt_t result;
    uint16_t sar0Result0 = 0u;

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

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

Averaging procedure

Code Listing 9 Configuring averaging

```
    }

    /* 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: ADC conversion averaging process \r\n");
       printf("**************************************************************
******************\r\n");

    /*Analog resource initialize*/
    init_analog_resources();

    /* 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);

    /* Then start the PWM */
    Cy_TCPWM_TriggerReloadOrIndex_Single(MY_PWM_HW, MY_PWM_NUM);

    for (;;)
```

Averaging procedure

Code Listing 9 Configuring averaging

```
    {
       if(sar0_isr_set)
          {
             sar0Result0 = Cy_SAR2_Channel_GetResult(MY_SAR_HW, 0u, NULL);
             sar0_isr_set = false;
             printf("sar0Result0 = %d \r\n",sar0Result0 );
          }
    }
}

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,0u) ==
CY_SAR2_INT_GRP_DONE)
    {
        sar0_isr_set = true;
    }

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

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);
```

Averaging procedure

Code Listing 9 Configuring averaging

```
    /*Configure and register SAR0 channel interrupts*/
    Cy_SysInt_Init(&SAR0_IRQ_cfg, sar0_interrupt_handler);
    NVIC_ClearPendingIRQ((IRQn_Type)SAR0_IRQ_cfg.intrSrc);
    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);

    /*Initialize SAR0 channel 0*/
    status = Cy_SAR2_Channel_Init(MY_SAR_HW, 0u, &MY_SAR_channel_0_config);
    if (CY_SAR2_SUCCESS != status)
    {
        CY_ASSERT(0);
    }

    /*Set SAR0 channel 0 interrupt mask*/
    Cy_SAR2_Channel_SetInterruptMask(MY_SAR_HW, 0u,CY_SAR2_INT_GRP_DONE);

    /*Enable SAR0 channel 0*/
    Cy_SAR2_Channel_Enable(MY_SAR_HW, 0u);

}
```

6 Range detection procedure

Range detection enables a check with a high and low threshold register without CPU involvement. Each logical channel can be enabled for range detection individually. This function is usually used to monitor abnormal values in the voltage.

This section shows an example application that converts the voltage values given to the ADC[0]_0 of the MCU to digital values. If the digital value is greater than or equal to '2500' or less than '1500', it generates an interrupt to read the value as shown in Figure 20. Analog-to-digital conversion is repeated at regular timing with the hardware trigger of a corresponding TCPWM. The physical setting of this application is the same as shown in Figure 1.

Ensure that you have configured the settings as described in the Basic ADC global and Configuring a logical channel with hardware trigger sections.

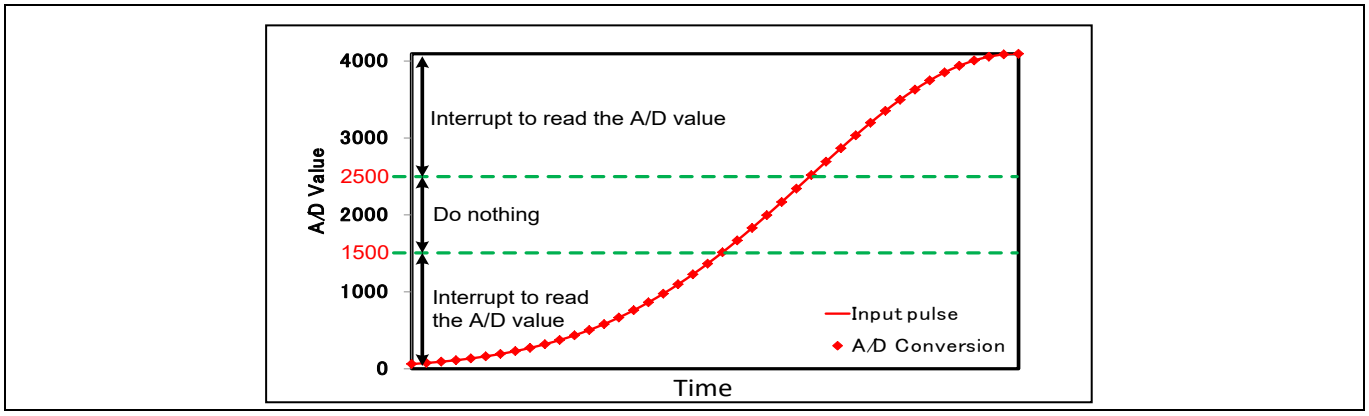

Figure 20 Example behavior of range detection application

Use the information in the following sections and the example to implement this application.

6.1 Configuring range detection

Figure 21 shows the example of configuring range detection.

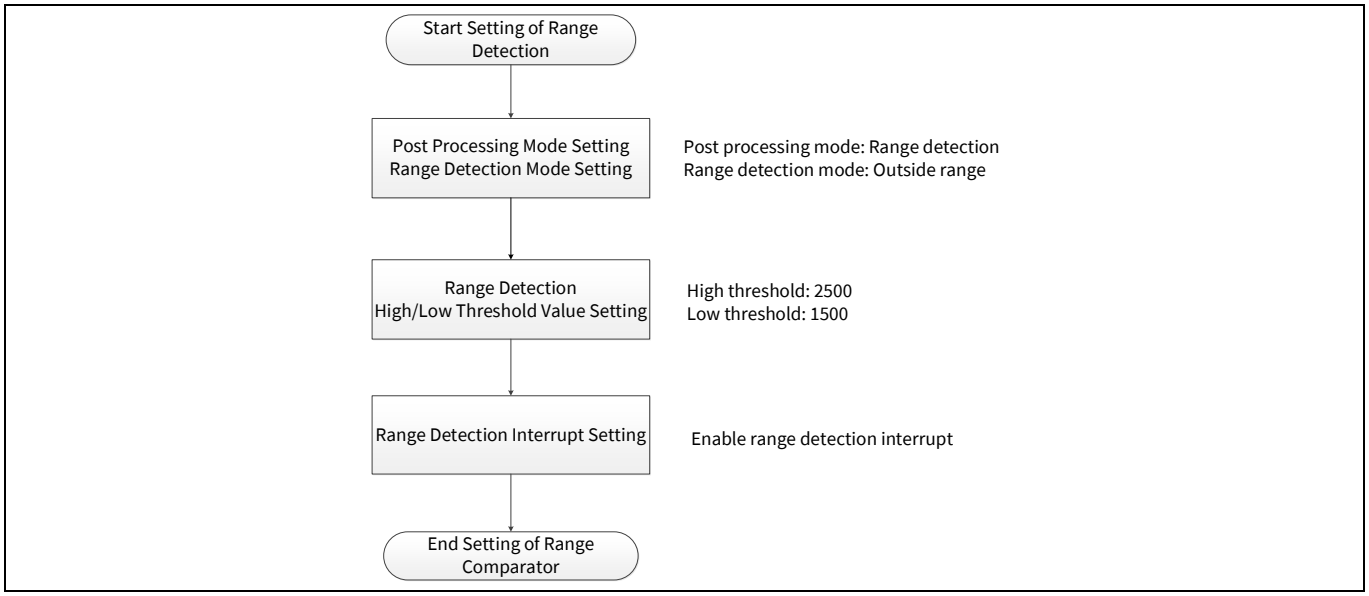

Figure 21 Example of configuring range detection

6.1.1 Use case

The following use case is an example of configuring range detection.

- Range detection threshold: Low: 1500 / High: 2500
- Other use case items are same as Section 2.2.1 and Section 3.1.1.

6.1.2 Configuration

Table 9 lists the functions of the configuration part of in the PDL for range detection.

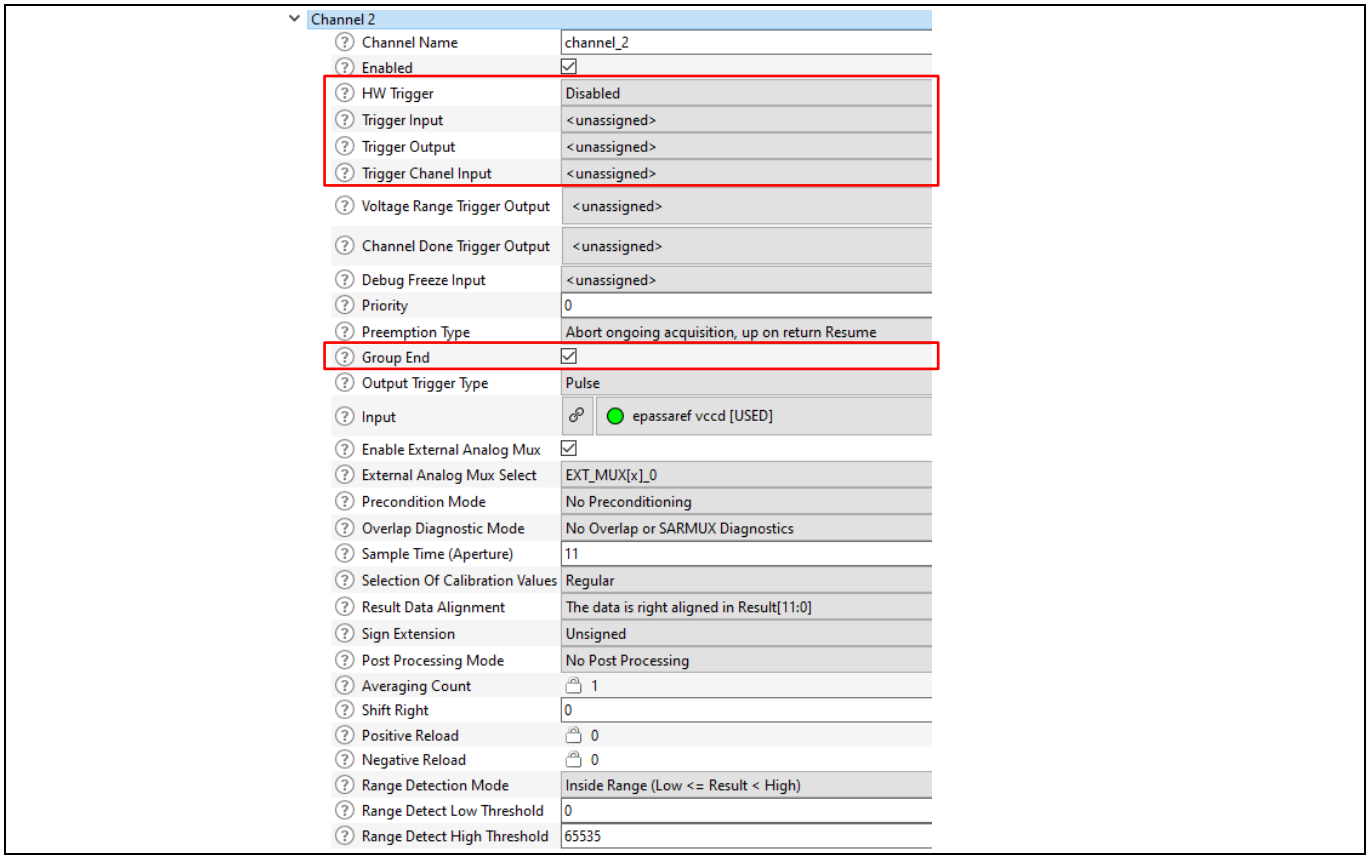

Figure 22 SAR2 channels configuration for range detection in Device Configurator

Table 9 Functions for configuring range detection

Functions	Description	Value
`Cy_SAR2_Channel_Init(PASS SARchannel,ADCchannel Configure)`	Initializes the ADC channel	–
`Cy_SysInt_InitIRQ(Config)`	Initializes the referenced system interrupt by setting the interrupt vector	–
`Cy_SAR2_Channel_Enable(PAS S SARchannel)`	Enables the corresponding channel	–
`Cy_SAR2_Channel_SoftwareTr igger(PASS SARchannel)`	Issues a software start trigger	–

6.1.3 Sample code

See Code Listing 10 for sample code for the initial configuration of configuring range detection.

Code Listing 10 Configuring range detection

```
const cy_stc_sar2_channel_config_t MY_SAR_channel_0_config =
{
.channelHwEnable = true,
.triggerSelection = CY_SAR2_TRIGGER_TCPWM,
.channelPriority = 0U,
.preenptionType = CY_SAR2_PREEMPTION_ABORT_RESUME,
.doneLevel = CY_SAR2_DONE_LEVEL_PULSE,
.isGroupEnd = true,
.pinAddress = SARMUX0_CH0_PIN_ADDR,
.portAddress = SARMUX0_CH0_PORT_ADDR,
.extMuxEnable = false,
.extMuxSelect = 0U,
.preconditionMode = CY_SAR2_PRECONDITION_MODE_OFF,
.overlapDiagMode = CY_SAR2_OVERLAP_DIAG_MODE_OFF,
.sampleTime = 11U,
.calibrationValueSelect = CY_SAR2_CALIBRATION_VALUE_REGULAR,
.postProcessingMode = CY_SAR2_POST_PROCESSING_MODE_RANGE,
.resultAlignment = CY_SAR2_RESULT_ALIGNMENT_RIGHT,
.signExtention = CY_SAR2_SIGN_EXTENTION_UNSIGNED,
.averageCount = 1U,
.rightShift = 0U,
.positiveReload = 0U,
.negativeReload = 0U,
.rangeDetectionMode = CY_SAR2_RANGE_DETECTION_MODE_OUTSIDE_RANGE,
.rangeDetectionLoThreshold = 1500U,
.rangeDetectionHiThreshold = 2500U,
};

/* SAR0 interrupt configuration structure */
const cy_stc_sysint_t SAR0_IRQ_cfg = {
    .intrSrc = ((NvicMux3_IRQn << 16) | MY_SAR_CH0_IRQ),
    .intrPriority = 7u
};

static bool sar0_isr_set = false;
```

Range detection procedure

Code Listing 10 Configuring range detection

```
int main(void)
{
    cy_rslt_t result;
    uint16_t sar0Result0 = 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: ADC conversion Range Detection\r\n");
      printf("**************************************************************
******************\r\n");

    /*Analog resource initialize*/
    init_analog_resources();

    /* Initialize PWM using the config structure generated using device
configurator*/
```

Range detection procedure

Code Listing 10 Configuring range detection

```
    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);

    /* Then start the PWM */
    Cy_TCPWM_TriggerReloadOrIndex_Single(MY_PWM_HW, MY_PWM_NUM);

    for (;;)
    {
      if(sar0_isr_set)
         {
            sar0Result0 = Cy_SAR2_Channel_GetResult(MY_SAR_HW, 0u, NULL);
            sar0_isr_set = false;
            printf("sar0Result0 = %d \r\n",sar0Result0 );
         }
    }
}

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);
```

Code Listing 10 Configuring range detection

```
    /*Initialize SAR0 channel 0*/
    status = Cy_SAR2_Channel_Init(MY_SAR_HW, 0u, &MY_SAR_channel_0_config);
    if (CY_SAR2_SUCCESS != status)
    {
        CY_ASSERT(0);
    }

    /*Set SAR0 channel 0 interrupt mask*/
    Cy_SAR2_Channel_SetInterruptMask(MY_SAR_HW, 0u,CY_SAR2_INT_CH_RANGE);

    /*Configure and register SAR0 channel interrupts*/
    Cy_SysInt_Init(&SAR0_IRQ_cfg, sar0_interrupt_handler);
    NVIC_ClearPendingIRQ((IRQn_Type)SAR0_IRQ_cfg.intrSrc);
    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);
}
```

6.2 A/D conversion ISR for range detection

Figure 23 shows the example of ADC ISR for a range detection. For details on CPU interrupt handing, see the architecture reference manual mentioned in References.

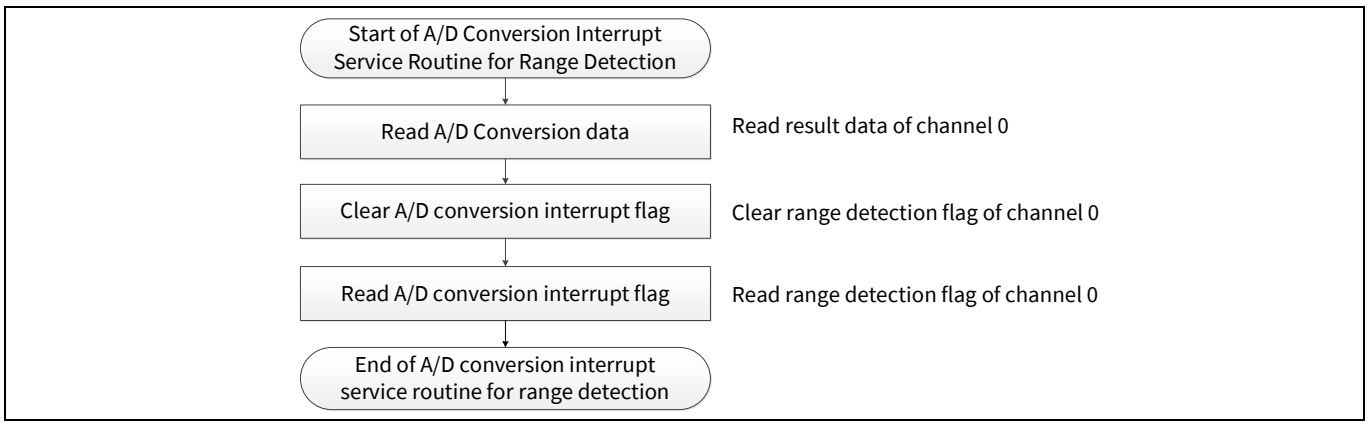

Figure 23 Example of ADC ISR for range detection

6.2.1 Configuration

Table 10 lists the functions of the configuration part of in the PDL for configuring the ADC ISR for range detection.

Range detection procedure

Table 10 Functions for configuring the ADC ISR for range detection

Functions	Description	Value
`Cy_SAR2_Channel_GetInterruptMaskedStatus(PASS SARchannel,INTR Source)`	Returns the interrupt status	–
`Cy_SAR2_Channel_GetResult(SAR Channel,Status)`	Gets the conversion result and status	–
`Cy_SAR2_Channel_ClearInterruptStatus(SAR Channel,Source)`	Clears the corresponding channel interrupt status	–

6.2.2 Sample code

See Code Listing 11 for sample code for the initial configuration of the ADC ISR for range detection.

Code Listing 11 Configuring the ADC ISR for range detection

```
static void sar0_interrupt_handler(void)
{
	/* Get interrupt status */
	uint32_t intrSource =
Cy_SAR2_Channel_GetInterruptStatusMasked(MY_SAR_HW, 0);
	if(CY_SAR2_INT_CH_RANGE == (intrSource & CY_SAR2_INT_CH_RANGE))
	{
		sar0_isr_set = true;
		/* Clear interrupt source */
		Cy_SAR2_Channel_ClearInterrupt(MY_SAR_HW, 0,
CY_SAR2_INT_CH_RANGE);
	}
}
```

7 Pulse detection procedure

The result of comparison from range detection can be filtered with the pulse detection function. For every logical channel, the pulse detection function has a pair of reload registers to store the initial value for the positive and negative down counters. The positive and negative counters decrement on positive and negative events obtained from the result of the comparison done by range detection. This function is usually used to avoid misdetections of abnormal voltage caused by noise and so on when monitoring the voltage.

For parameters described in the Configuring range detection, an analog-to-digital conversion result greater than or equal to 2500 or less than 1500 is a positive event; one greater than or equal to 1500 and less than 2500 is a negative event. This example application generates an interrupt if the positive event has occurred five times in a row as shown in Figure 24.

The continuous number of positive events is cancelled when two negative events occur in a row; that is, the count of positive event continues even if a negative event has occurred just once as shown in Figure 25.

The physical setting of this application is the same as shown in Figure 1.

Ensure that you have configured the settings described in the Basic ADC global configuration and Configuring a logical channel with hardware trigger sections.

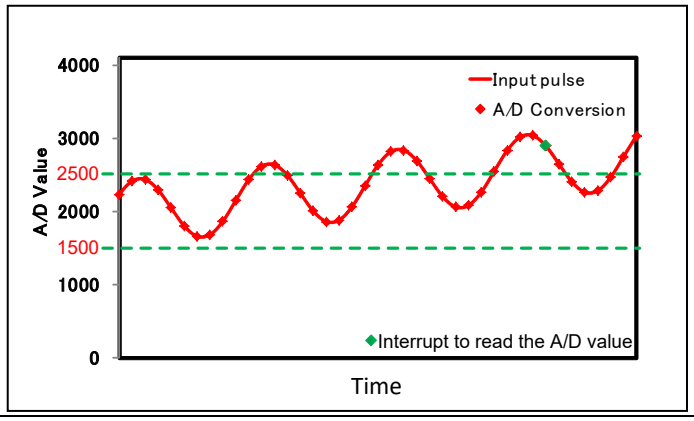

Figure 24 Example behavior of pulse detection application 1

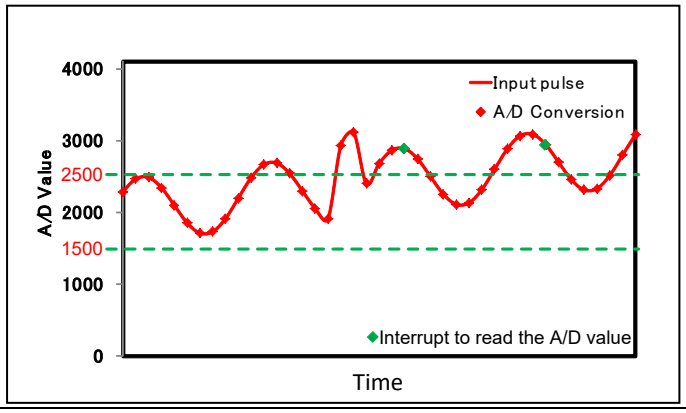

Figure 25 Example behavior of pulse detection application 2

Use the information in the following sections and the example to implement this application.

7.1 Pulse detection settings

Figure 26 shows an example of pulse detection setting.

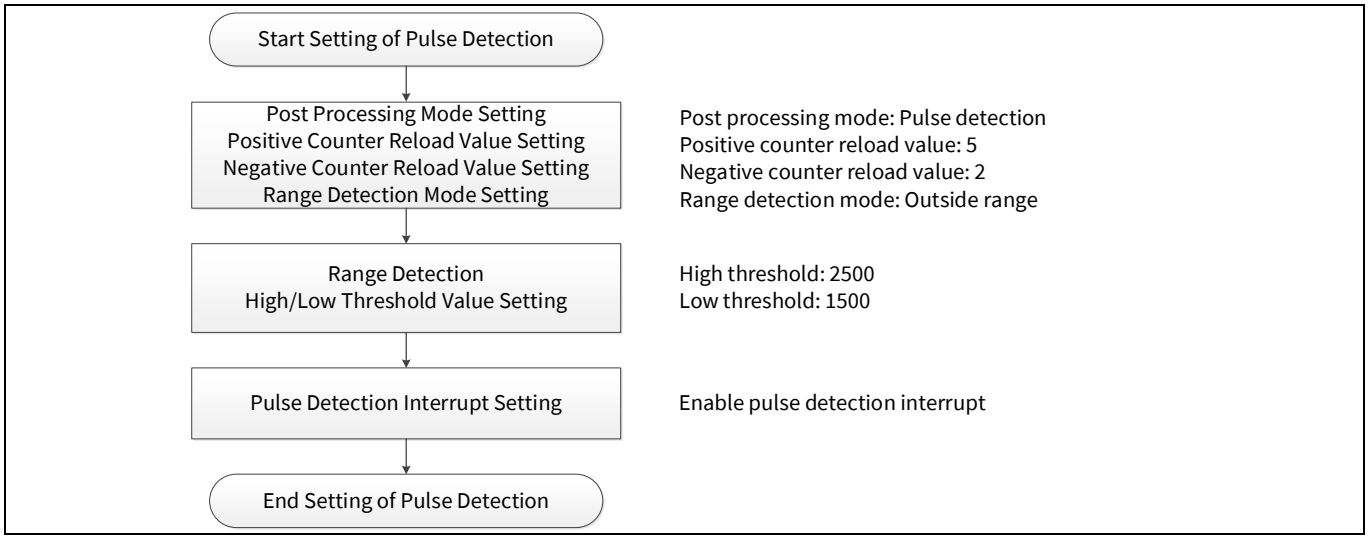

Figure 26 Example of configuring pulse detection

7.1.1 Use case

The following use case is an example of configuring pulse detection.

- Analog input: ADC[0]_0
- Range detection threshold: Low: 1500 / High: 2500
- ADC trigger: TCPWM
- Positive counter reload value: 5
- Negative counter reload value: 2

Other use case items are same as Section 2.2.1 and Section 3.1.1.

7.1.2 Configuration

Table 11 lists the functions of the configuration part in the PDL for pulse detection.

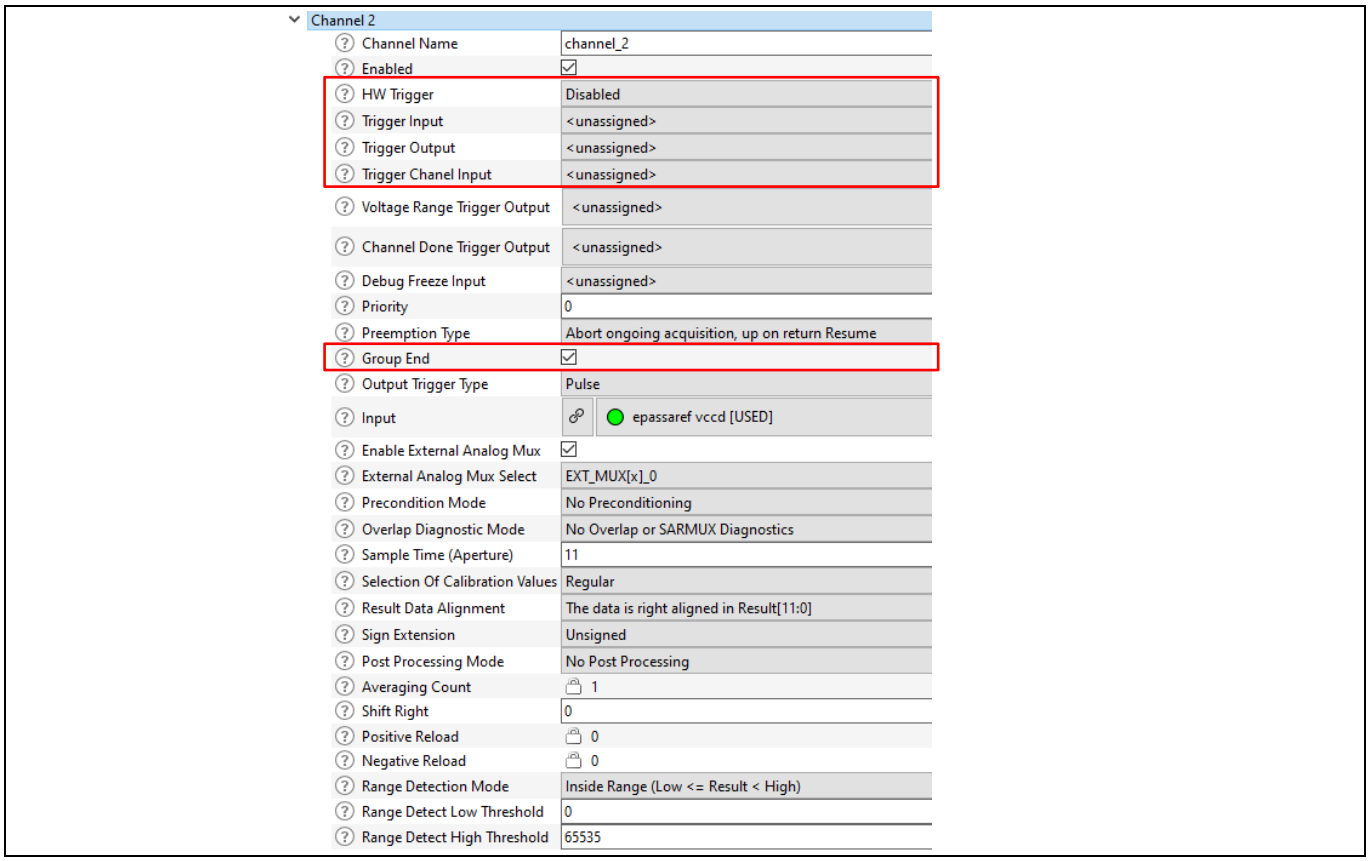

Figure 27 SAR2 channels configuration for pulse detection in Device Configurator

Table 11 Functions for configuring pulse detection

Functions	Description	Value
`Cy_SAR2_Channel_Init(PASS SARchannel,ADCchannel Configure)`	Initializes the ADC channel	–
`Cy_SysInt_InitIRQ(Config)`	Initializes the referenced system interrupt by setting the interrupt vector	–
`Cy_SAR2_Channel_Enable(PASS SARchannel)`	Enables the corresponding channel	–
`Cy_Tcpwm_Pwm_Enable(Group Counter)`	De-initializes the TCPWM block	–
`Cy_Tcpwm_TriggerStart(Group Counter)`	Triggers a software start on the selected TCPWMs	–

7.1.3 Sample code

See Code Listing 12.

Code Listing 12 Configuring pulse detection

```
/* SAR0 interrupt configuration structure */
const cy_stc_sysint_t SAR0_IRQ_cfg = {
    .intrSrc = ((NvicMux3_IRQn << 16) | MY_SAR_CH0_IRQ),
    .intrPriority = 7u
};

static bool sar0_isr_set = false;
const cy_stc_sar2_channel_config_t MY_SAR_channel_0_config =
{
.channelHwEnable = true,
.triggerSelection = CY_SAR2_TRIGGER_TCPWM,
.channelPriority = 0U,
.preenptionType = CY_SAR2_PREEMPTION_ABORT_RESUME,
.doneLevel = CY_SAR2_DONE_LEVEL_PULSE,
.isGroupEnd = true,
.pinAddress = SARMUX0_CH0_PIN_ADDR,
.portAddress = SARMUX0_CH0_PORT_ADDR,
.extMuxEnable = false,
.extMuxSelect = 0U,
.preconditionMode = CY_SAR2_PRECONDITION_MODE_OFF,
.overlapDiagMode = CY_SAR2_OVERLAP_DIAG_MODE_OFF,
.sampleTime = 11U,
.calibrationValueSelect = CY_SAR2_CALIBRATION_VALUE_REGULAR,
.postProcessingMode = CY_SAR2_POST_PROCESSING_MODE_RANGE_PULSE,
.resultAlignment = CY_SAR2_RESULT_ALIGNMENT_RIGHT,
.signExtention = CY_SAR2_SIGN_EXTENTION_UNSIGNED,
.averageCount = 1U,
.rightShift = 0U,
.positiveReload = 5U,
.negativeReload = 2U,
.rangeDetectionMode = CY_SAR2_RANGE_DETECTION_MODE_OUTSIDE_RANGE,
.rangeDetectionLoThreshold = 1500U,
.rangeDetectionHiThreshold = 2500U,
```

Pulse detection procedure

Code Listing 12 Configuring pulse detection

```
};

int main(void)
{
    cy_rslt_t result;
    uint16_t sar0Result0 = 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: ADC conversion Pulse Detection\r\n");
      printf("**************************************************************
******************\r\n");

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

Pulse detection procedure

Code Listing 12 Configuring pulse detection

```
    /* 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);

    /* Then start the PWM */
    Cy_TCPWM_TriggerReloadOrIndex_Single(MY_PWM_HW, MY_PWM_NUM);

    for (;;)
    {
       if(sar0_isr_set)
          {
                sar0Result0 = Cy_SAR2_Channel_GetResult(MY_SAR_HW, 0u,
NULL);
                sar0_isr_set = false;
                printf("sar0Result0 = %d \r\n",sar0Result0 );
          }
    }
}

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);
    }
    /* Set ePASS MMIO reference buffer mode for bangap voltage */
```

Code Listing 12 Configuring pulse detection

```
    Cy_SAR2_SetReferenceBufferMode(PASS0_EPASS_MMIO,
CY_SAR2_REF_BUF_MODE_OFF);

    /*Enable SAR0*/
    Cy_SAR2_Enable(MY_SAR_HW);

    /*Initialize SAR0 channel 0*/
    status = Cy_SAR2_Channel_Init(MY_SAR_HW, 0u, &MY_SAR_channel_0_config);
    if (CY_SAR2_SUCCESS != status)
    {
        CY_ASSERT(0);
    }

    /*Set SAR0 channel 0 interrupt mask*/
    Cy_SAR2_Channel_SetInterruptMask(MY_SAR_HW, 0u,CY_SAR2_INT_CH_PULSE);

    /*Configure and register SAR0 channel interrupts*/
    Cy_SysInt_Init(&SAR0_IRQ_cfg, sar0_interrupt_handler);
    NVIC_ClearPendingIRQ((IRQn_Type)SAR0_IRQ_cfg.intrSrc);
    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);
}
```

7.2 A/D conversion ISR for pulse detection

Figure 28 shows the example of an ADC ISR for pulse detection. For details on CPU interrupt handling, see the architecture reference manual mentioned in References.

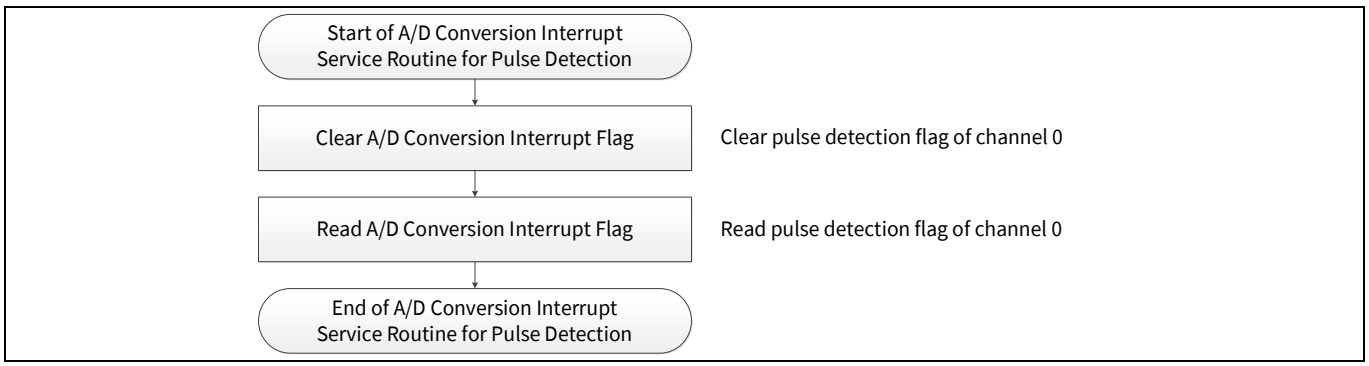

Figure 28 Example of ADC ISR for pulse detection

7.2.1 Configuration

Table 12 lists the functions of the global configuration part of in the PDL for the ADC

Table 12 Functions for configuring the ADC ISR for pulse detection

Functions	Description	Value
`Cy_SAR2_Channel_GetInterruptMaskedStatus(PASS SARchannel,INTR Source)`	Returns the interrupt status	–
`Cy_SAR2_Channel_ClearInterruptStatus(SAR Channel,Source)`	Clears the corresponding channel interrupt status	–

7.2.2 Sample code

See Code Listing 13 for the initial configuration of the ADC ISR for pulse detection.

Code Listing 13 Configuring the ADC ISR for pulse detection

```
static void sar0_interrupt_handler(void)
{
	/* Get interrupt status */
	uint32_t intrSource =
Cy_SAR2_Channel_GetInterruptStatusMasked(MY_SAR_HW, 0);
	if(CY_SAR2_INT_CH_PULSE == (intrSource & CY_SAR2_INT_CH_PULSE))
	{
		sar0_isr_set = true;
		/* Clear interrupt source */
		Cy_SAR2_Channel_ClearInterrupt(MY_SAR_HW, 0,
CY_SAR2_INT_CH_PULSE);
	}
}
```

8 Diagnosis function

This section shows flowcharts and example that explain how to use the diagnosis function on the ADC.

It is possible to input from the ADC to reference voltages V_{REFH} and V_{REFL}, as shown in Figure 29.

V_{REFH} is the upper-limit reference voltage. The A/D value is 0xFFF (= 4095).

V_{REFL} is the lower-limit reference voltage. The A/D value is 0x000 (= 0).

These are the functions for ADC diagnosis and ADC calibration.

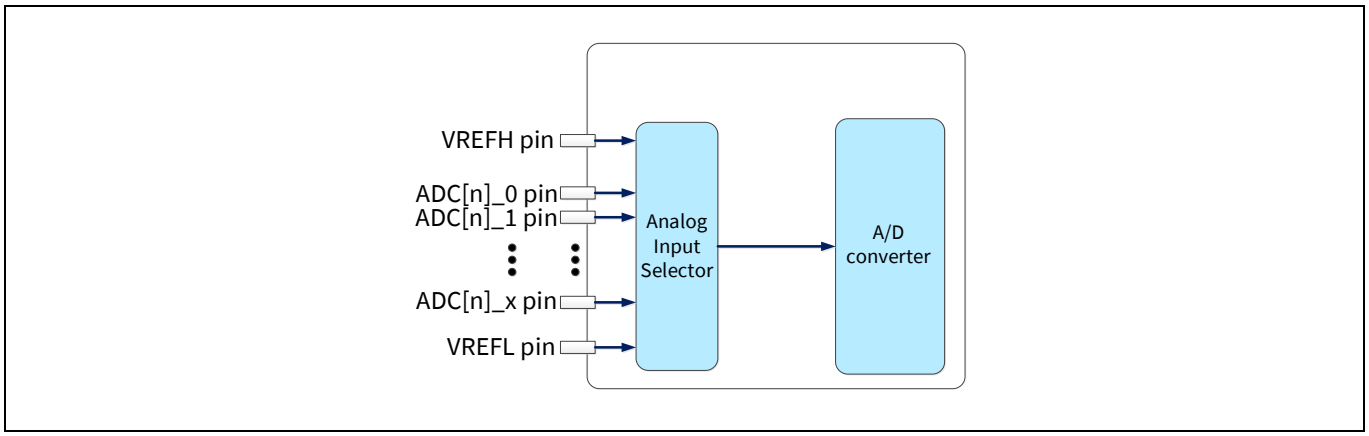

Figure 29 VREFH and VREFL as analog input

8.1 Check VZT and VFST

To conduct ADC diagnosis, see the datasheet mentioned in References to check the maximum value of the zero-transition voltage (VZT) and the minimum value of the full-scale transition voltage (VFST).

According to the datasheet, the maximum value of VZT is (V_{REFL} + 0.5 LSb) + 20 mV. The A/D value is 16.8 (0 + 0.5 + 20 / (5000 / 4096)).

Therefore, when V_{REFL} is input to the ADC, the value must be 16 or lower. (The A/D value ≤ 16 = 0x010.)

The minimum value of V_{FST} is (V_{REFH} – 1.5 LSb) – 20 mV. The AD value is 4077.1 (4095 – 1.5 – 20 / (5000 / 4096)).

Therefore, when V_{REFH} is input to the ADC, the value must be 4078 or higher. (The A/D value ≥ 4078 = 0xFEE.)

8.2 Diagnosis procedure

Figure 30 shows the example of the ADC diagnosis flowchart. In this example, the minimum value of the sample time is used. If the temperature sensor channel is enabled, reference buffer mode must also be enabled. To find the proper sample time for your system, see the datasheet mentioned in References.

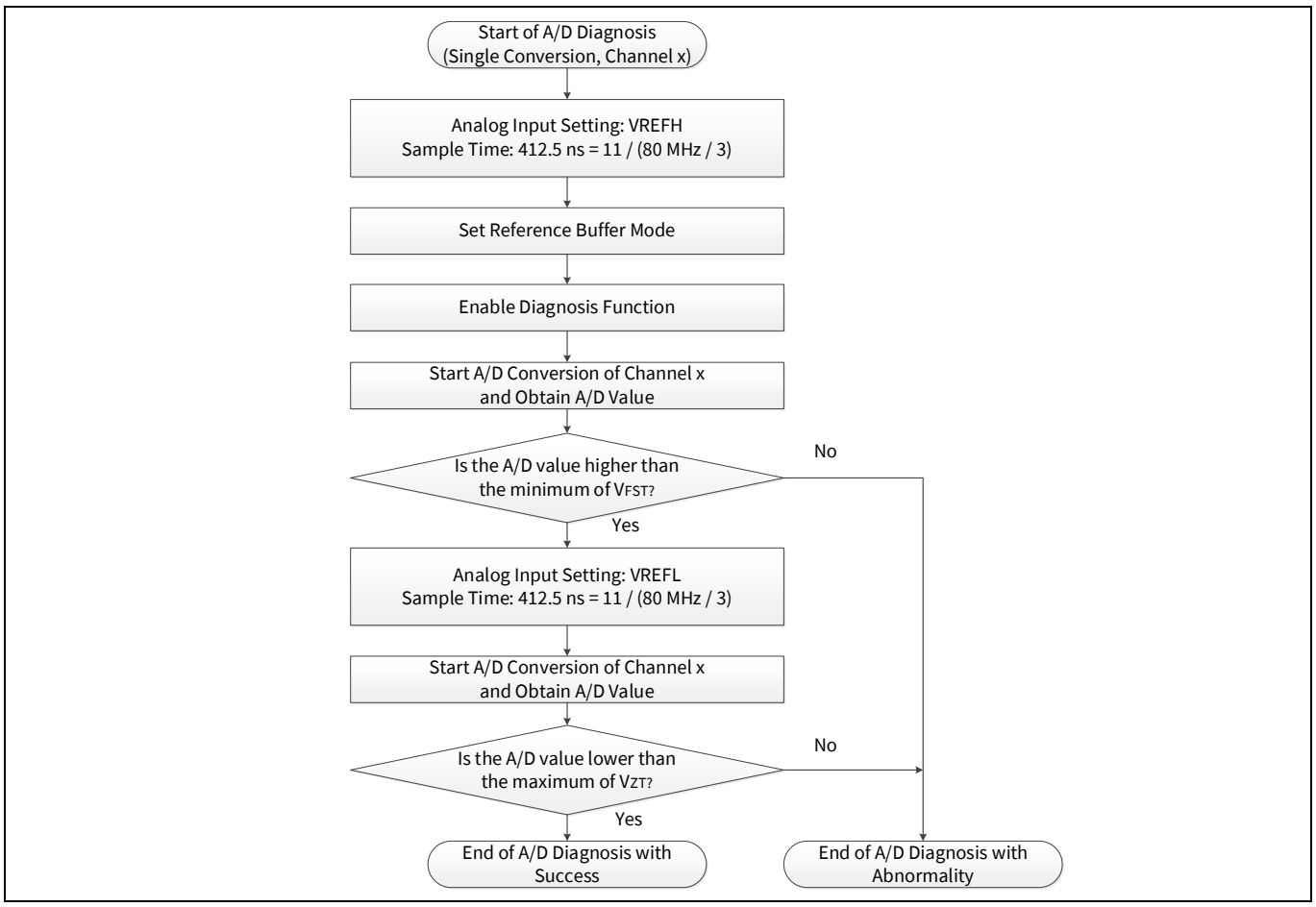

Figure 30 Example of ADC diagnosis flowchart

8.2.1 Use case

The following use case is an example of configuring ADC diagnosis.

- Analog input setting:
  - VREFH
  - VREFL

8.2.2 Configuration

Figure 31 lists the parameters and Table 13 lists the functions of the configuration part of in the PDL for ADC diagnosis.

Diagnosis function

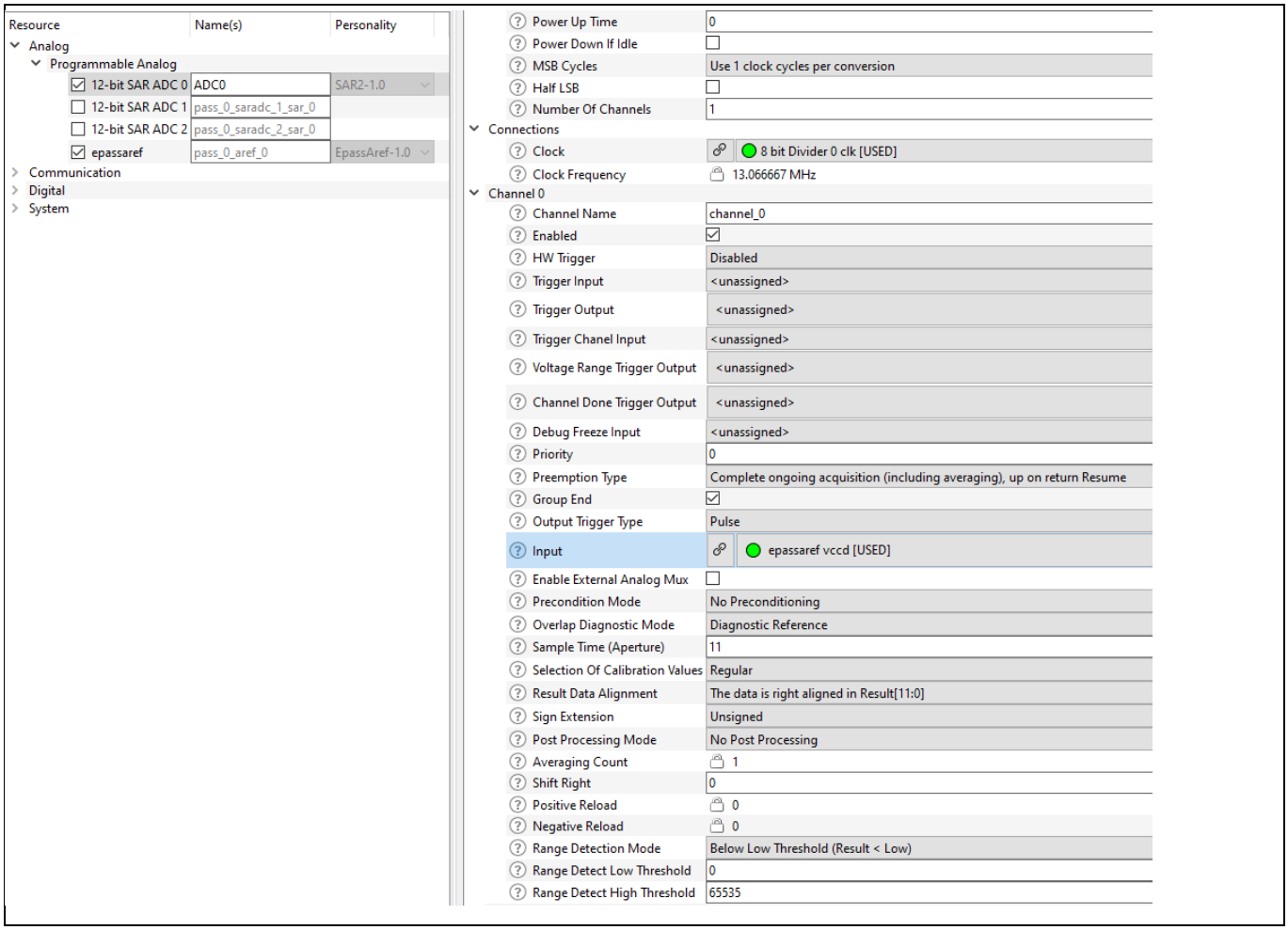

Figure 31 SAR2 channels configuration for diagnosis in Device Configurator

Table 13 Functions for configuring ADC diagnosis

Functions	Description	Value
`Cy_SAR2_Channel_Init(PASS SARchannel,ADCchannel Configure)`	Initializes the ADC channel	–
`Cy_SAR2_SetReferenceBufferMode(PASS0_EPASS_MMIO,RefBufMode)`	Sets ePASS MMIO reference buffer mode	–
`Cy_SAR2_Diag_Enable(AnalogMacro)`	Enables the diagnosis function	–
`Cy_SAR2_Channel_Enable(PASS SARchannel)`	Enables the corresponding channel	–
`Cy_SAR2_Channel_SoftwareTrigger(PASS SARchannel)`	Issues a software start trigger	–
`Cy_SAR2_Channel_GetGroupStatus(PASS SARchannel, GroupStatus)`	Returns the group conversion status	–
`Cy_SAR2_Channel_GetResult(SAR Channel,Status)`	Gets the conversion result and status	–

8.2.3 Sample code

See Code Listing 14 for sample code for the initial configuration of ADC diagnosis.

Code Listing 14 Configuring ADC diagnosis

```
/* ADC channel number definition */
#define ADC_LOGICAL_CHANNEL     (0u)
#define ADC_IRQ_NUM             (NvicMux2_IRQn)
#define ADC_INTR_NUM            (((ADC_IRQ_NUM << 16u) | ADC0_CH0_IRQ))
#define ADC_INTR_PRIORITY       (7u)

cy_stc_sar2_channel_config_t ADC0_channel_0_config_M =
{
    .channelHwEnable = true,
    .triggerSelection = CY_SAR2_TRIGGER_OFF,
    .channelPriority = 0U,
    .preenptionType = CY_SAR2_PREEMPTION_FINISH_RESUME,
    .doneLevel = CY_SAR2_DONE_LEVEL_PULSE,
    .isGroupEnd = true,
    .pinAddress = CY_SAR2_PIN_ADDRESS_VREF_H,
    .portAddress = CY_SAR2_PORT_ADDRESS_SARMUX0,
    .extMuxEnable = false,
    .extMuxSelect = 0U,
    .preconditionMode = CY_SAR2_PRECONDITION_MODE_OFF,
    .overlapDiagMode = CY_SAR2_OVERLAP_DIAG_MODE_OFF,
    .sampleTime = 11U,
    .calibrationValueSelect = CY_SAR2_CALIBRATION_VALUE_REGULAR,
    .postProcessingMode = CY_SAR2_POST_PROCESSING_MODE_NONE,
    .resultAlignment = CY_SAR2_RESULT_ALIGNMENT_RIGHT,
    .signExtention = CY_SAR2_SIGN_EXTENTION_UNSIGNED,
    .averageCount = 1U,
    .rightShift = 0U,
    .positiveReload = 0U,
    .negativeReload = 0U,
    .rangeDetectionMode = CY_SAR2_RANGE_DETECTION_MODE_BELOW_LO,
    .rangeDetectionLoThreshold = 0U,
    .rangeDetectionHiThreshold = 65535U,
};

cy_stc_sar2_config_t ADC0_config_M =
```

Diagnosis function

Code Listing 14 Configuring ADC diagnosis

```
{
    .preconditionTime = 0U,
    .powerupTime = 0U,
    .enableIdlePowerDown = false,
    .msbStretchMode = CY_SAR2_MSB_STRETCH_MODE_1CYCLE,
    .enableHalfLsbConv = false,
    .sarMuxEnable = true,
    .adcEnable = true,
    .sarIpEnable = true,
    .channelConfig = {(cy_stc_sar2_channel_config_t
*)&ADC0_channel_0_config_M,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL},
};

uint8_t Test_Completed = 0;
uint8_t Flag_VREFL = 0;
int main(void)
{
    cy_rslt_t result;

    /* 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)
      {
```

Diagnosis function

Code Listing 14 Configuring ADC diagnosis

```
            CY_ASSERT(0);
      }
      printf("\x1b[2J\x1b[;H");
      printf("**************************************************************
******************\r\n");
      printf("XMC7000 MCU: ADC Diagnosis\r\n");
      printf("**************************************************************
******************\r\n");

    /* Initialize the SAR2 module */
    Cy_SAR2_Init(ADC0_HW, &ADC0_config_M);
    /* Set ePASS MMIO reference buffer mode for bangap voltage */
    Cy_SAR2_SetReferenceBufferMode(PASS0_EPASS_MMIO,
CY_SAR2_REF_BUF_MODE_OFF);

    /* Initialize the SAR2 Channel to VERF_H */
    ADC0_channel_0_config_M.pinAddress = CY_SAR2_PIN_ADDRESS_VREF_H;
    ADC0_channel_0_config_M.portAddress = CY_SAR2_PORT_ADDRESS_SARMUX0;
    Cy_SAR2_Channel_Init(ADC0_HW,0, &ADC0_channel_0_config_M);

    /* Issue software start trigger */
    Cy_SAR2_Channel_SoftwareTrigger(ADC0_HW, 0);
    Cy_SysLib_Delay(10);

    for (;;)
    {
      DiagnosisProcedure();
      if(Test_Completed == 1 ) //&& status == CY_SAR2_SUCCESS
      {
            while(1);
      }
      Cy_SysLib_Delay(10);
    }
}

void DiagnosisProcedure(void)
{
      uint32_t adc_status = 0;
```

Diagnosis function

Code Listing 14 Configuring ADC diagnosis

```
    uint16_t resultBuff;

       resultBuff = Cy_SAR2_Channel_GetResult(ADC0_HW, 0, &adc_status);
       if(CY_SAR2_STATUS_VALID == (adc_status & CY_SAR2_STATUS_VALID))
       {
              /* Print out the ADC conversion result by UART */
              printf("ADC VREF_H result: %04d\r\n", resultBuff);
       }

    /* Is the A/D value Higher than the Minimum of Vfst */
    if(resultBuff >= 4078 && Flag_VREFL == 0) /*  (VREFH -1.5 LSb) - 20 mV
*/
    {
        Flag_VREFL = 1;

        /* Initialize the SAR2 Channel to VERF_H */
        ADC0_channel_0_config_M.pinAddress = CY_SAR2_PIN_ADDRESS_VREF_L;
        ADC0_channel_0_config_M.portAddress = CY_SAR2_PORT_ADDRESS_SARMUX0;
        Cy_SAR2_Channel_Init(ADC0_HW,0, &ADC0_channel_0_config_M);

        /* Trigger next conversion */
        Cy_SAR2_Channel_SoftwareTrigger(ADC0_HW, 0);
        Cy_SysLib_Delay(10);
    }
    else
    {
        /* A/D value lower than the Minimum of Vfst */
       printf("ADC Diagnosis Failed\r\n");
    }

    if(Flag_VREFL == 1)
    {
       resultBuff = Cy_SAR2_Channel_GetResult(ADC0_HW, 0, &adc_status);
       if(CY_SAR2_STATUS_VALID == (adc_status & CY_SAR2_STATUS_VALID))
       {
              /* Print out the ADC conversion result by UART */
              printf("ADC VREF_L result: %04d\r\n", resultBuff);
       }
```

Diagnosis function

Code Listing 14 Configuring ADC diagnosis

```
      /*Is the AD Value lower than the Maximum of Vzt */
         if(resultBuff <= 17) /* (VREFL + 0.5 LSb) + 20 mV */
         {
             Flag_VREFL = 0;
             Test_Completed = 1;

             printf("ADC Diagnosis Successfully\r\n");
         }
         else
         {
             /*AD Value Higher than the Maximum of Vzt */
              printf("ADC Diagnosis Failed\r\n");
         }
     }
}
```

9 Calibration function

It is possible to adjust the offset and gain of the ADC using the SARn_ANA_CAL register.

This section describes the effects of this register and how to process the calibration of the ADC.

9.1 Offset adjustment direction

The ADC has an offset adjustment function to compensate for offset error. The SARn_ANA_CAL[7:0] register can adjust the offset. It is possible to select code from +127 to –128 in a decimal number for SARn_ANA_CAL[7:0]. The offset adjustment step is a quarter of 1LSb.

The equation follows (OFST = the value of SARn_ANA_CAL[7:0]):

$$\text{Output Digital Code} = \max\left(0, \min\left(4095, \text{floor}\left(\frac{\text{VIN}}{\text{VREFH}} \times 4096 + \frac{\text{OFST}}{4}\right)\right)\right)$$

The relationship between OFST and the offset shift direction is shown in Figure 32.

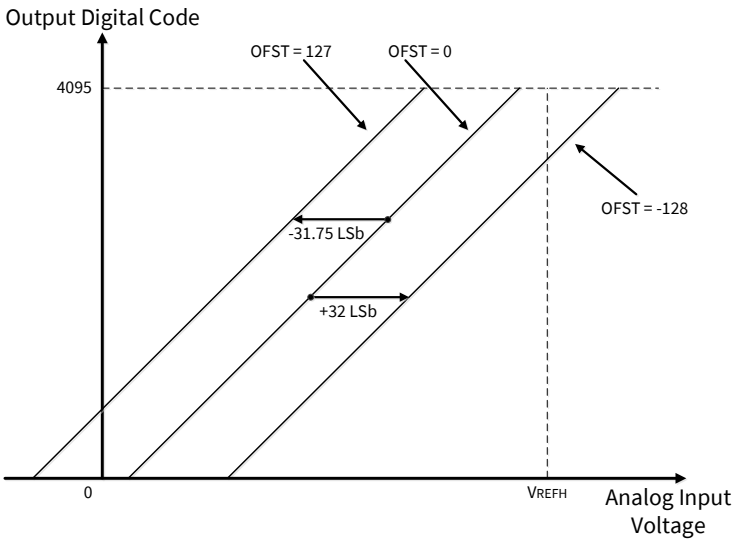

Figure 32 Relationship between OFST and offset shift direction

9.2 Gain adjustment direction

The ADC has a gain adjustment function to compensate for gain error. The SARn_ANA_CAL[20:16] register can adjust the gain. It is possible to select code from +15 to –15 in a decimal number for SARn_ANA_CAL[20:16]. The gain adjustment step is a quarter of 1LSb.

The equation follows (gain= the value of SARn_ANA_CAL[20:16]):

$$\text{Output Digital Code} = \max\left(0, \min\left(4095, \text{floor}\left(\frac{4096 - \text{GAIN}}{\text{VREFH}} \times \left(VIN - \frac{\text{VREFH}}{2}\right) + 2048\right)\right)\right)$$

Calibration function

The relationship between the gain and the gain shift direction is shown in Figure 33.

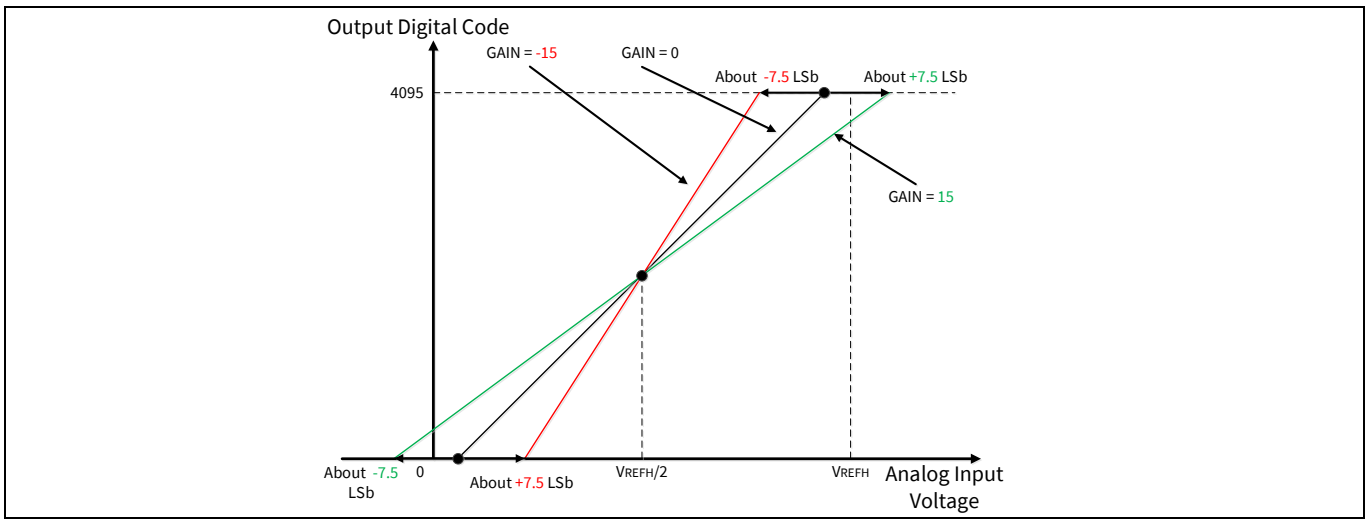

Figure 33 Relationship between gain and gain shift direction

9.3 Calibration procedure

Figure 34 shows the example flowchart for ADC calibration. First, adjust the offset, and then adjust the gain. If the temperature sensor channel is enabled, reference buffer mode must also be enabled.

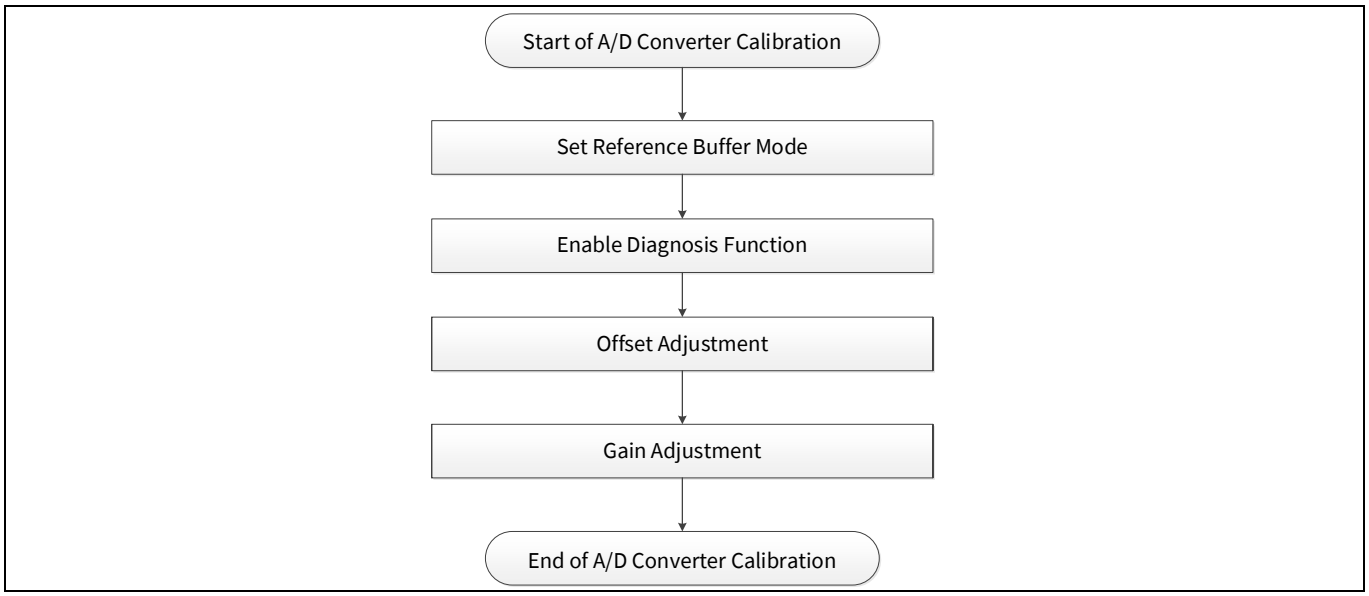

Figure 34 Example flowchart of ADC calibration

9.3.1 Configuration

Table 14 lists the parameters of the configuration part of in the PDL for the ADC calibration procedure.

Calibration function

Table 14 Functions for configuring the ADC calibration procedure

Functions	Description	Value
`Cy_SAR2_SetReferenceBufferMode(PASS0_EPASS_MMIO,RefBufMode)`	Sets ePASS MMIO reference buffer mode	–
`Cy_SAR2_Diag_Enable(AnalogMacro)`	Enables the diagnosis function	–

9.3.2 Sample code

See Code Listing 15 for a sample code for initial configuration of the ADC calibration procedure.

Code Listing 15 Configuring the ADC calibration procedure

```
/* ADC channel number definition */
#define ADC_LOGICAL_CHANNEL     (0u)
#define ADC_IRQ_NUM             (NvicMux2_IRQn)
#define ADC_INTR_NUM            (((ADC_IRQ_NUM << 16u) | ADC0_CH0_IRQ))
#define ADC_INTR_PRIORITY       (7u)

/*******************************************************************************
*****
* Global Variables
*******************************************************************************
****/
/* Calibration Setting */
cy_stc_sar2_analog_calibration_conifg_t calibrationConfig =
{
    .offset = 127ul,/* Offset Calibration Setting */
    .gain   = 15ul,/* Gain Calibration Setting */
};


cy_stc_sar2_channel_config_t ADC0_channel_0_config_M =
{
    .channelHwEnable = true,
    .triggerSelection = CY_SAR2_TRIGGER_OFF,
    .channelPriority = 0U,
    .preenptionType = CY_SAR2_PREEMPTION_FINISH_RESUME,
    .doneLevel = CY_SAR2_DONE_LEVEL_PULSE,
    .isGroupEnd = true,
    .pinAddress = CY_SAR2_PIN_ADDRESS_VREF_H,
    .portAddress = CY_SAR2_PORT_ADDRESS_SARMUX0,
    .extMuxEnable = false,
```

Calibration function

Code Listing 15 Configuring the ADC calibration procedure

```
    .extMuxSelect = 0U,
    .preconditionMode = CY_SAR2_PRECONDITION_MODE_OFF,
    .overlapDiagMode = CY_SAR2_OVERLAP_DIAG_MODE_OFF,
    .sampleTime = 11U,
    .calibrationValueSelect = CY_SAR2_CALIBRATION_VALUE_REGULAR,
    .postProcessingMode = CY_SAR2_POST_PROCESSING_MODE_NONE,
    .resultAlignment = CY_SAR2_RESULT_ALIGNMENT_RIGHT,
    .signExtention = CY_SAR2_SIGN_EXTENTION_UNSIGNED,
    .averageCount = 1U,
    .rightShift = 0U,
    .positiveReload = 0U,
    .negativeReload = 0U,
    .rangeDetectionMode = CY_SAR2_RANGE_DETECTION_MODE_BELOW_LO,
    .rangeDetectionLoThreshold = 0U,
    .rangeDetectionHiThreshold = 65535U,
};

cy_stc_sar2_config_t ADC0_config_M =
{
    .preconditionTime = 0U,
    .powerupTime = 0U,
    .enableIdlePowerDown = false,
    .msbStretchMode = CY_SAR2_MSB_STRETCH_MODE_1CYCLE,
    .enableHalfLsbConv = false,
    .sarMuxEnable = true,
    .adcEnable = true,
    .sarIpEnable = true,
    .channelConfig = {(cy_stc_sar2_channel_config_t
*)&ADC0_channel_0_config_M,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL},
};
int main(void)
{
    cy_rslt_t result;

    /* Initialize the device and board peripherals */
```

Calibration function

Code Listing 15 Configuring the ADC calibration procedure

```
    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: ADC Calibration\r\n");
      printf("**************************************************************
******************\r\n");

    /* Initialize the SAR2 module */
    Cy_SAR2_Init(ADC0_HW, &ADC0_config_M);
    /* Set ePASS MMIO reference buffer mode for bangap voltage */
    Cy_SAR2_SetReferenceBufferMode(PASS0_EPASS_MMIO,
CY_SAR2_REF_BUF_MODE_OFF);

    /* Set Gain Coreection to 0" */
    calibrationConfig.gain = 0ul;

    /* Initialize the SAR2 Channel to VERF_H */
    ADC0_channel_0_config_M.pinAddress = CY_SAR2_PIN_ADDRESS_VREF_L;
    ADC0_channel_0_config_M.portAddress = CY_SAR2_PORT_ADDRESS_SARMUX0;
```

Calibration function

Code Listing 15 Configuring the ADC calibration procedure

```
    Cy_SAR2_Channel_Init(ADC0_HW,0, &ADC0_channel_0_config_M);

    /* Set Offset Compensation to "127" */
    calibrationConfig.offset = 127ul;

    /* Issue software start trigger */
    Cy_SAR2_Channel_SoftwareTrigger(ADC0_HW, 0);
    Cy_SysLib_Delay(10);

    for (;;)
    {
       if(adc_offset_calibration() == true)
           {
               if(adc_gain_calibration() == true)
               {
                   /* Test Completed */
                    printf("ADC Calibration Successfully\r\n");
                   while(1);
               }
               else
               {
                   /* Error */
                    printf("ADC Gain Calibration Failed: \r\n");
                   while(1);
               }
           }
           else
           {
               /* Error */
              printf("ADC Offset Calibration Failed\r\n");
               while(1);
           }
    }
}

uint16_t getAdcValue(void)
{
    /* Trigger ADC. */
```

Calibration function

Code Listing 15 Configuring the ADC calibration procedure

```
    Cy_SAR2_Channel_SoftwareTrigger(ADC0_HW, 0U);
    /* Wait for group of single channel to complete. */
    while (CY_SAR2_INT_GRP_DONE !=
Cy_SAR2_Channel_GetInterruptStatus(ADC0_HW, 0U));
    Cy_SAR2_Channel_ClearInterrupt(ADC0_HW, 0U, CY_SAR2_INT_GRP_DONE);
    /* Get the result(s) */
    uint16_t resultBuff = Cy_SAR2_Channel_GetResult(ADC0_HW, 0U, NULL);
    printf("ADC VREF result: %04d\r\n", resultBuff);
    return (resultBuff);
}
```

9.4 Offset adjustment procedure

Figure 35 shows the example flowchart of offset adjustment.

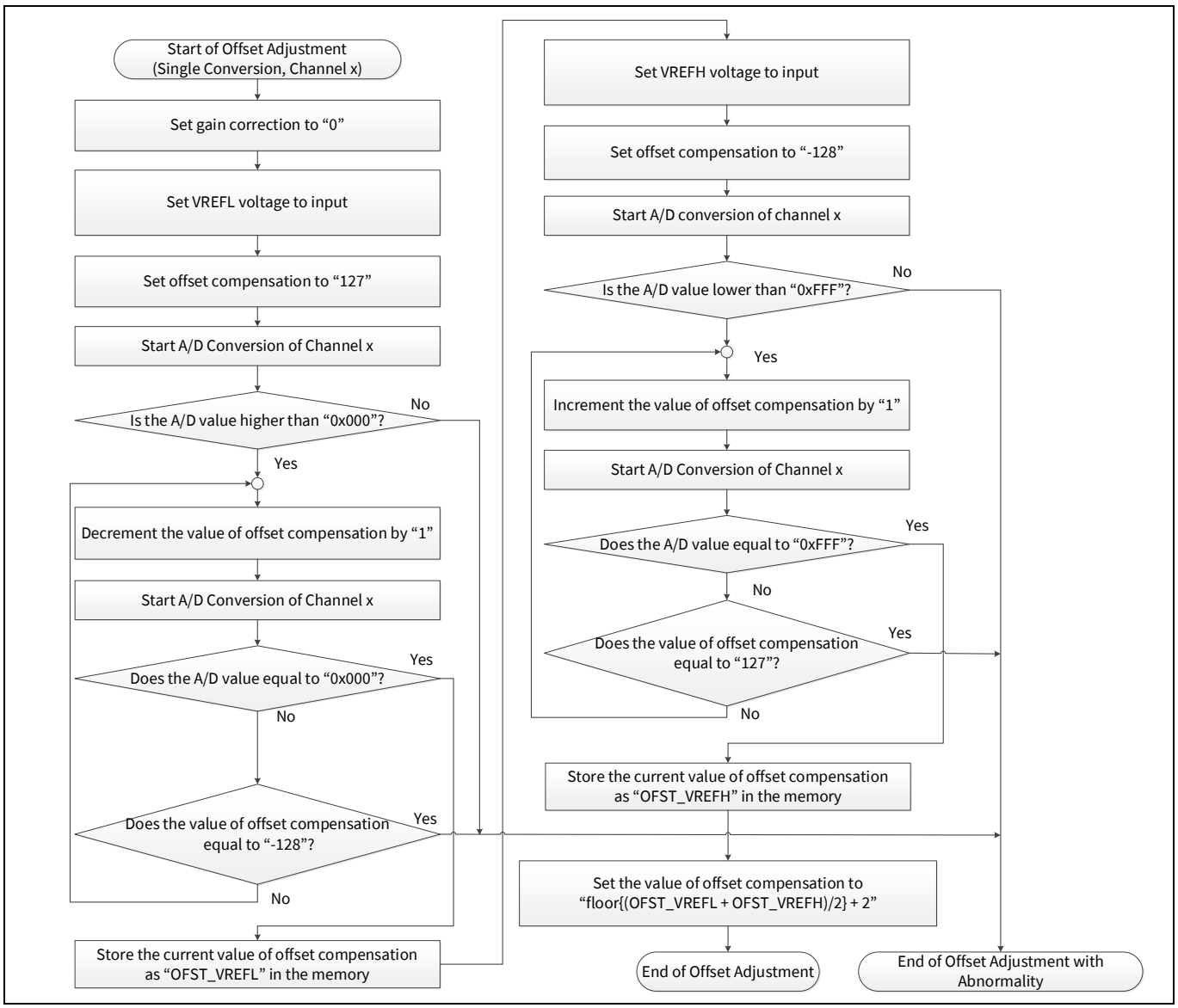

Figure 35 Example flowchart of offset adjustment

9.4.1 Use case

The following use case is an example of offset adjustment using ADC logical channel 0.

9.4.2 Configuration

Table 15 lists the functions of the configuration part of in the PDL for offset adjustment procedure.

Table 15 Functions for configuring offset adjustment procedure

Functions	Description	Value
`Cy_SAR2_Channel_SoftwareTrigger(PASS SARchannel)`	Issues a software start trigger	–
`Cy_SAR2_Channel_GetGroupStatus(PASS SARchannel, GroupStatus)`	Returns the group conversion status	–

Calibration function

Functions	Description	Value
`Cy_SAR2_Channel_GetResult(SAR Channel,Status)`	Gets the conversion result and status	–
`Cy_SAR2_SetAnalogCalibrationValue(PASS SARchannel,&calibrationConfig)`	Sets the analog calibration value	–
`Cy_SAR2_Channel_Init(PASS SARchannel,ADCchannel Configure)`	Initializes the ADC channel	–
`Cy_SAR2_SetReferenceBufferMode(PASS0_EPASS_MMIO,RefBufMode)`	Sets ePASS MMIO reference buffer mode	–
`Cy_SAR2_Diag_Enable(AnalogMacro)`	Enables the diagnosis function	–
`Cy_SAR2_Channel_Enable(PASS SARchannel)`	Enables the corresponding channel	–

9.4.3 Sample code

See Code Listing 16 for sample code snippets for initial configuration of offset adjustment procedure settings.

Code Listing 16 Configuring the offset adjustment procedure

```
bool adc_offset_calibration(void)
{
    /* Start ADC conversion and get the ADC value */
    int16_t result = getAdcValue();
    int16_t offset_VREFH = 0U;
    int16_t offset_VREFL = 0U;
    int16_t value = 0U;

    /* Initialize the SAR2 Channel to VERF_L */
    ADC0_channel_0_config_M.pinAddress = CY_SAR2_PIN_ADDRESS_VREF_L;
    ADC0_channel_0_config_M.portAddress = CY_SAR2_PORT_ADDRESS_SARMUX0;
    Cy_SAR2_SetAnalogCalibrationValue(ADC0_HW, &calibrationConfig);
    /* Initialize channel. */
    Cy_SAR2_Channel_Init(ADC0_HW,0, &ADC0_channel_0_config_M);

    if(result > 0)
    {
        /* Decrement the Value of Offset Compensation by "1" */
        for(offset_VREFL = 127; offset_VREFL >= -128; offset_VREFL -= 1)
        {
            /* Set "offset_VREFL" to offset register value */
```

Calibration function

Code Listing 16 Configuring the offset adjustment procedure

```
            calibrationConfig.offset = offset_VREFL;
            Cy_SAR2_SetAnalogCalibrationValue(ADC0_HW, &calibrationConfig);
            /* Get ADC conversion value. */
            result = getAdcValue();
            /* Does the ADC conversion value equal to 0? */
            if(result == 0)
            {
             printf("ADC offset_VREFL value: %d\r\n", offset_VREFL);
             printf("ADC Offset VREF_L Calibration Successfully\r\n");
                break;
            }
            /* If offset has reached lower bound. */
            if(offset_VREFL == -128)
            {
                /* Report error. */
                return false;
            }
        }
    }
    else
    {
        /* End of Offset Adjustment with Abnormality. */
        return false;
    }
    /* Initialize the SAR2 Channel to VERF_H */
    ADC0_channel_0_config_M.pinAddress = CY_SAR2_PIN_ADDRESS_VREF_H;
    ADC0_channel_0_config_M.portAddress = CY_SAR2_PORT_ADDRESS_SARMUX0;
       /* Set Offset Compensation to "-128" and init channel. */
       calibrationConfig.offset = -128;
    Cy_SAR2_SetAnalogCalibrationValue(ADC0_HW, &calibrationConfig);
    /* Initialize channel. */
    Cy_SAR2_Channel_Init(ADC0_HW,0, &ADC0_channel_0_config_M);

    /* Start ADC Conversion and get the A/D value. */
    result = getAdcValue();
    /* Is the ADC Conversion value lower than 0xFFF */
    if(result < 0xFFF)
    {
```

Calibration function

Code Listing 16 Configuring the offset adjustment procedure

```
        /* Increment the Value of Offset Compensation by "1". */
        for(offset_VREFH = -128; offset_VREFH <= 127; offset_VREFH += 1)
        {
            /* Set "offset_VREFH" to offset register value. */
            calibrationConfig.offset = offset_VREFH;
            Cy_SAR2_SetAnalogCalibrationValue(ADC0_HW, &calibrationConfig);
            /* Get ADC Conversion value. */
            result = getAdcValue();
            /* Does the ADC Conversion value Equal to "4095"? */
            if(result == 0xFFF)
            {
             printf("ADC offset_VREFH value: %d\r\n", offset_VREFH);
             printf("ADC Offset VREF_H Calibration Successfully\r\n");
                break;
            }
            /* Does the value of offset compensation equal to "127"? */
            if(offset_VREFH == 127)
            {
                /* Offset adjustment failed - report error. */
                return false;
            }
        }
    }
    else
    {
       printf("ADC Offset VREF_H Calibration Failed\r\n");
        /* Offset adjustment failed - report error. */
        return false;
    }
    /* Does "(floor((offset_VREFH+offset_VREFL)/2) + 2)" greater than "125"
?*/
    value = (floor((offset_VREFH+offset_VREFL)/2U) + 2U);
    if((value - 125U) <= 0)
    {
       printf("ADC Offset VREF Calibration Value: %d\r\n", value);
        /* Offset adjustment failed - report error. */
        return false;
    }
```

Code Listing 16 Configuring the offset adjustment procedure

```
    /* Set the value of offset compensation to
(floor((offset_VREFH+offset_VREFL)/2) + 2) */
    calibrationConfig.offset = (floor((offset_VREFH+offset_VREFL)/2U) +
2U);


    return true;
}
```

9.5 Gain adjustment procedure

Figure 36 shows the example flowchart of gain adjustment.

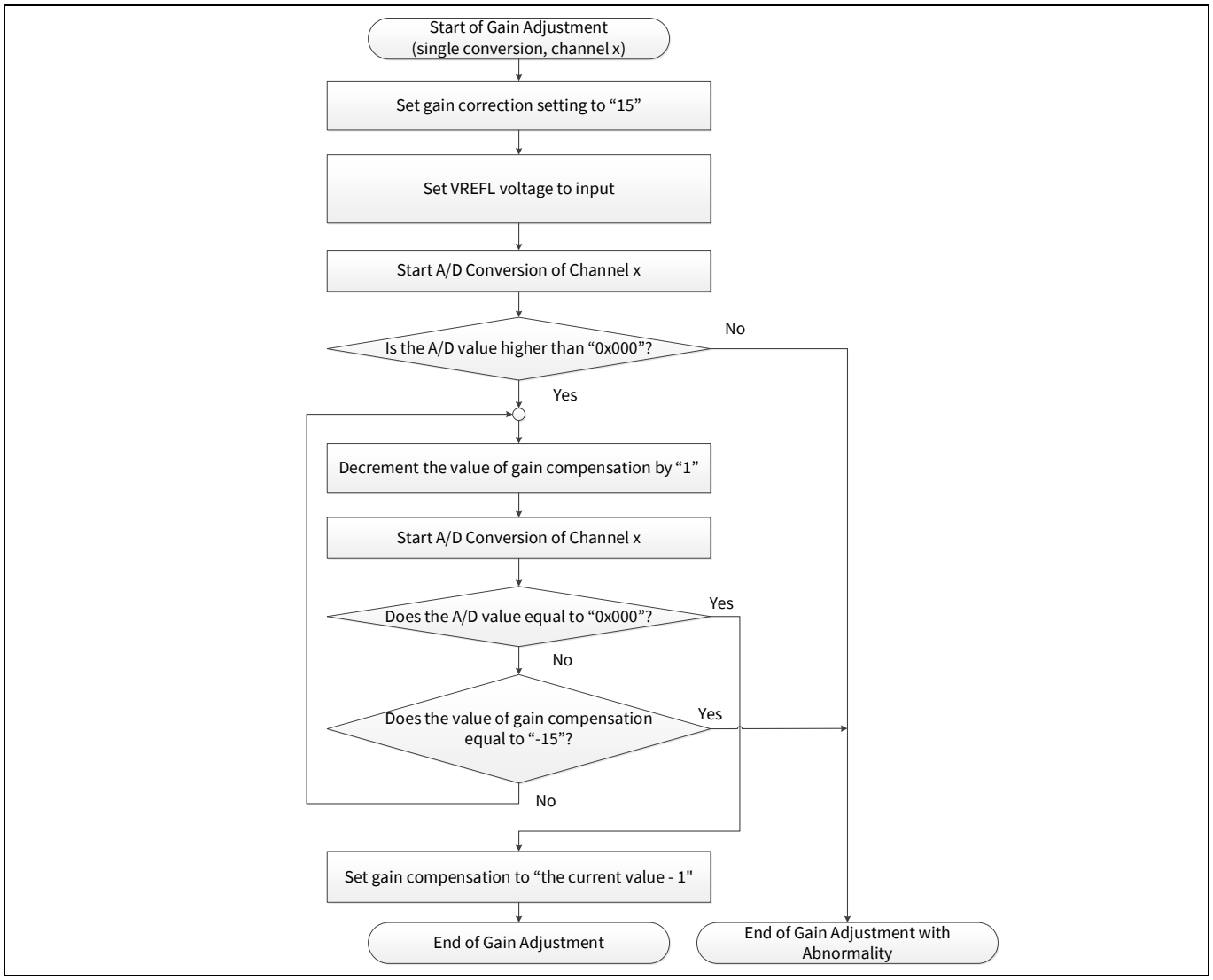

Figure 36 Example flowchart of gain adjustment

9.5.1 Use case

The following use case is an example of gain adjustment using ADC logical channel 0.

9.5.2 Configuration

Table 16 lists the functions of the global configuration part of in the PDL for the ADC .

Table 16 Functions for configuring the gain adjustment procedure

Functions	Description	Value
`Cy_SAR2_Channel_SoftwareTrigger(PASS SARchannel)`	Issues a software start trigger	–
`Cy_SAR2_Channel_GetGroupStatus(PASS SARchannel, GroupStatus)`	Returns the group conversion status	–
`Cy_SAR2_Channel_GetResult(SAR Channel,Status)`	Gets the conversion result and status	–
`Cy_SAR2_SetAnalogCalibrationValue(PASS SARchannel,&calibrationConfig)`	Sets the analog calibration value	–
`Cy_SAR2_Channel_Init(PASS SARchannel,ADCchannel Configure)`	Initializes the ADC channel	–
`Cy_SAR2_Channel_Enable(PASS SARchannel)`	Enables the corresponding channel	–

9.5.3 Sample code

See Code Listing 17 for sample code for the initial configuration of the gain adjustment procedure.

Code Listing 17 Configuring the gain adjustment procedure

```
bool adc_gain_calibration(void)
{
    int16_t gain;
    int16_t result = 0U;
    /* Set Gain Correction setting to "15". */
    calibrationConfig.gain = 15;

    /* Initialize the SAR2 Channel to VERF_L */
    ADC0_channel_0_config_M.pinAddress = CY_SAR2_PIN_ADDRESS_VREF_L;
    ADC0_channel_0_config_M.portAddress = CY_SAR2_PORT_ADDRESS_SARMUX0;
    /* Initialize channel. */
       Cy_SAR2_Channel_Init(ADC0_HW,0, &ADC0_channel_0_config_M);
       /* Apply the initial calibration settings. */
    Cy_SAR2_SetAnalogCalibrationValue(ADC0_HW, &calibrationConfig);

    /* Start ADC value Conversion */
    result = getAdcValue();
    if(result > 0)
    {
         for(gain = 15; gain >=-14; gain -= 1)
         {
            /* Set gain to register */
            calibrationConfig.gain = gain;
            Cy_SAR2_SetAnalogCalibrationValue(ADC0_HW, &calibrationConfig);
            result = getAdcValue();
            if(result == 0)
            {
                /* Set gain compensation to "gain - 1" */
                calibrationConfig.gain = gain - 1;
                Cy_SAR2_SetAnalogCalibrationValue(ADC0_HW,
&calibrationConfig);
                printf("ADC Gain Calibration Successfully\r\n");
                break;
            }
            if(gain == -14)
```

Calibration function

Code Listing 17 Configuring the gain adjustment procedure

```
            {
                /* Gain adjustment failed - report error. */
             printf("ADC Gain Calibration Failed\r\n");
                return false;
            }
        }
    }
    return true;
}
```

10 Temperature sensor

The temperature sensor can measure the chip temperature using an ADC. To accurately measure the temperature at runtime, use the reference measurement done during production. This reference data is stored in the SFlash along with other calibration data. See the “SAR ADC” chapter of the architecture reference manual for the exact address to read this data.

This section shows an example chip temperature measurement flow.

Ensure that you have configured the settings as described in the Basic ADC global configuration section.

10.1 Temperature measurement procedure

Figure 35 shows the example flowchart of the chip temperature measurement.

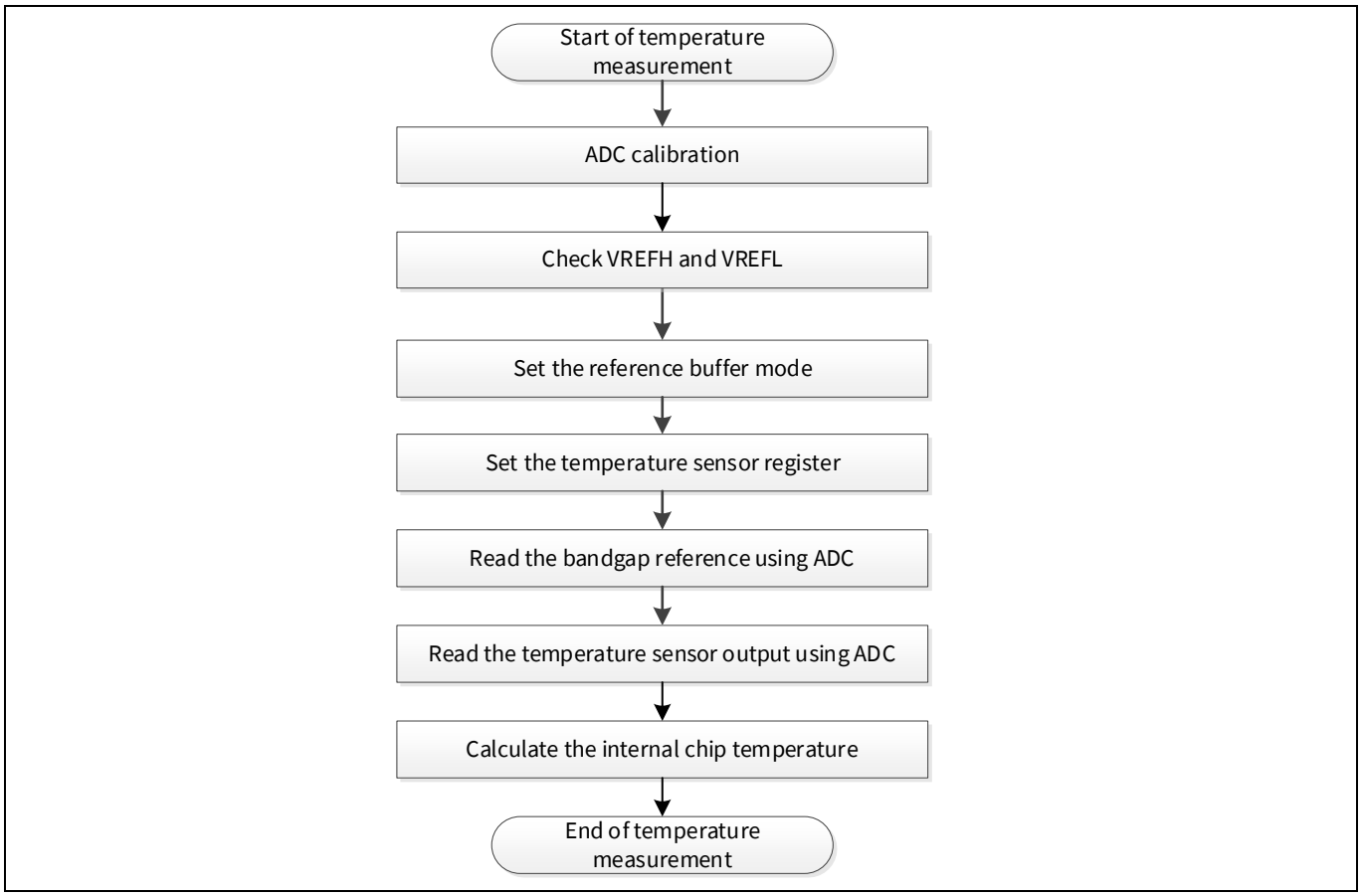

Figure 37 Example flowchart of temperature measurement

1. ADC calibration: See Section 9.3 for example of offset and gain adjustment.
2. Check VREFH and VREL: See Section 8.2 for example of VREFH and VREFL diagnosis.
3. Set the reference buffer mode: Set the PASS_CTL register to enable reference buffer mode.
4. Set the temperature sensor register: Set bit 9, 8 and 6 of PASS_TEST_CTL register to ‘1’ only for CYT2B. For other devices, keep the default value.
5. Read the bandgap reference (V_{BG}) using ADC: Store the A/D conversion result of the V_{BG}.
6. Read the temperature sensor output (V_{BE}) using ADC: Store the A/D conversion result of the V_{BE}. To find the proper sample time for the temperature sensor, see the datasheet mentioned in References.

7. Calculate the internal chip temperature:
   a) V_{BE} (temperature sensor output) has a second-order dependency on temperature (T) and can be described by the following equation:

$$\mathrm{VBE} = aT^2 + bT + \mathrm{c}$$

To calculate the temperature from V_{BE}, you must know the coefficients (a, b, and c) of the above equation. These coefficients can be calculated using the data (V_{BE} measured at three different temperatures) stored in the SFlash. See the architecture reference manual for details. The three combinations of (V_{BE}, T) to be used depend on the supply voltage (V_{DDA}). For example, to calculate polynomial coefficients:

- When V_{DDA} = 3.3 V, use SFlash data set#0 and set#1 in the architecture reference manual
- When V_{DDA} = 5.0 V, use SFlash data set#0 and set#2 in the architecture reference manual

   b) Because the ADC reference voltage may have changed from the calibration, V_{BE} must be scaled using the ratio of V_{BG_S3} and V_{BG}, where $\mathrm{VBE\_NEW} = VBE \times \left(\frac{VBG\_S3}{VBG}\right)$ before it is used to calculate the temperature.

   c) Calculate temperature using the above polynomial

$$\mathrm{VBE\_NEW} = aT^2 + bT + \mathrm{c}$$

   d) After calculating the temperature, the accuracy can be improved by using the bandgap reference from the nearest temperature in step b). Essentially, if the temperature calculated in step c) is close to

- COLD (-40), repeat step b) with V_{BG_S2} and recalculate the temperature using step c).
- HOT (150), repeat step b) with V_{BG_CHI} and recalculate the temperature using step c).
- ROOM (27), temperature calculated in step c) is the final temperature.

10.2 Use case

- V_{DDDA} = 5.0 V
- ADC logical channel: 0
- Number of iterations to get the ADC readings: 16
- Sampling time: 120 ADC clock cycles

10.3 Configuration

Table 18 lists the functions of the configuration part of in the PDL for temperature measurement.

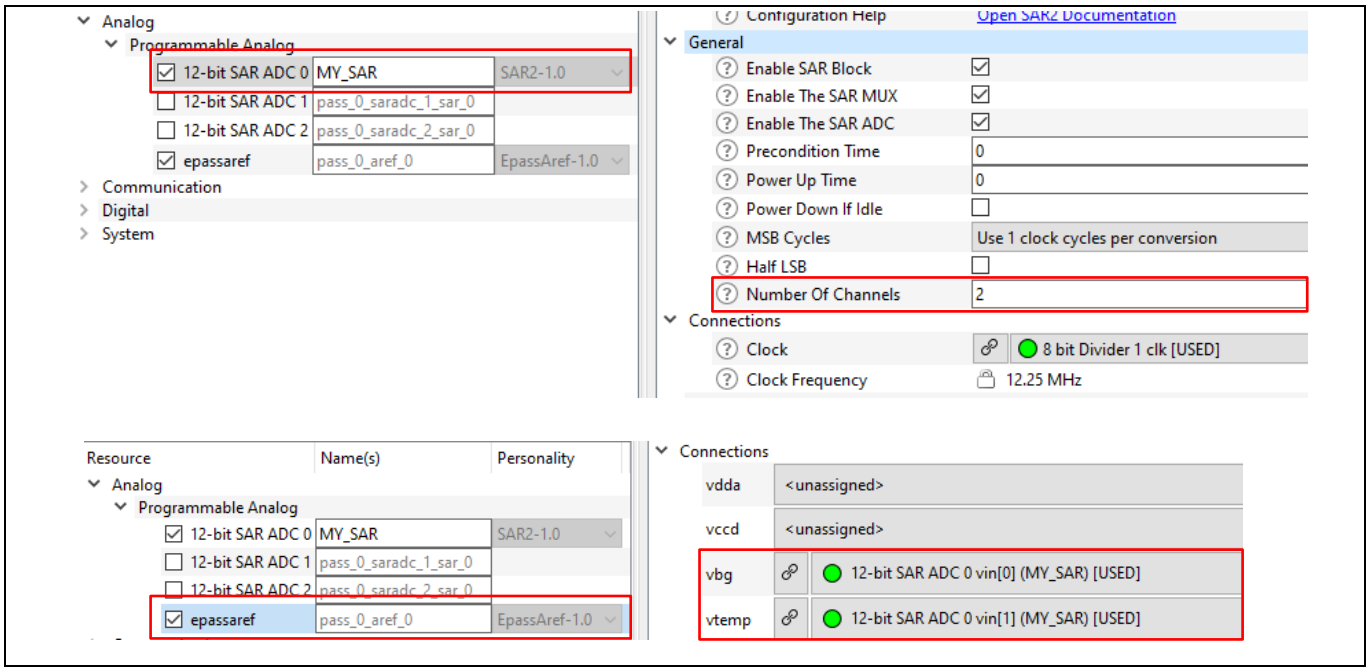

Figure 38 SAR2 basic configuration for temperature in Device Configurator

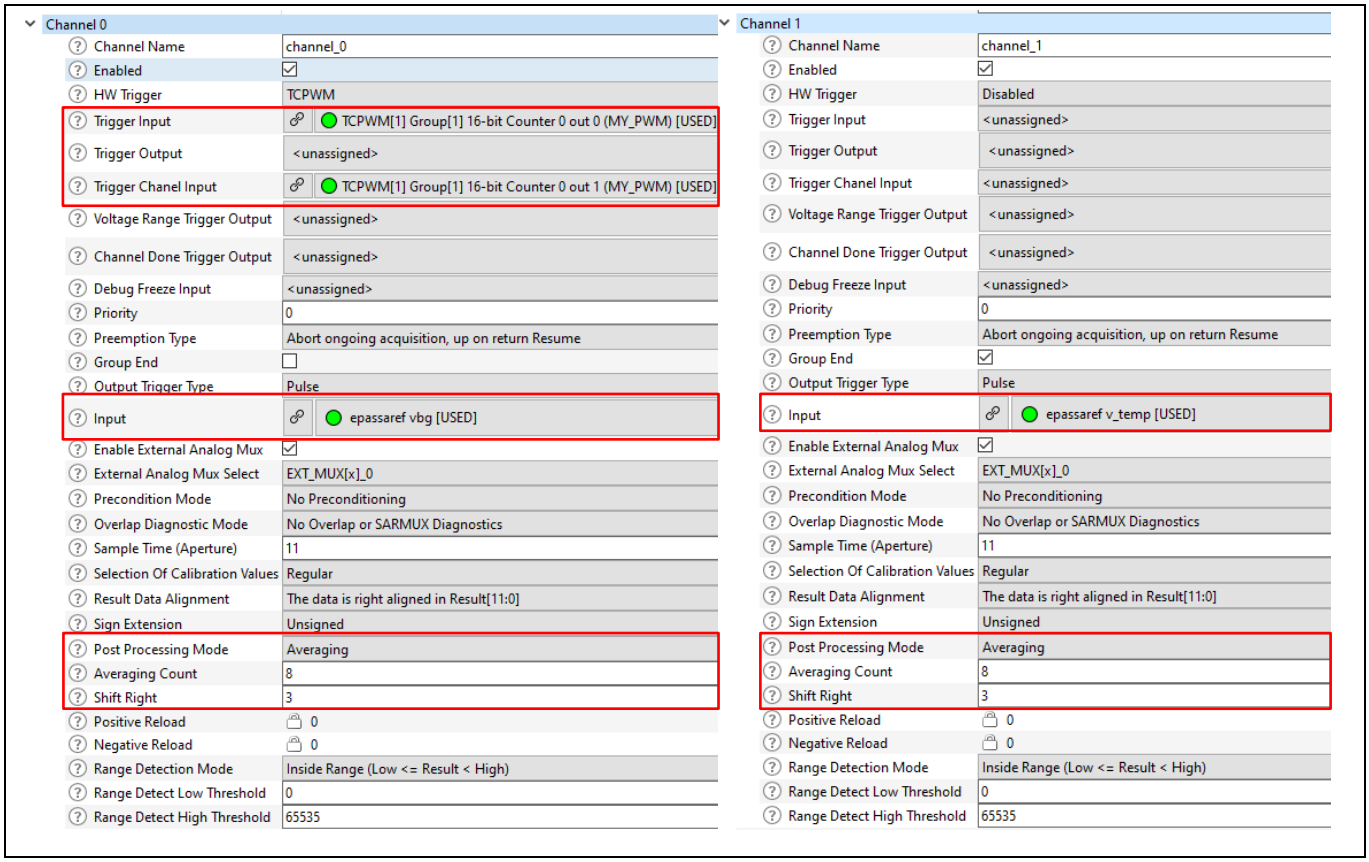

Figure 39 SAR2 channel configuration for temperature in Device Configurator

Temperature sensor

Table 18 Functions for temperature measurement

Functions	Description	Value
`AdcReadChannelData(cy_en_adc_pin_address_t channelAddress)`	Reads the ADC conversion result of the target pin address	CY_ADC_IN_VBG (Y_ADC_PIN_ADDRESS_VBG = 38ul), CY_ADC_IN_TEMP (CY_ADC_PIN_ADDRESS_VTEMP = 39ul)
`AdcConfigureChannel(cy_en_adc_pin_address_t channelAddress)`	Initializes and enable the ADC channel	
`Cy_SAR2_CalculateDieTemperature(cy_stc_adc_temp_ref_t *refValue, cy_stc_adc_temp_raw_t *rawValue)`	Calculates the chip temperature	&tempRefValue, &tempRawValue

10.4 Sample code

See Code Listing 18 and Code Listing 19 for the sample code of temperature measurement.

Code Listing 18 Configuring temperature measurement

```
#define MY_SAR_HW PASS0_SAR0
#define MY_SAR_CH0_HW PASS0_SAR0_CH0
#define MY_SAR_CH1_HW PASS0_SAR0_CH1
#define MY_SAR_channel_0_HW MY_SAR_CH0_HW
#define MY_SAR_channel_1_HW MY_SAR_CH1_HW
#define MY_SAR_CH0_IRQ pass_0_interrupts_sar_0_IRQn
#define MY_SAR_CH1_IRQ pass_0_interrupts_sar_1_IRQn
#define MY_SAR_channel_0_IRQ MY_SAR_CH0_IRQ
#define MY_SAR_channel_1_IRQ MY_SAR_CH1_IRQ
#define MY_SAR_VDDA_RANGE CY_SAR2_VDDA_2_7V_TO_4_5V
#ifndef SARMUX0_CH0_PORT_ADDR
#define SARMUX0_CH0_PORT_ADDR 0
#endif
#ifndef SARMUX0_CH1_PORT_ADDR
#define SARMUX0_CH1_PORT_ADDR 0
#endif
#define MY_SAR_channel_0_IDX (0UL)
#define MY_SAR_channel_1_IDX (1UL)

const cy_stc_sar2_channel_config_t MY_SAR_channel_0_config =
{
.channelHwEnable = true,
```

Temperature sensor

Code Listing 18 Configuring temperature measurement

```
.triggerSelection = CY_SAR2_TRIGGER_TCPWM,
.channelPriority = 0U,
.preenptionType = CY_SAR2_PREEMPTION_ABORT_RESUME,
.doneLevel = CY_SAR2_DONE_LEVEL_PULSE,
.isGroupEnd = false,
.pinAddress = SARMUX0_CH0_PIN_ADDR,
.portAddress = SARMUX0_CH0_PORT_ADDR,
.extMuxEnable = true,
.extMuxSelect = 0U,
.preconditionMode = CY_SAR2_PRECONDITION_MODE_OFF,
.overlapDiagMode = CY_SAR2_OVERLAP_DIAG_MODE_OFF,
.sampleTime = 11U,
.calibrationValueSelect = CY_SAR2_CALIBRATION_VALUE_REGULAR,
.postProcessingMode = CY_SAR2_POST_PROCESSING_MODE_AVG,
.resultAlignment = CY_SAR2_RESULT_ALIGNMENT_RIGHT,
.signExtention = CY_SAR2_SIGN_EXTENTION_UNSIGNED,
.averageCount = 8U,
.rightShift = 3U,
.positiveReload = 0U,
.negativeReload = 0U,
.rangeDetectionMode = CY_SAR2_RANGE_DETECTION_MODE_INSIDE_RANGE,
.rangeDetectionLoThreshold = 0U,
.rangeDetectionHiThreshold = 65535U,
};
const cy_stc_sar2_channel_config_t MY_SAR_channel_1_config =
{
.channelHwEnable = true,
.triggerSelection = CY_SAR2_TRIGGER_OFF,
.channelPriority = 0U,
.preenptionType = CY_SAR2_PREEMPTION_ABORT_RESUME,
.doneLevel = CY_SAR2_DONE_LEVEL_PULSE,
.isGroupEnd = true,
.pinAddress = SARMUX0_CH1_PIN_ADDR,
.portAddress = SARMUX0_CH1_PORT_ADDR,
.extMuxEnable = true,
.extMuxSelect = 0U,
.preconditionMode = CY_SAR2_PRECONDITION_MODE_OFF,
.overlapDiagMode = CY_SAR2_OVERLAP_DIAG_MODE_OFF,
```

Temperature sensor

Code Listing 18 Configuring temperature measurement

```
.sampleTime = 11U,
.calibrationValueSelect = CY_SAR2_CALIBRATION_VALUE_REGULAR,
.postProcessingMode = CY_SAR2_POST_PROCESSING_MODE_AVG,
.resultAlignment = CY_SAR2_RESULT_ALIGNMENT_RIGHT,
.signExtention = CY_SAR2_SIGN_EXTENTION_UNSIGNED,
.averageCount = 8U,
.rightShift = 3U,
.positiveReload = 0U,
.negativeReload = 0U,
.rangeDetectionMode = CY_SAR2_RANGE_DETECTION_MODE_INSIDE_RANGE,
.rangeDetectionLoThreshold = 0U,
.rangeDetectionHiThreshold = 65535U,
};
const cy_stc_sar2_config_t MY_SAR_config =
{
.preconditionTime = 0U,
.powerupTime = 0U,
.enableIdlePowerDown = false,
.msbStretchMode = CY_SAR2_MSB_STRETCH_MODE_1CYCLE,
.enableHalfLsbConv = false,
.sarMuxEnable = true,
.adcEnable = true,
.sarIpEnable = true,
.channelConfig = {(cy_stc_sar2_channel_config_t *)&MY_SAR_channel_0_config,
(cy_stc_sar2_channel_config_t *)&MY_SAR_channel_1_config,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL, NULL,
NULL, NULL, NULL, NULL, NULL, NULL},
   001       };
```

Code Listing 19 Temperature measurement

```
int main(void)
{
    cy_rslt_t result;
    uint16_t resultVBG = 0;
    uint16_t resultTemp = 0;
    float dieTemperature = 0.0;
```

Temperature sensor

Code Listing 19 Temperature measurement

```
    /* 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: ADC Temperature Measure \r\n");
      printf("**************************************************************
******************\r\n");

    /*Analog resource initialize*/
    init_analog_resources();

    /* 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);
       }
```

Temperature sensor

Code Listing 19 Temperature measurement

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

    /* Then start the PWM */
    Cy_TCPWM_TriggerReloadOrIndex_Single(MY_PWM_HW, MY_PWM_NUM);

    for (;;)
    {
       if(sar0_isr_set)
          {
                resultVBG = Cy_SAR2_Channel_GetResult(MY_SAR_HW, 0u, NULL);
             printf("resultVBG = %d \r\n",resultVBG );

             resultTemp = Cy_SAR2_Channel_GetResult(MY_SAR_HW, 1u, NULL);
             printf("resultTemp = %d \r\n",resultTemp );

                       dieTemperature =
Cy_SAR2_CalculateDieTemperature(CY_SAR2_VDDA_2_7V_TO_4_5V, resultTemp,
resultVBG);
                       printf("ADC die Temperature value: %f\r\n",
dieTemperature);
             sar0_isr_set = false;
          }
       Cy_SysLib_Delay(10);
    }
}

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,1u) ==
CY_SAR2_INT_GRP_DONE)
    {
        sar0_isr_set = true;
    }

    /* Clear the interrupts */
    Cy_SAR2_Channel_ClearInterrupt(MY_SAR_HW, 1u ,CY_SAR2_INT_GRP_DONE);
```

Temperature sensor

Code Listing 19 Temperature measurement

```
}

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);
    }
    /* Set ePASS MMIO reference buffer mode for bangap voltage */
      Cy_SAR2_SetReferenceBufferMode(PASS0_EPASS_MMIO,
CY_SAR2_REF_BUF_MODE_ON);

    /*Configure and register SAR0 channel interrupts*/
    Cy_SysInt_Init(&SAR0_IRQ_cfg, sar0_interrupt_handler);
    NVIC_ClearPendingIRQ((IRQn_Type)SAR0_IRQ_cfg.intrSrc);
    NVIC_EnableIRQ((IRQn_Type) NvicMux3_IRQn);

    /*Initialize SAR0 channel 0/1*/
    Cy_SAR2_Channel_Init(MY_SAR_HW, 0u, &MY_SAR_channel_0_config);
    Cy_SAR2_Channel_Init(MY_SAR_HW, 1u, &MY_SAR_channel_1_config);

    /*Set SAR0 channel 1 interrupt mask*/
    Cy_SAR2_Channel_SetInterruptMask(MY_SAR_HW, 1u,CY_SAR2_INT_GRP_DONE);
}
```

11 Glossary

Terms	Description
SAR ADC	Successive approximation registers analog-to-digital converter
SARMUX	Analog input multiplexer
TCPWM	Timer, counter, and pulse width modulator
V_{REFH}	High reference voltage
V_{REFL}	Low reference voltage
V_{BG}	Bandgap reference voltage
V_{BE}	Temperature sensor output

References

Contact Technical Support to obtain XMC7000 family reference documents.

[1] Device datasheet

- XMC7100 series 32-bit Arm® Cortex®-M7 microcontroller datasheet
- XMC7200 series 32-bit Arm® Cortex®-M7 microcontroller datasheet

[2] Device reference manual

- XMC7000 MCU family architecture reference manual
- XMC7100 registers reference manual
- XMC7200 registers reference manual

[3] Application notes

- AN234802 - Secure system configuration in XMC7000 family
- AN234334 - Getting started with XMC7000 MCU on ModusToolbox™
- AN234254 - Multi-core handling guide in XMC7000
- AN234197 - How to use trigger multiplexer in XMC7000 family
- AN234119 - Timer, Counter, and PWM TCPWM usage in XMC7000 family

Revision history

Document version	Date of release	Description of changes
**	2022-06-27	Initial release.
*A	2024-03-19	Updated all the document device configurator pictures. Updated all 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 2024-03-19

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Email: erratum@infineon.com

Document reference

002-34282 Rev. *A

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.