Using a SAR ADC in Traveo II Family

About this document

Associated Part Family

- CYT2 Series
- CYT3 Series
- CYT4 Series

Scope and purpose

AN219755 demonstrates how to configure and use a SAR ADC in Traveo™ II Family MCU with a software trigger, a hardware trigger, group processing, averaging, range detection, pulse detection, diagnosis, and calibration.

Intended audience

This document is intended for anyone who uses the SAR ADC of the Traveo II family.

Table of contents

1 Introduction

This application note describes how to use a SAR ADC for Cypress Traveo II 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 and diagnosis and calibration.

To understand the functionality described and terminology used in this application note, see the SAR ADC chapter of the **Architecture Technical Reference Manual (TRM).**

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 by using a software trigger.

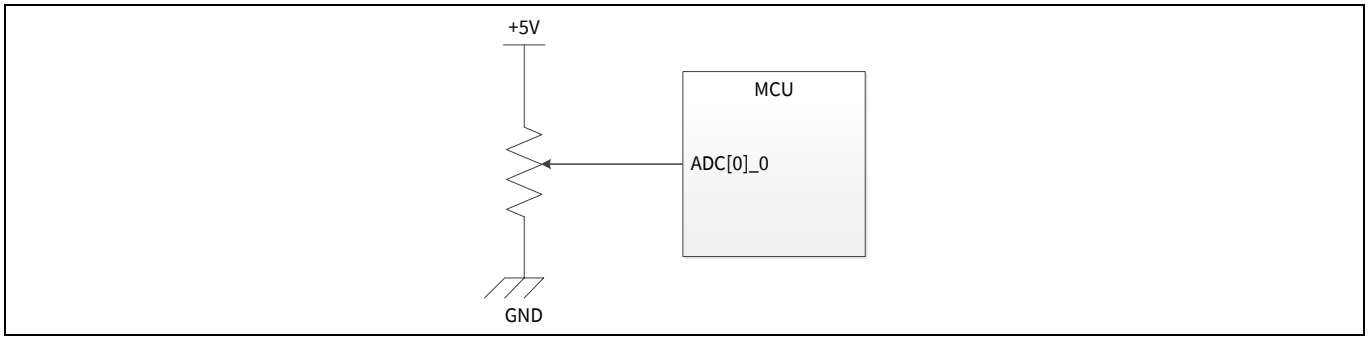


Figure 1 Example of Analog-to-Digital Conversion Connection

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 Settings

This section describes how to configure the ADC based on a use case using the Sample Driver Library (SDL) provided by Cypress. The code snippets in this application note are part of SDL. See **Other References**.

SDL has a configuration part and a driver part. The configuration part configures the parameter values for the desired operation.

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

You can configure the configuration part according to your system.

2.2 ADC Global Settings

The following are the procedures to configure common ADC settings in each channel:

- ADC enable and auto idle power down setting by the SARn_CTL register
- Debug freeze setting by the PASS_PASS_CTL register

Figure 2 shows an example of the ADC global settings.

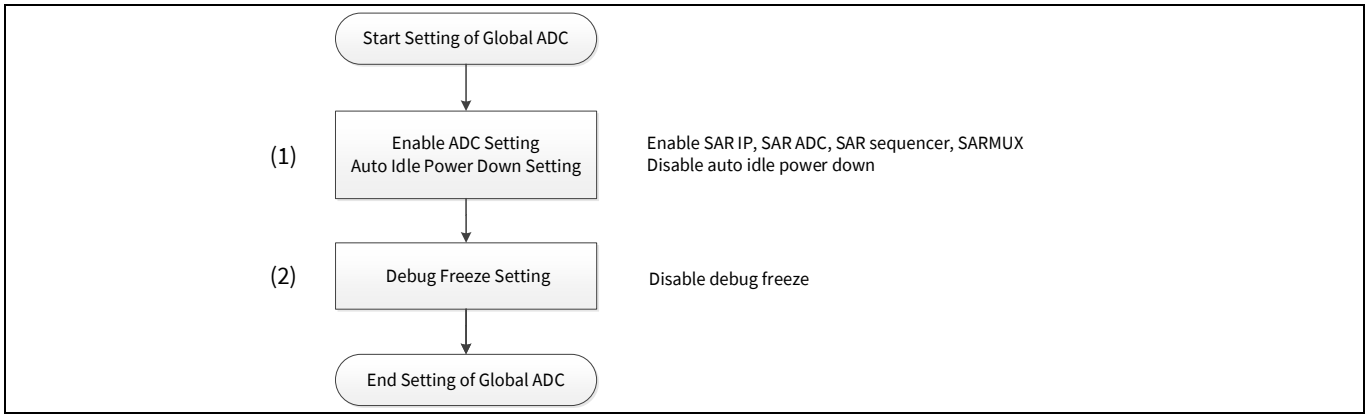


Figure 2 Example of ADC Global Settings

2.2.1 Use Case

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

- ADC Logical Channel: 0
- ADC Operation Frequency: 26.67 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)

2.2.2 Configuration

Table 1 lists the parameters and Table 2 lists the functions of the configuration part of in SDL for ADC global settings.

Table 1 List of ADC Global Settings Parameters

Parameters	Description	Value
BB_POTI_ANALOG_MACRO	Analog macro for the potentiometer on Traveo II Base Board	CY_ADC_POT_MACRO: PASS0_SAR0
CY_ADC_BAD_PARAM	Bad parameter was passed	0x01ul
FreezeConfig.enableFreezeAdc0	Freeze ADC0 in Debug Mode	0ul
FreezeConfig.enableFreezeAdc1	Freeze ADC1 in Debug Mode	0ul
FreezeConfig.enableFreezeAdc2	Freeze ADC2 in Debug Mode	0ul
FreezeConfig.enableFreezeAdc3	Freeze ADC3 in Debug Mode	0ul
adcConfig.preconditionTime	Pre-condition Time	0ul
adcConfig.powerupTime	Power-up Time	0ul
adcConfig.enableIdlePowerDown	Enable Idle Power Down	false
adcConfig.msbStretchMode	MSB Stretch Mode 0: CY_ADC_MSB_STRETCH_MODE_1CYCLE 1: CY_ADC_MSB_STRETCH_MODE_2CYCLE	0ul
adcConfig.enableHalfLsbConv	Enable Half LSB Conversion	0ul
adcConfig.sarMuxEnable	Enable SAR MUX	true
adcConfig.adcEnable	Enable ADC	true
adcConfig.sarIpEnable	Enable SAR IP	true

Software Trigger Procedure

Table 2 List of ADC Global Settings Functions

Functions	Description	Value
`Cy_Adc_Init(PASS SAR,ADC Configure)`	Initialize the ADC module.	PASS SAR = BB_POTI_ANALOG_MACRO ADC Configure = adcConfig

2.2.3 Sample Code

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

- base->unENABLE.stcField.u1CHAN_EN is the PASS0_SAR0_CH0_ENABLE.CHAN_EN setting mentioned in the Registers TRM. Other registers are also described in the same manner.
- Performance improvement measures
- To improve the performance of register setting, the SDL 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.

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

See *cyip_pass.h* under *hdr/rev_x/ip* for more information on the union and structure representation of registers.

Code Listing 1 General Configuration of ADC Global Settings

```
:
#define BB_POTI_ANALOG_MACRO        CY_ADC_POT_MACRO
:
/* Control freeze feature for debugging. */
cy_stc_adc_debug_freeze_config_t FreezeConfig =
{
    /* If true, freeze ADC0 in debug mode. */
    .enableFreezeAdc0 = 0ul,
    .enableFreezeAdc1 = 0ul,
    .enableFreezeAdc2 = 0ul,
    .enableFreezeAdc3 = 0ul,
};
:
int main(void)
{
```

Software Trigger Procedure

Code Listing 1 General Configuration of ADC Global Settings

```
:
    __enable_irq(); /* Enable global interrupts. */

   /* Initialize ADC */
    {
        cy_stc_adc_config_t adcConfig =
        {
            .preconditionTime    = 0ul,
            .powerupTime         = 0ul,
            .enableIdlePowerDown = false,
            .msbStretchMode      = CY_ADC_MSB_STRETCH_MODE_1CYCLE,
            .enableHalfLsbConv   = 0ul,
            .sarMuxEnable        = true,
            .adcEnable           = true,
            .sarIpEnable         = true,
        };
        Cy_Adc_Init(BB_POTI_ANALOG_MACRO, &adcConfig);
        Cy_Adc_SetDebugFreezeMode(PASS0_EPASS_MMIO,&FreezeConfig);
:
    }
:
    for(;;)
    {
    }
}
```

ADC Configuration.

ADC Initialize. See **Code Listing 2**.

Set Debug freeze mode. See **Code Listing 3**.

Code Listing 2 Cy_Adc_Init() Function

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

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

(1) Enable DC Setting. Auto Idle Power Down Setting.

Software Trigger Procedure

Code Listing 2 Cy_Adc_Init() Function

```
    }
    else
    {
        ret = CY_ADC_BAD_PARAM;
    }

    return ret;
}
```

Code Listing 3 Cy_Adc_SetDebugFreezeMode () Function

```
cy_en_adc_status_t Cy_Adc_SetDebugFreezeMode(volatile stc_PASS_EPASS_MMIO_t * base, const
cy_stc_adc_debug_freeze_config_t * config)
{
    cy_en_adc_status_t ret  = CY_ADC_SUCCESS;
    uint32_t           temp = 0ul;
    if (NULL != config)
    {
        temp |= (config->enableFreezeAdc0) ? 1ul : 0ul;
        temp |= (config->enableFreezeAdc1) ? 2ul : 0ul;
        temp |= (config->enableFreezeAdc2) ? 4ul : 0ul;
        temp |= (config->enableFreezeAdc3) ? 8ul : 0ul;
        base->unPASS_CTL.stcField.u4DBG_FREEZE_EN = temp;
    }
    else
    {
        ret = CY_ADC_BAD_PARAM;
    }
    return ret;
}
```

(2) Debug Freeze disabled by default.

2.3 Logical Channel Settings with Software Trigger

There is one A/D conversion result register per logical channel. A logical channel can be assigned as any analog input (ADC[n]_i) by setting the SARn_CHx_SAMPLE_CTL register.

Figure 3 shows the example of logical channel setting 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 **Related Documents**.

Software Trigger Procedure

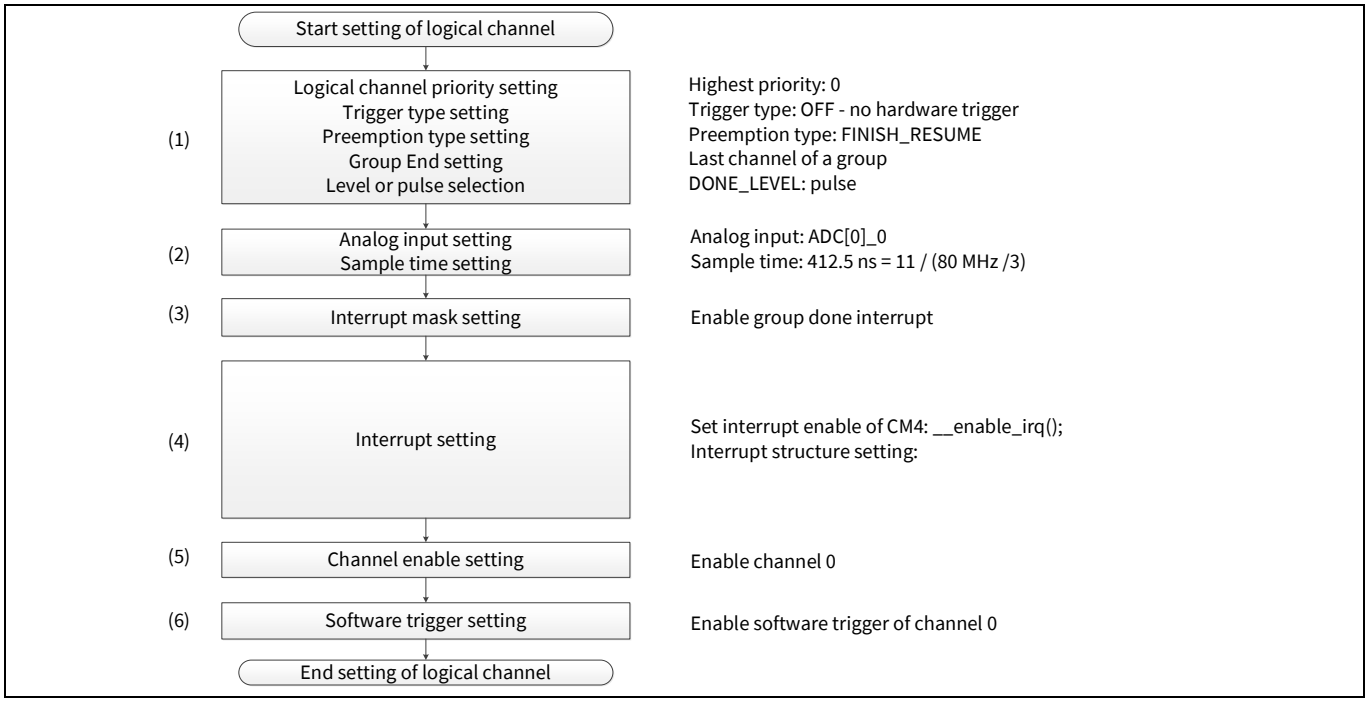


Figure 3 Example of Logical Channel Setting with Software Trigger

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 Logical Channel Setting with Software Trigger.

- Analog Input: ADC[0]_0
- Sample Time: 412.5 ns = 11 / (80 MHz /3)
- 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
- 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

Software Trigger Procedure

- Mask Channel: Not Range/Not Pulse/Not Overflow

2.3.2 Configuration

Table 3 lists the parameters and Table 4 lists the functions of the configuration part of in SDL for Logical Channel Settings with Software Trigger settings.

Table 3 List of Logical Channel Settings with Software Trigger Settings Parameters

Parameters	Description	Value
BB_POTI_ANALOG_MACRO	Analog macro for the potentiometer on Traveo II Base Board	CY_ADC_POT_MACRO: PASS0_SAR0
BB_POTI_ANALOG_INPUT_NO	Analog Input Number for the potentiometer on Traveo II Base Board	CH0 (CH[ADC_LOGICAL_CHANNEL])
adcConfig.msbStretchMode	MSB Stretch Mode	0ul (See Table 1)
adcChannelConfig.triggerSelection	Trigger OFF 0: CY_ADC_TRIGGER_OFF 1: CY_ADC_TRIGGER_TCPWM 2: CY_ADC_TRIGGER_GENERIC0 3: CY_ADC_TRIGGER_GENERIC1 4: CY_ADC_TRIGGER_GENERIC2 5: CY_ADC_TRIGGER_GENERIC3 6: CY_ADC_TRIGGER_GENERIC4 7: CY_ADC_TRIGGER_CONTINUOUS	0ul
adcChannelConfig.channelPriority	Channel Priority	0ul
adcChannelConfig.preenptionType	Pre-emption Type 0: CY_ADC_PREEMPTION_ABORT_CANCEL 1: CY_ADC_PREEMPTION_ABORT_RESTART 2: CY_ADC_PREEMPTION_ABORT_RESUME 3: CY_ADC_PREEMPTION_FINISH_RESUME	3ul
adcChannelConfig.isGroupEnd	Is Group End?	true
adcChannelConfig.doneLevel	Done Level 0: CY_ADC_DONE_LEVEL_PULSE 1: CY_ADC_DONE_LEVEL_LEVEL	0ul
adcChannelConfig.pinAddress	Pin Address	BB_POTI_ANALOG_INPUT_NO
adcChannelConfig.portAddress	Port Address	0ul

Software Trigger Procedure

Parameters	Description	Value
	0: CY_ADC_PORT_ADDRESS_SARMUX0 1: CY_ADC_PORT_ADDRESS_SARMUX1 2: CY_ADC_PORT_ADDRESS_SARMUX2 3: CY_ADC_PORT_ADDRESS_SARMUX3	
adcChannelConfig.extMuxSelect	External MUX Select	0ul
adcChannelConfig.extMuxEnable	Enable External MUX	true
adcChannelConfig.preconditionMode	Pre-condition Mode 0: CY_ADC_PRECONDITION_MODE_OFF 1: CY_ADC_PRECONDITION_MODE_VREFL 2: CY_ADC_PRECONDITION_MODE_VREFH 3: CY_ADC_PRECONDITION_MODE_DIAG	0ul
adcChannelConfig.overlapDiagMode	Overlap Diagnostics Mode 0: CY_ADC_OVERLAP_DIAG_MODE_OFF 1: CY_ADC_OVERLAP_DIAG_MODE_HALF 2: CY_ADC_OVERLAP_DIAG_MODE_FULL 3: CY_ADC_OVERLAP_DIAG_MODE_MUX_DIAG	0ul
adcChannelConfig.sampleTime	Sample Time	samplingCycle (Calculated Value)
adcChannelConfig.calibrationValueSelect	Calibration Value Select 0: CY_ADC_CALIBRATION_VALUE_REGULAR 1: CY_ADC_CALIBRATION_VALUE_ALTERNATE	0ul
adcChannelConfig.postProcessingMode	Post Processing Mode 0: CY_ADC_POST_PROCESSING_MODE_NONE 1: CY_ADC_POST_PROCESSING_MODE_AVG 2: CY_ADC_POST_PROCESSING_MODE_AVG_RANGE 3: CY_ADC_POST_PROCESSING_MODE_RANGE 4: CY_ADC_POST_PROCESSING_MODE_RANGE_PULSE	0ul
adcChannelConfig.resultAlignment	Result Alignment 0: CY_ADC_RESULT_ALIGNMENT_RIGHT 1: CY_ADC_RESULT_ALIGNMENT_LEFT	0ul
adcChannelConfig.signExtention	Sign Extension 0: CY_ADC_SIGN_EXTENTION_UNSIGNED 1: CY_ADC_SIGN_EXTENTION_SIGNED	0ul
adcChannelConfig.averageCount	Average Count	0ul
adcChannelConfig.rightShift	Right Shift	0ul
adcChannelConfig.rangeDetectionMode	Range Detection Mode 0: CY_ADC_RANGE_DETECTION_MODE_BELOW_LO	1ul

Software Trigger Procedure

Parameters	Description	Value
	1: CY_ADC_RANGE_DETECTION_MODE_INSIDE_RANGE 2: CY_ADC_RANGE_DETECTION_MODE_ABOVE_HI 3: CY_ADC_RANGE_DETECTION_MODE_OUTSIDE_RANGE	
`adcChannelConfig.rangeDetectionLoThreshold`	Range Detection Low Threshold	0x0000ul
`adcChannelConfig.rangeDetectionHiThreshold`	Range Detection High Threshold	0x0FFFul
`adcChannelConfig.mask.grpDone`	Mask Group Done	true
`adcChannelConfig.mask.grpCancelled`	Mask Group Cancelled	false
`adcChannelConfig.mask.grpOverflow`	Mask Group Overflow	false
`adcChannelConfig.mask.chRange`	Mask Channel Range	false
`adcChannelConfig.mask.chPulse`	Mask Channel Pulse	false
`adcChannelConfig.mask.chOverflow`	Mask Channel Overflow	false

Table 4 List of Logical Channel Settings with Software Trigger Settings Functions

Functions	Description	Value
`Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)`	Initialize the ADC channel	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig
`Cy_Adc_Channel_SetInterruptMask(PASS SARchannel, INTR Source)`	ADC Channel Set Interrupt Mask	PASS SARchannel = base INTR Source = mask
`Cy_SysInt_InitIRQ(Config)`	Initialize the referenced system interrupt by setting the interrupt vector	Config = irq_cfg
`Cy_Adc_Channel_Enable(SARchannel)`	Enable the corresponding channel	SARchannel = PASS_SAR_CH
`Cy_Adc_Channel_SoftwareTrigger(PASS SARchannel)`	Issue a software start trigger	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

2.3.3 Sample Code

See **Code Listing 4** to **Code Listing 9** for sample code for initial configuration of logical channel settings with software trigger settings.

Code Listing 4 General Configuration of ADC Global Settings

```
:
#define BB_POTI_ANALOG_MACRO        CY_ADC_POT_MACRO
#define BB_POTI_ANALOG_INPUT_NO     ((cy_en_adc_pin_address_t)CY_ADC_POT_IN_NO)
:
int main(void)
{
:
    __enable_irq(); /* Enable global interrupts. */

   /* Initialize ADC */
    {
:
        cy_stc_adc_config_t adcConfig =
        {
            .preconditionTime    = 0ul,
            .powerupTime         = 0ul,
            .enableIdlePowerDown = false,
            .msbStretchMode      = CY_ADC_MSB_STRETCH_MODE_1CYCLE,
            .enableHalfLsbConv   = 0ul,
            .sarMuxEnable        = true,
            .adcEnable           = true,
            .sarIpEnable         = true,
        };
        cy_stc_adc_channel_config_t adcChannelConfig =
        {
            .triggerSelection          = CY_ADC_TRIGGER_OFF,
            .channelPriority           = 0ul,
            .preenptionType            = CY_ADC_PREEMPTION_FINISH_RESUME,
            .isGroupEnd                = true,
            .doneLevel                 = CY_ADC_DONE_LEVEL_PULSE,
            .pinAddress                = BB_POTI_ANALOG_INPUT_NO,
            .portAddress               = CY_ADC_PORT_ADDRESS_SARMUX0,
            .extMuxSelect              = 0ul,
            .extMuxEnable              = true,
            .preconditionMode          = CY_ADC_PRECONDITION_MODE_OFF,
            .overlapDiagMode           = CY_ADC_OVERLAP_DIAG_MODE_OFF,
            .sampleTime                = 0ul,
            .calibrationValueSelect    = CY_ADC_CALIBRATION_VALUE_REGULAR,
            .postProcessingMode        = CY_ADC_POST_PROCESSING_MODE_NONE,
            .resultAlignment           = CY_ADC_RESULT_ALIGNMENT_RIGHT,
            .signExtention             = CY_ADC_SIGN_EXTENTION_UNSIGNED,
            .averageCount              = 0ul,
            .rightShift                = 0ul,
```

ADC Configuration.

(1) Logical channel priority setting
Trigger type setting
Preemption type setting
Group End setting
Level or pulse selection

(2) Analog input setting
Sample time setting

Software Trigger Procedure

Code Listing 4 General Configuration of ADC Global Settings

```
            .rangeDetectionMode       = CY_ADC_RANGE_DETECTION_MODE_INSIDE_RANGE,
            .rangeDetectionLoThreshold = 0x0000ul,
            .rangeDetectionHiThreshold = 0x0FFFul,
            .mask.grpDone             = true,
            .mask.grpCancelled        = false,
            .mask.grpOverflow         = false,
            .mask.chRange             = false,
            .mask.chPulse             = false,
            .mask.chOverflow          = false,
        };

        Cy_Adc_Init(BB_POTI_ANALOG_MACRO, &adcConfig);
        Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &adcChannelConfig);
    }
:
        Cy_SysInt_InitIRQ(&irq_cfg);
:
    /* Enable ADC ch. */
    Cy_Adc_Channel_Enable(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);
    /* Issue SW trigger */
    Cy_Adc_Channel_SoftwareTrigger(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);
    for(;;)
    {
    }
}
```

Continuation of (2)

(3) Interrupt mask setting

ADC Initialize. See **Code Listing 2**.

Initialize ADC Channel. See **Code Listing 5**.

Initialize Interrupt Request. See **Code Listing 7**.

Enable ADC Channel. See **Code Listing 8**.

Software Trigger Setting. See **Code Listing 9**.

Code Listing 5 Cy_Adc_Channel_Init() Function

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

    if (NULL != config)
    {
:
        /* Clear whole interrupt flags */
        unIntr.stcField.u1CH_OVERFLOW       = 1ul;
        unIntr.stcField.u1CH_PULSE          = 1ul;
        unIntr.stcField.u1CH_RANGE          = 1ul;
        unIntr.stcField.u1GRP_CANCELLED     = 1ul;
        unIntr.stcField.u1GRP_DONE          = 1ul;
```

Software Trigger Procedure

Code Listing 5 Cy_Adc_Channel_Init() Function

```
        unIntr.stcField.u1GRP_OVERFLOW      = 1ul;
        base->unINTR.u32Register            = unIntr.u32Register;

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

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

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

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

        Cy_Adc_Channel_SetInterruptMask(base, &config->mask);
    }
    else
    {
        ret = CY_ADC_BAD_PARAM;
    }
    return ret;
}
```

ADC Channel Set Interrupt Mask.
See **Code Listing 6**.

Code Listing 6 Cy_Adc_Channel_SetInterruptMask() Function

```
cy_en_adc_status_t Cy_Adc_Channel_SetInterruptMask(volatile stc_PASS_SAR_CH_t * base, const
cy_stc_adc_interrupt_source_t * mask)
{
    cy_en_adc_status_t ret = CY_ADC_SUCCESS;
```

Software Trigger Procedure

Code Listing 6 Cy_Adc_Channel_SetInterruptMask() Function

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

Code Listing 7 Cy_SysInt_InitIRQ()Function

```
cy_en_sysint_status_t Cy_SysInt_InitIRQ(const cy_stc_sysint_irq_t* config)
{
    cy_en_sysint_status_t status = CY_SYSINT_SUCCESS;

    #if (CY_CPU_CORTEX_M0P)
        un_CPUSS_CM0_SYSTEM_INT_CTL_t unIntCtl = { 0ul };
    #else
        #if defined (tviibe512k) || defined (tviibe1m) || defined (tviibe2m)|| defined (tviibe4m)
            un_CPUSS_CM4_SYSTEM_INT_CTL_t unIntCtl = { 0ul };
        #elif defined (tviibh4m) || defined (tviibh8m) || defined (tviic2d6m) || defined (tviic2d4m) ||
defined (tviic2d6mddr)
            un_CPUSS_CM7_0_SYSTEM_INT_CTL_t unIntCtl0 = { 0ul };
            un_CPUSS_CM7_1_SYSTEM_INT_CTL_t unIntCtl1 = { 0ul };
        #endif
    #endif
    if(NULL != config)
    {
        #if (CY_CPU_CORTEX_M0P) //rmkn u3CM0_CPU_INT_IDX->u3CPU_INT_IDX
            #if defined (tviibe512k) || defined (tviibe1m) || defined (tviibe2m)|| defined (tviibe4m)
                unIntCtl.stcField.u3CPU_INT_IDX = (uint8_t)config->intIdx;
            #elif defined (tviibh4m) || defined (tviibh8m) || defined (tviic2d6m) || defined
(tviic2d4m) || defined (tviic2d6mddr)
                unIntCtl.stcField.u3CM0_CPU_INT_IDX = (uint8_t)config->intIdx;
            #endif
            unIntCtl.stcField.u1CPU_INT_VALID = config->isEnabled ? 1ul : 0ul;
            CPUSS->unCM0_SYSTEM_INT_CTL[config->sysIntSrc].u32Register = unIntCtl.u32Register;
        #else
```

Software Trigger Procedure

Code Listing 7 Cy_SysInt_InitIRQ()Function

```
            #if defined (tviibe512k) || defined (tviibe1m) || defined (tviibe2m)|| defined (tviibe4m)
                unIntCtl.stcField.u3CPU_INT_IDX = (uint8_t)config->intIdx;
                unIntCtl.stcField.u1CPU_INT_VALID = config->isEnabled ? 1ul : 0ul;
                CPUSS->unCM4_SYSTEM_INT_CTL[config->sysIntSrc].u32Register = unIntCtl.u32Register;
            #elif defined (tviibh4m) || defined (tviibh8m) || defined (tviic2d6m) || defined
(tviic2d4m) || defined (tviic2d6mddr)
                if(CPUSS->unIDENTITY.stcField.u4MS == CPUSS_MS_ID_CM7_0)
                {
                    unIntCtl0.stcField.u4CPU_INT_IDX = (uint8_t)config->
                    unIntCtl0.stcField.u1CPU_INT_VALID = config->isEnabl
                    CPUSS->unCM7_0_SYSTEM_INT_CTL[config->sysIntSrc].u32
unIntCtl0.u32Register;
                }
                else // should be CPUSS_MS_ID_CM7_1
                {
                    unIntCtl1.stcField.u4CPU_INT_IDX = (uint8_t)config->intIdx;
                    unIntCtl1.stcField.u1CPU_INT_VALID = config->isEnabled ? 1ul : 0ul;
                    CPUSS->unCM7_1_SYSTEM_INT_CTL[config->sysIntSrc].u32Register =
unIntCtl1.u32Register;
                }
            #endif
        #endif
    }
    else
    {
        status = CY_SYSINT_BAD_PARAM;
    }

    return(status);
}
```

(4) Set Interrupt Enable of CM4, Interrupt structure setting, Priority setting,
Clear pending status and Enable IRQ.

Code Listing 8 Cy_Adc_Channel_Enable() Function

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

(5) Enable Channel 0.

Code Listing 9 Cy_Adc_Channel_SoftwareTrigger() Function

```
void Cy_Adc_Channel_SoftwareTrigger(volatile stc_PASS_SAR_CH_t * base)
{
    base->unTR_CMD.stcField.u1START = 1ul;
}
```

(6) Enable Software Trigger.

2.4 A/D Conversion ISR with Software Trigger

Figure 4 shows the example of ADC ISR with a software trigger. For details on CPU interrupt handling, see the Architecture TRM mentioned in **Related Documents**.

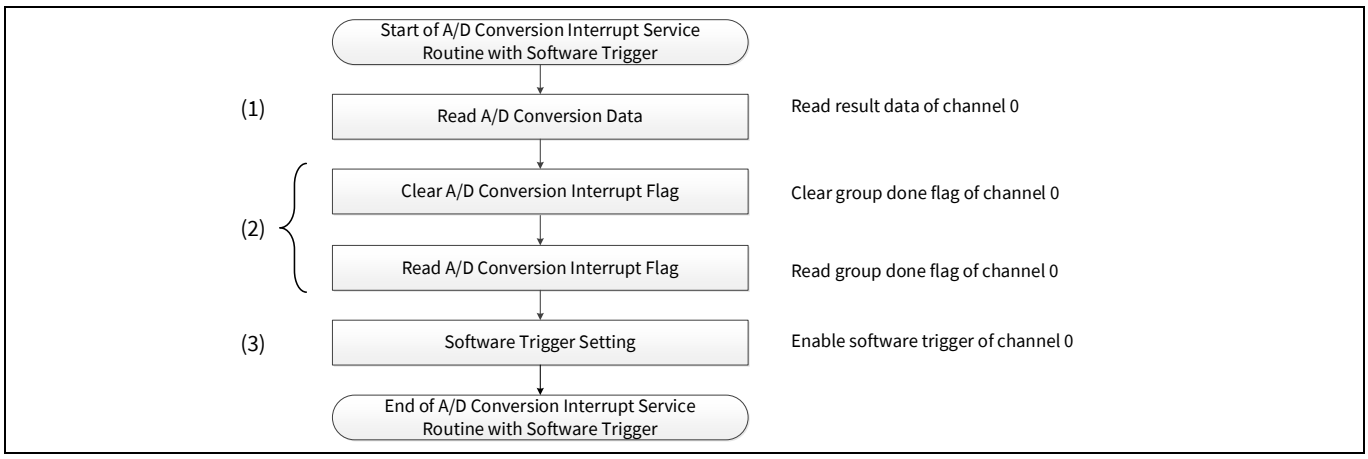


Figure 4 Example of ADC ISR with Software Trigger

2.4.1 Use Case

See Section **2.3.1**.

2.4.2 Configuration

Table 5 lists the parameters and **Table 6** lists the functions of the configuration part of in SDL for ADC Global settings.

Table 5 List of ADC ISR with Software Trigger Settings Parameters

Parameters	Description	Value
`BB_POTI_ANALOG_MACRO`	Analog macro for the potentiometer on Traveo II Base Board	CY_ADC_POT_MACRO: PASS0_SAR0
`resultBuff`	Conversion Result Buffer	- (Calculated Value)
`statusBuff`	Status Result Buffer	- (Calculated Value)
`resultIdx`	Index Result	- (Calculated Value)
`intrSource`	Interrupt Source	- (Calculated Value)

Table 6 List of ADC ISR with Software Trigger Settings Functions

Functions	Description	Value
`Cy_Adc_Channel_GetResult(SAR Channel,Status)`	Get the conversion result and status	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Status = resultBuff[resultIdx]
`Cy_Adc_Channel_ClearInterruptStatus(SAR Channel,Source)`	Clear the corresponding channel interrupt status.	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],

Software Trigger Procedure

Functions	Description	Value
		Source = intrSource
`Cy_Adc_Channel_SoftwareTrigger(SAR Channel)`	Issue the software start trigger	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

2.4.3 Sample Code

See **Code Listing 10** to **Code Listing 12** for sample code for initial configuration of ADC ISR with Software Trigger Settings.

Code Listing 10 ADC ISR with Software Trigger Settings

```
#define BB_POTI_ANALOG_MACRO        CY_ADC_POT_MACRO
:
uint16_t               resultBuff[16];
cy_stc_adc_ch_status_t statusBuff[16];
uint8_t                resultIdx;
:

void AdcIntHandler(void)
{
         cy_stc_adc_interrupt_source_t intrSource = { false };
:
    {
        /* Get the result(s) */
        Cy_Adc_Channel_GetResult(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&resultBuff[resultIdx], &statusBuff[resultIdx]);

        /* Display ADC result */
        printf("\rADC result = %04d          ", resultBuff[resultIdx]);

        /* Increment result idx */
        resultIdx = (resultIdx + 1) % (sizeof(resultBuff) / sizeof(resultBuff[0]));

        /* Clear interrupt source */
        Cy_Adc_Channel_ClearInterruptStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&intrSource);

        /* Trigger next conversion */
        Cy_Adc_Channel_SoftwareTrigger(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);
    }
    else
    {
        // Unexpected interrupt
        CY_ASSERT(false);
    }
}
```

(1) Read A/D conversion data. See **Code Listing 11**.

(2) Clear and read A/D conversion flag. See **Code Listing 12**.

(3) Software Trigger Setting. See **Code Listing 9**.

Code Listing 11 Cy_Adc_Channel_GetResult() Function

```
Cy_Adc_Channel_GetResult(const volatile stc_PASS_SAR_CH_t * base, uint16_t * result,
cy_stc_adc_ch_status_t * status)
{
    cy_en_adc_status_t ret = CY_ADC_SUCCESS;
    un_PASS_SAR_CH_RESULT_t value;

    if ((NULL != result) && (NULL != status))
    {
        value.u32Register = base->unRESULT.u32Register;
        *result           = value.stcField.u16RESULT;
        status->aboveHi   = (value.stcField.u1ABOVE_HI_MIR   != 0ul) ? true : false;
        status->pulseIntr = (value.stcField.u1PULSE_INTR_MIR != 0ul) ? true : false;
        status->rangeIntr = (value.stcField.u1RANGE_INTR_MIR != 0ul) ? true : false;
        status->valid     = (value.stcField.u1VALID_MIR      != 0ul) ? true : false;
    }
    else
    {
        ret = CY_ADC_BAD_PARAM;
    }
    return ret;
}
```

Code Listing 12 Cy_Adc_Channel_ClearInterruptStatus() Function

```
cy_en_adc_status_t Cy_Adc_Channel_ClearInterruptStatus(volatile stc_PASS_SAR_CH_t * base, const
cy_stc_adc_interrupt_source_t * source)
{
    cy_en_adc_status_t ret = CY_ADC_SUCCESS;
    un_PASS_SAR_CH_INTR_t unIntr = { 0ul };
    if (NULL != source)
    {
        unIntr.stcField.u1CH_OVERFLOW   = source->chOverflow   ? 1ul : 0ul;
        unIntr.stcField.u1CH_PULSE      = source->chPulse      ? 1ul : 0ul;
        unIntr.stcField.u1CH_RANGE      = source->chRange      ? 1ul : 0ul;
        unIntr.stcField.u1GRP_CANCELLED = source->grpCancelled ? 1ul : 0ul;
        unIntr.stcField.u1GRP_DONE      = source->grpDone      ? 1ul : 0ul;
        unIntr.stcField.u1GRP_OVERFLOW  = source->grpOverflow  ? 1ul : 0ul;
        base->unINTR.u32Register        = unIntr.u32Register;
    }
    else
    {
        ret = CY_ADC_BAD_PARAM;
    }
    return ret;
}
```

3 Hardware Trigger Procedure

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

This section explains the example application that converts voltage values given to ADC[0]_0 pin of the MCU to digital values. The analog-to-digital conversion is repeated at regular intervals by 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 Settings**.

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

3.1 Logical Channel Setting with Hardware Trigger

Figure 5 shows the example of logical channel setting with a hardware trigger in CYT2 series. 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 **Related Documents**.

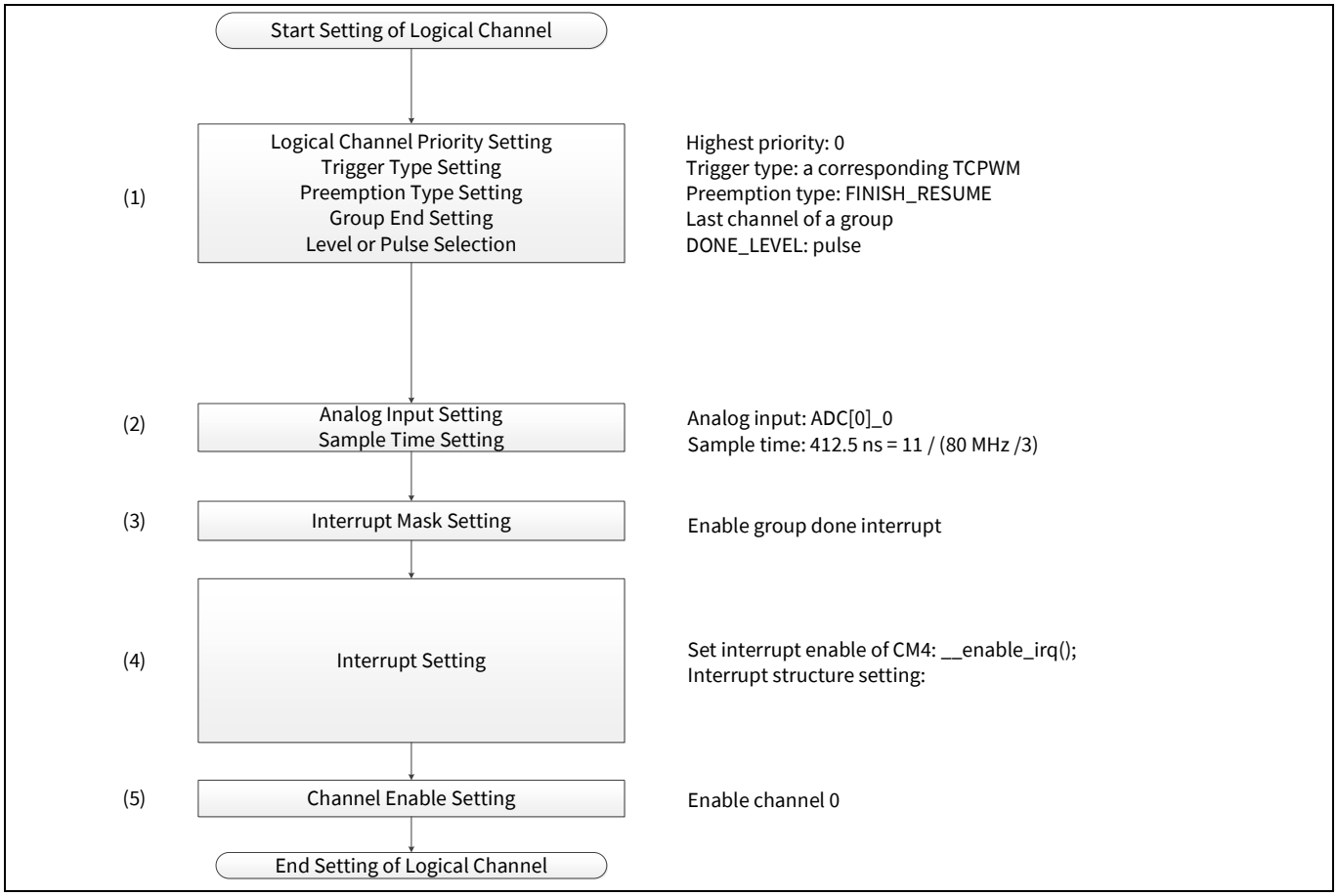


Figure 5 Example of Logical Channel Setting with Hardware Trigger in CYT2 Series

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 Logical Channel Setting with Hardware Trigger.

- Analog Input: ADC[0]_0
- Sample Time: 412.5 ns = 11 / (80 MHz /3)
- 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

3.1.2 Configuration

Table 7 lists the parameters and **Table 8** lists the functions of the configuration part of in SDL for Logical Channel Setting with Hardware Trigger in CYT2 Series settings.

Table 7 List of Logical Channel Setting with Hardware Trigger in CYT2 Series Settings Parameters

Parameters	Description	Value
BB_POTI_ANALOG_MACRO	Analog macro for the potentiometer on Traveo II Base Board	CY_ADC_POT_MACRO: PASS0_SAR0
CY_GPIO_DM_STRONG_IN_OFF	Strong Drive, Input buffer off	0x06ul
pin_cfg1.outVal	Pin output state	0ul
pin_cfg1.driveMode	Drive Mode	CY_GPIO_DM_STRONG_IN_OFF
pin_cfg1.hsiom	High Speed IO Matrix Selection	TCPWMx_LINEx_MUX
pin_cfg1.intEdge	Interrupt edge type	0ul
pin_cfg1.intMask	Interrupt enable mask	0ul
pin_cfg1.vtrip	Input buffer voltage trip type	0ul
pin_cfg1.slewRate	Output buffer slew rate	0ul
pin_cfg1.driveSel	Drive strength	0ul
CY_GPIO_DM_ANALOG	Analog High-Z, Input buffer off	0x00ul
adcPinConfig.outVal	Pin output state	0ul

Hardware Trigger Procedure

Parameters	Description	Value
adcPinConfig.driveMode	Drive Mode	CY_GPIO_DM_ANALOG
adcPinConfig.hsiom	High Speed IO Matrix Selection	P6_0_GPIO
adcPinConfig.intEdge	Interrupt edge type	0ul
adcPinConfig.intMask	Interrupt enable mask	0ul
adcPinConfig.vtrip	Input buffer voltage trip type	0ul
adcPinConfig.slewRate	Output buffer slew rate	0ul
adcPinConfig.driveSel	Drive strength	0ul
adcConfig.preconditionTime	Pre-condition time	0ul
adcConfig.powerupTime	Power-up Time	0ul
adcConfig.enableIdlePowerDown	Enable Idle Power Down	false
adcConfig.msbStretchMode	MSB Stretch Mode	0ul (See **Table 1**)
adcConfig.enableHalfLsbConv	Enable Half LSB Conversion	0ul
adcConfig.sarMuxEnable	Enable SAR MUX	true
adcConfig.adcEnable	Enable ADC	true
adcConfig.sarIpEnable	Enable SAR IP	true
adcChannelConfig.triggerSelection	Trigger Selection	1ul (See **Table 3**)
adcChannelConfig.channelPriority	Channel Priority	0ul
adcChannelConfig.preenptionType	Pre-emption Type	3ul (See **Table 3**)
adcChannelConfig.isGroupEnd	Is Group End?	true
adcChannelConfig.doneLevel	Done Level	0ul (See **Table 3**)
adcChannelConfig.pinAddress	Pin Address	BB_POTI_ANALOG_INPUT_NO
adcChannelConfig.portAddress	Port Address	0ul (See **Table 3**)
adcChannelConfig.extMuxSelect	External MUX Select	0ul
adcChannelConfig.extMuxEnable	Enable External MUX	true
adcChannelConfig.preconditionMode	Pre-condition Mode	0ul (See **Table 3**)
adcChannelConfig.overlapDiagMode	Overlap Diagnostics Mode	0ul (See **Table 3**)
adcChannelConfig.sampleTime	Sample Time	0ul
adcChannelConfig.calibrationValueSelect	Calibration Value Select	0ul (See **Table 3**)
adcChannelConfig.postProcessingMode	Post Processing Mode	0ul (See **Table 3**)
adcChannelConfig.resultAlignment	Result Alignment	0ul (See **Table 3**)
adcChannelConfig.signExtention	Sign Extension	0ul (See **Table 3**)
adcChannelConfig.averageCount	Average Count	0ul
adcChannelConfig.rightShift	Right Shift	0ul
adcChannelConfig.rangeDetectionMode	Range Detection Mode	1ul (See **Table 3**)
adcChannelConfig.rangeDetectionLoThreshold	Range Detection Low Threshold	0x0000ul
adcChannelConfig.rangeDetectionHiThreshold	Range Detection High Threshold	0x0FFFul

Hardware Trigger Procedure

Parameters	Description	Value
`adcChannelConfig.mask.grpDone`	Mask Group Done	true
`adcChannelConfig.mask.grpCancelled`	Mask Group Cancelled	false
`adcChannelConfig.mask.grpOverflow`	Mask Group Overflow	false
`adcChannelConfig.mask.chRange`	Mask Channel Range	false
`adcChannelConfig.mask.chPulse`	Mask Channel Pulse	false
`adcChannelConfig.mask.chOverflow`	Mask Channel Overflow	false

Table 8 List of Logical Channel Setting with Hardware Trigger in CYT2 Series Settings Functions

Functions	Description	Value
`Cy_Adc_Channel_GetInterruptMaskedStatus(PASS SARchannel,INTR Source)`	Return the interrupt status	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] INTR Source = &intrSource
`Cy_SysInt_SetSystemIrqVector(Source,INTR handler)`	Change the User ISR vector for the system interrupt	Source = sysIntSrc INTR handler = AdcIntHandler
`Cy_GPIO_Pin_Init(Port Number,Pin Number,gpio_pin_config)`	Initialize all pin configuration setting for the pin	Port Number = TCPWMx_LINEx_PORT Pin Number = TCPWMx_LINEx_PIN gpio_pin_config = TCPWMx_LINEx_PIN, &pin_cfg1
`Cy_Tcpwm_Pwm_Init(Group Counter, TCPWM PWM config)`	Initialize the TCPWM for PWM operation	Group Counter = TCPWMx_GRPx_CNTx_PWM TCPWM PWM config = &MyPWM_config
`Cy_Tcpwm_Pwm_Enable(Group Counter)`	De-initialize the TCPWM block	Group Counter = TCPWMx_GRPx_CNTx_PWM
`Cy_Tcpwm_TriggerStart(Group Counter)`	Trigger a software start on selected TCPWMs	Group Counter = TCPWMx_GRPx_CNTx_PWM
`Cy_TrigMux_Connect1To1(Trigger MUX,Invert,Trigger Type,Debug Freeze Enable)`	Connect an input trigger source and output trigger	Trigger MUX = TRIG_IN_1TO1_1_TCPWM_TO_PASS_CH_TR0 Invert = 0ul Trigger Type = TRIGGER_TYPE_PASS_TR_SAR_CH_IN__EDGE Debug Freeze Enable = 0ul

3.1.3 Sample Code

See **Code Listing 13** to **Code Listing 20** for sample code for initial configuration of logical channel setting with hardware trigger in CYT2 series settings.

Hardware Trigger Procedure

Code Listing 13 Logical Channel Setting with Hardware Trigger in CYT2 Series Settings

```
#define BB_POTI_ANALOG_MACRO        PASS0_SAR0
:
#define DIV_ROUND_UP(a,b) (((a) + (b)/2) / (b))

/* A/D value result buff */
uint8_t                resultIdx;
uint16_t               resultBuff[16];
/* A/D Status */
cy_stc_adc_ch_status_t statusBuff[16];

/* PWM CONFIGURATION */

/* PWM Mode Configuration def */
#define TCPWMx_GRPx_CNTx_PWM            TCPWM0_GRP1_CNT0
#define PCLK_TCPWMx_CLOCKSx_PWM         PCLK_TCPWM0_CLOCKS256
#define TCPWM_PERI_CLK_DIVIDER_NO_PWM   0ul
#define TCPWMx_PWM_PRESCALAR_DIV_x      CY_TCPWM_PWM_PRESCALER_DIVBY_128  // 2,000,000 / 128 = 15,625Hz
#define TCPWMx_PERIOD                   0x1000    // 15,625Hz / 4096 (0x1000) = 3.815Hz (PWM frequency)
#define TCPWMx_COMPARE0                 0x800     // 0x800 / 0x1000 = 0.5 (PWM duty)
/* TCPWM_LINE0 */
#define TCPWMx_LINEx_PORT                GPIO_PRT3
#define TCPWMx_LINEx_PIN                 5
#define TCPWMx_LINEx_MUX                 P3_5_TCPWM0_LINE256

/* PWM */
cy_stc_gpio_pin_config_t pin_cfg1 =
{
     .outVal    = 0ul,
     .driveMode = CY_GPIO_DM_STRONG_IN_OFF,
     .hsiom     = TCPWMx_LINEx_MUX,
     .intEdge   = 0ul,
     .intMask   = 0ul,
     .vtrip     = 0ul,
     .slewRate  = 0ul,
     .driveSel  = 0ul,
};

cy_stc_tcpwm_pwm_config_t const MyPWM_config =
{
     .pwmMode            = CY_TCPWM_PWM_MODE_PWM,
     .clockPrescaler     = TCPWMx_PWM_PRESCALAR_DIV_x,
     .debug_pause        = false,
     .Cc0MatchMode       = CY_TCPWM_PWM_TR_CTRL2_CLEAR,
     .OverflowMode       = CY_TCPWM_PWM_TR_CTRL2_SET,
     .UnderflowMode      = CY_TCPWM_PWM_TR_CTRL2_NO_CHANGE,
     .Cc1MatchMode       = CY_TCPWM_PWM_TR_CTRL2_NO_CHANGE,
     .deadTime           = 0ul,
```

GPIO Pin Configuration

TCPWM Configuration

Code Listing 13 Logical Channel Setting with Hardware Trigger in CYT2 Series Settings

```
      .deadTimeComp       = 0ul,
      .runMode            = CY_TCPWM_PWM_CONTINUOUS,
      .period             = TCPWMx_PERIOD - 1ul,
      .period_buff        = 0ul,
      .enablePeriodSwap   = false,
      .compare0           = TCPWMx_COMPARE0,
      .compare1           = 0ul,
      .enableCompare0Swap = false,
      .enableCompare1Swap = false,
      .interruptSources   = 0ul,
     .invertPWMOut       = 0ul,
      .invertPWMOutN      = 0ul,
      .killMode           = CY_TCPWM_PWM_STOP_ON_KILL,
      .switchInputMode    = 3ul,
      .switchInput        = 0ul,
      .reloadInputMode    = 3ul,
      .reloadInput        = 0ul,
      .startInputMode     = 3ul,
      .startInput         = 0ul,
      .kill0InputMode     = 3ul,
      .kill0Input         = 0ul,
      .kill1InputMode     = 3ul,
      .kill1Input         = 0ul,
      .countInputMode     = 3ul,
      .countInput         = 1ul,
};

/* ADC CONFIGURATION */

/* ADC port setting (Note default port setting after reset is just fine) */
cy_stc_gpio_pin_config_t adcPinConfig =
{
      .outVal    = 0ul,
      .driveMode = CY_GPIO_DM_ANALOG,
      .hsiom     = P6_0_GPIO,
      .intEdge   = 0ul,
      .intMask   = 0ul,
      .vtrip     = 0ul,
      .slewRate  = 0ul,
      .driveSel  = 0ul,
};
/* ADC Channel Configuration */
cy_stc_adc_config_t adcConfig =
{
     .preconditionTime    = 0ul,
      .powerupTime         = 0ul,
      .enableIdlePowerDown = false,
      .msbStretchMode      = CY_ADC_MSB_STRETCH_MODE_1CYCLE,
```

ADC Pin Configuration.

ADC Configuration. See **Code Listing 2**.

Hardware Trigger Procedure

Code Listing 13 Logical Channel Setting with Hardware Trigger in CYT2 Series Settings

```
    .enableHalfLsbConv    = 0ul,
    .sarMuxEnable         = true,
    .adcEnable            = true,
    .sarIpEnable          = true,
};
cy_stc_adc_channel_config_t adcChannelConfig =
{
    /* Trigger type: Trigger from corresponding TCPWM channel */
.triggerSelection          = CY_ADC_TRIGGER_TCPWM,
    .channelPriority           = 0ul,
    .preenptionType            = CY_ADC_PREEMPTION_FINISH_RESUME,
.isGroupEnd                = true,
    /* CY_ADC_DONE_LEVEL_PULSE = 0 */
.doneLevel                 = CY_ADC_DONE_LEVEL_PULSE,
    .pinAddress                = BB_POTI_ANALOG_INPUT_NO,
    .portAddress               = CY_ADC_PORT_ADDRESS_SARMUX0,
    .extMuxSelect              = 0ul,
    .extMuxEnable              = true,
    .preconditionMode          = CY_ADC_PRECONDITION_MODE_OFF,
    .overlapDiagMode           = CY_ADC_OVERLAP_DIAG_MODE_OFF,
    .sampleTime                = 0ul,
    .calibrationValueSelect    = CY_ADC_CALIBRATION_VALUE_REGULAR,
    .postProcessingMode        = CY_ADC_POST_PROCESSING_MODE_NONE,
    .resultAlignment           = CY_ADC_RESULT_ALIGNMENT_RIGHT,
    .signExtention             = CY_ADC_SIGN_EXTENTION_UNSIGNED,
    .averageCount              = 0ul,
    .rightShift                = 0ul,
    .rangeDetectionMode        = CY_ADC_RANGE_DETECTION_MODE_INSIDE_RANGE,
    .rangeDetectionLoThreshold = 0x0000ul,
    .rangeDetectionHiThreshold = 0x0FFFul,
    .mask.grpDone              = true,
    .mask.grpCancelled         = false,
    .mask.grpOverflow          = false,
    .mask.chRange              = false,
    .mask.chPulse              = false,
    .mask.chOverflow           = false,
};
void AdcIntHandler(void)
{
    cy_stc_adc_interrupt_source_t intrSource = { false };

    /* Get interrupt source */
    Cy_Adc_Channel_GetInterruptMaskedStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&intrSource);

    if(intrSource.grpDone)
    {
        /* Get the result(s) */
```

ADC Channel Configuration. See **Code Listing 5**.

(1) Logical Channel Priority Setting
Trigger Type Setting
Preemption Type Setting
Group End Setting
Level or Pulse Selection

(2) Analog Input Setting
Sample Time Setting

(3) Interrupt Mask Setting

Get Interrupt Masked Status. See **Code Listing 14**.

Code Listing 13 Logical Channel Setting with Hardware Trigger in CYT2 Series Settings

```
        Cy_Adc_Channel_GetResult(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &resultBuff[resultIdx],
&statusBuff[resultIdx]);


         /* Increment result idx */
        resultIdx = (resultIdx + 1) % (sizeof(resultBuff) / sizeof(resultBuff[0]));


        /* Clear interrupt source */
        Cy_Adc_Channel_ClearInterruptStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&intrSource);
     }
     else
     {
         /* Unexpected interrupt */
         CY_ASSERT(false);
     }
}


/* This is an ADC example file for HW trigger: TCPWM */


int main(void)
{
     SystemInit();


     __enable_irq(); /* Enable global interrupts. */


:
     /* Initialize ADC */
    uint32_t samplingCycle = (uint32_t)DIV_ROUND_UP((ANALOG_IN_SAMPLING_TIME_MIN_IN_NS *
(uint64_t)actualAdcOperationFreq), 1000000000ull);
     adcChannelConfig.sampleTime = samplingCycle;


     Cy_Adc_Init(BB_POTI_ANALOG_MACRO, &adcConfig);
     Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &adcChannelConfig);


     /* Register ADC interrupt handler and enable interru
     cy_stc_sysint_irq_t irq_cfg;
     irq_cfg = (cy_stc_sysint_irq_t){
             .sysIntSrc  = pass_0_interrupts_sar_0_IRQn,
             .intIdx     = CPUIntIdx3_IRQn,
             .isEnabled  = true,
     };
     Cy_SysInt_InitIRQ(&irq_cfg);
     Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, AdcIntHandler);
     NVIC_SetPriority(irq_cfg.intIdx, 0ul);
     NVIC_EnableIRQ(irq_cfg.intIdx);
  :
     /* Port Configuration for TCPWM */
    Cy_GPIO_Pin_Init(TCPWMx_LINEx_PORT, TCPWMx_LINEx_PIN, &pin_cfg1);
```

Read A/D conversion data. See **Code Listing 11**.

Clear and read A/D conversion flag. See **Code Listing 12**.

ADC Initialize. See **Code Listing 2**.

ADC Channel Initialize. See **Code Listing 5**.

(4) Initialize Interrupt Request. See **Code Listing 7**.

Initialize Interrupt Request. See **Code Listing 15**.

Initialize GPIO Pin. See **Code Listing 16**.

Hardware Trigger Procedure

Code Listing 13 Logical Channel Setting with Hardware Trigger in CYT2 Series Settings

```
    /* Initialize TCPWM0_GRPx_CNTx_PWM_PR as PWM Mode & Enable */
    Cy_Tcpwm_Pwm_Init(TCPWMx_GRPx_CNTx_PWM, &MyPWM_config);


    /* Enable ADC ch. and PWM */
    Cy_Adc_Channel_Enable(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);
    Cy_Tcpwm_Pwm_Enable(TCPWMx_GRPx_CNTx_PWM);
   Cy_Tcpwm_TriggerStart(TCPWMx_GRPx_CNTx_PWM);

    /* Trigger MUX */
    Cy_TrigMux_Connect1To1(TRIG_IN_1TO1_1_TCPWM_TO_PASS_CH_TR0,
                           0ul,
                           TRIGGER_TYPE_PASS_TR_SAR_CH_IN__EDGE,
                           0ul);

    for(;;);
}
```

Initialize TCPWM PWM. See **Code Listing 17**.

(5) Enable ADC channel. See **Code Listing 8**.

TCPWM PWM Enable. See **Code Listing 18**.

TCPWM Trigger Start Select. See **Code Listing 19**.

Connects an input trigger source and output trigger. See **Code Listing 20**.

Code Listing 14 Cy_Adc_Channel_GetInterruptMaskedStatus() Function

```
cy_en_adc_status_t Cy_Adc_Channel_GetInterruptStatus(const volatile stc_PASS_SAR_CH_t * base,
cy_stc_adc_interrupt_source_t * status)
{
    cy_en_adc_status_t ret = CY_ADC_SUCCESS;
    un_PASS_SAR_CH_INTR_t unStat;

    if (NULL != status)
    {
        unStat.u32Register   = base->unINTR.u32Register;
        status->chOverflow   = (unStat.stcField.u1CH_OVERFLOW   != 0ul) ? true : false;
        status->chPulse      = (unStat.stcField.u1CH_PULSE      != 0ul) ? true : false;
        status->chRange      = (unStat.stcField.u1CH_RANGE      != 0ul) ? true : false;
        status->grpCancelled = (unStat.stcField.u1GRP_CANCELLED != 0ul) ? true : false;
        status->grpDone      = (unStat.stcField.u1GRP_DONE      != 0ul) ? true : false;
        status->grpOverflow  = (unStat.stcField.u1GRP_OVERFLOW  != 0ul) ? true : false;
    }
    else
    {
        ret = CY_ADC_BAD_PARAM;
    }
    return ret;
}
```

Hardware Trigger Procedure

Code Listing 15 Cy_SysInt_SetSystemIrqVector() Function

```
void  Cy_SysInt_SetSystemIrqVector(cy_en_intr_t sysIntSrc, cy_systemIntr_Handler userIsr)
{
    if (Cy_SysInt_SystemIrqUserTableRamPointer != NULL)
    {
        Cy_SysInt_SystemIrqUserTableRamPointer[sysIntSrc] = userIsr;
    }
}
```

Code Listing 16 Cy_GPIO_Pin_Init() Function

```
cy_en_gpio_status_t Cy_GPIO_Pin_Init(volatile stc_GPIO_PRT_t *base, uint32_t pinNum, const
cy_stc_gpio_pin_config_t *config)
{
    cy_en_gpio_status_t status = CY_GPIO_SUCCESS;

    if((NULL != base) && (NULL != config))
    {
        Cy_GPIO_Write(base, pinNum, config->outVal);
        Cy_GPIO_SetHSIOM(base, pinNum, config->hsiom);
        Cy_GPIO_SetVtrip(base, pinNum, config->vtrip);
        Cy_GPIO_SetSlewRate(base, pinNum, config->slewRate);
        Cy_GPIO_SetDriveSel(base, pinNum, config->driveSel);
        Cy_GPIO_SetDrivemode(base, pinNum, config->driveMode);
        Cy_GPIO_SetInterruptEdge(base, pinNum, config->intEdge);
        Cy_GPIO_ClearInterrupt(base, pinNum);
        Cy_GPIO_SetInterruptMask(base, pinNum, config->intMask);
    }
    else
    {
        status = CY_GPIO_BAD_PARAM;
    }

    return(status);
}
```

Code Listing 17 Cy_Tcpwm_Pwm_Init() Function

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

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

Hardware Trigger Procedure

Code Listing 17 Cy_Tcpwm_Pwm_Init() Function

```
    un_TCPWM_GRP_CNT_CTRL_t workCTRL = {.u32Register = 0ul};
    workCTRL.stcField.u1ONE_SHOT              = config->runMode;
    workCTRL.stcField.u2UP_DOWN_MODE          = config->countDirection;
    workCTRL.stcField.u3MODE                  = config->pwmMode;
    workCTRL.stcField.u1DBG_FREEZE_EN         = config->debug_pause;
    workCTRL.stcField.u1AUTO_RELOAD_CC0       = config->enableCompare0Swap;
    workCTRL.stcField.u1AUTO_RELOAD_CC1       = config->enableCompare1Swap;
    workCTRL.stcField.u1AUTO_RELOAD_PERIOD    = config->enablePeriodSwap;
    workCTRL.stcField.u1AUTO_RELOAD_LINE_SEL = config->enableLineSelSwap;
    workCTRL.stcField.u1PWM_SYNC_KILL         = config->killMode;
    workCTRL.stcField.u1PWM_STOP_ON_KILL      = (config->killMode>>1ul);
    ptscTCPWM->unCTRL.u32Register             = workCTRL.u32Register;

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

    ptscTCPWM->unCC0.u32Register = config->compare0;

    ptscTCPWM->unCC1.u32Register = config->compare1;

    ptscTCPWM->unPERIOD.u32Register = config->period;

    un_TCPWM_GRP_CNT_TR_IN_SEL0_t workTR_IN_SEL0 = {.u32Register = 0ul};
    workTR_IN_SEL0.stcField.u8CAPTURE0_SEL = config->switchInput;
    workTR_IN_SEL0.stcField.u8RELOAD_SEL   = config->reloadInput;
    workTR_IN_SEL0.stcField.u8STOP_SEL     = config->kill0Input;
    workTR_IN_SEL0.stcField.u8COUNT_SEL    = config->countInput;
    ptscTCPWM->unTR_IN_SEL0.u32Register    = workTR_IN_SEL0.u32Register;

    un_TCPWM_GRP_CNT_TR_IN_SEL1_t workTR_IN_SEL1 = {.u32Register = 0ul};
    workTR_IN_SEL1.stcField.u8CAPTURE1_SEL = config->kill1Input;
    workTR_IN_SEL1.stcField.u8START_SEL    = config->startInput;
    ptscTCPWM->unTR_IN_SEL1.u32Register    = workTR_IN_SEL1.u32Register;

    un_TCPWM_GRP_CNT_TR_IN_EDGE_SEL_t workTR_IN_EDGE_SEL = {.u32Register = 0ul};
    workTR_IN_EDGE_SEL.stcField.u2CAPTURE0_EDGE = config->switchInputMode;
    workTR_IN_EDGE_SEL.stcField.u2CAPTURE1_EDGE = config->kill1InputMode;
    workTR_IN_EDGE_SEL.stcField.u2RELOAD_EDGE   = config->reloadInputMode;
```

Hardware Trigger Procedure

Code Listing 17 Cy_Tcpwm_Pwm_Init() Function

```
        workTR_IN_EDGE_SEL.stcField.u2START_EDGE    = config->startInputMode;
        workTR_IN_EDGE_SEL.stcField.u2STOP_EDGE     = config->kill0InputMode;
        workTR_IN_EDGE_SEL.stcField.u2COUNT_EDGE    = config->countInputMode;
        ptscTCPWM->unTR_IN_EDGE_SEL.u32Register     = workTR_IN_EDGE_SEL.u32Register;

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

        un_TCPWM_GRP_CNT_TR_PWM_CTRL_t workTR_PWM_CTRL = {.u32Register = 0ul};
        workTR_PWM_CTRL.stcField.u2CC0_MATCH_MODE = config->Cc0MatchMode;
        workTR_PWM_CTRL.stcField.u2CC1_MATCH_MODE = config->Cc1MatchMode;
        workTR_PWM_CTRL.stcField.u2OVERFLOW_MODE  = config->OverflowMode;
        workTR_PWM_CTRL.stcField.u2UNDERFLOW_MODE = config->UnderflowMode;
        ptscTCPWM->unTR_PWM_CTRL.u32Register      = workTR_PWM_CTRL.u32Register;

        un_TCPWM_GRP_CNT_DT_t workDT = {.u32Register = 0ul};
        workDT.stcField.u16DT_LINE_COMPL_OUT = config->deadTimeComp;
        workDT.stcField.u8DT_LINE_OUT_H      = (config->deadTime>>8u);
        if(config->pwmMode == CY_TCPWM_PWM_MODE_DEADTIME)
        {
            workDT.stcField.u8DT_LINE_OUT_L = config->deadTime;
        }
        else
        {
            workDT.stcField.u8DT_LINE_OUT_L = config->clockPrescaler;
        }
        ptscTCPWM->unDT.u32Register          = workDT.u32Register;

        status = CY_RET_SUCCESS;

    }

    return (status);
}
```

Code Listing 18 Cy_Tcpwm_Pwm_Enable() Function

```
void Cy_Tcpwm_Pwm_Enable(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unCTRL.stcField.u1ENABLED = 0x1ul;
}
```

Hardware Trigger Procedure

Code Listing 19 Cy_Tcpwm_TriggerStart() Function

```
void Cy_Tcpwm_TriggerStart(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unTR_CMD.stcField.u1START = 0x1ul;
}
```

Code Listing 20 Cy_TrigMux_Connect1To1() Function

```
cy_en_trigmux_status_t Cy_TrigMux_Connect1To1(uint32_t trig, uint32_t invert, en_trig_type_t trigType,
uint32_t dbg_frz_en)
{
    cy_en_trigmux_status_t retVal = CY_TRIGMUX_BAD_PARAM;

    /* Validate output trigger */
    if(Cy_TrigMux_IsOneToOne(trig) == false)
    {
        /* input trigger parameter is not One-To-One type */
        return retVal;
    }

    /* Distill group and trigger No value from input trigger parameter */
    uint8_t trigGroup = Cy_TrigMux_GetGroup(trig);
    uint8_t trigNo    = Cy_TrigMux_GetNo(trig);

    /* Get a pointer to target trigger setting register */
    volatile stc_PERI_TR_1TO1_GR_TR_CTL_field_t* pTR_CTL;
    pTR_CTL = &(PERI->TR_1TO1_GR[trigGroup].unTR_CTL[trigNo].stcField);

    /* Set input parameters to the register */
    pTR_CTL->u1TR_SEL        = true;
    pTR_CTL->u1TR_INV        = invert;
    pTR_CTL->u1TR_EDGE       = trigType;
    pTR_CTL->u1DBG_FREEZE_EN = dbg_frz_en;

    /* Return success status */
    retVal = CY_TRIGMUX_SUCCESS;
    return retVal;
}
```

3.2 A/D Conversion ISR with Hardware Trigger

Figure 6 shows the example of ADC ISR with a hardware trigger. For details on CPU interrupt handing, see the Architecture TRM mentioned in **Related Documents**.

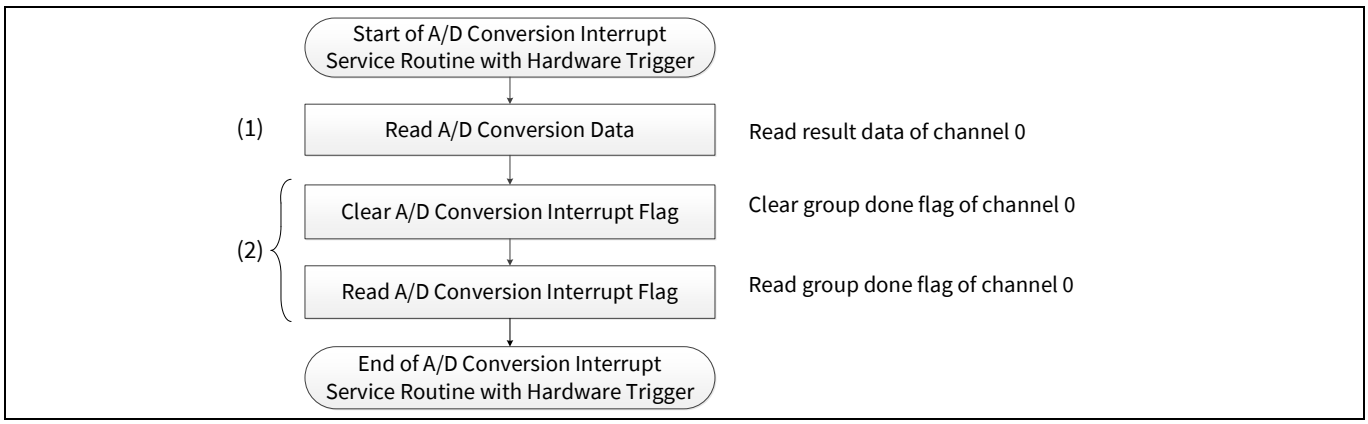


Figure 6 Example of ADC ISR with Hardware Trigger

3.2.1 Use Case

See Section **3.1.1**.

3.2.2 Configuration

Table 9 lists the parameters and **Table 10** lists the functions of the configuration part of in SDL for ADC ISR with Hardware Trigger settings.

Table 9 List of ADC ISR with Hardware Trigger Parameters

Parameters	Description	Value
`BB_POTI_ANALOG_MACRO`	Analog macro for the potentiometer on Traveo II Base Board	CY_ADC_POT_MACRO: PASS0_SAR0
`resultBuff`	Conversion Result Buffer	- (Calculated Value)
`statusBuff`	Status Result Buffer	- (Calculated Value)
`resultIdx`	Index Result	- (Calculated Value)
`intrSource`	Interrupt Source	- (Calculated Value)

Table 10 List of ADC ISR with Hardware Trigger Settings Functions

Functions	Description	Value
`Cy_Adc_Channel_GetResult(SAR Channel,Status)`	Get the conversion result and status.	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Status = resultBuff[resultIdx]
`Cy_Adc_Channel_ClearInterruptStatus(SAR Channel,Source)`	Clear the corresponding channel interrupt status.	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Source = intrSource

3.2.3 Sample Code

See **Code Listing 21** for sample code for initial configuration of ADC ISR with software trigger settings.

Code Listing 21 ADC ISR with Software Trigger Settings

```
#define BB_POTI_ANALOG_MACRO        CY_ADC_POT_MACRO
:
uint16_t               resultBuff[16];
cy_stc_adc_ch_status_t statusBuff[16];
uint8_t                resultIdx;
:

void AdcIntHandler(void)
{
         cy_stc_adc_interrupt_source_t intrSource = { false };
:
    {
        /* Get the result(s) */
        Cy_Adc_Channel_GetResult(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&resultBuff[resultIdx], &statusBuff[resultIdx]);

        /* Display ADC result */
        printf("\rADC result = %04d          ", resultBuff[resultIdx]);

        /* Increment result idx */
        resultIdx = (resultIdx + 1) % (sizeof(resultBuff) / sizeof(resultBuff[0]));

        /* Clear interrupt source */
        Cy_Adc_Channel_ClearInterruptStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&intrSource);
 :
    }
    else
    {
        // Unexpected interrupt
        CY_ASSERT(false);
    }
}
```

(1) Read A/D conversion data. See Code Listing 11.

(2) Clear and read A/D conversion flag. See **Code Listing 21**.

4 Group Procedure

The SAR ADC has a function for consecutive conversion using multiple pins with one trigger. The pins and the order of conversions can be selected from any ports that can be configured as 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. The first channel of the group has the trigger type set to ‘OFF’ (no hardware trigger), ‘TCPWM’, ‘Generic 0 to 4’, or ‘Continuous’.

If the “continue group with next channel” is not set by 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 7 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 Settings**.

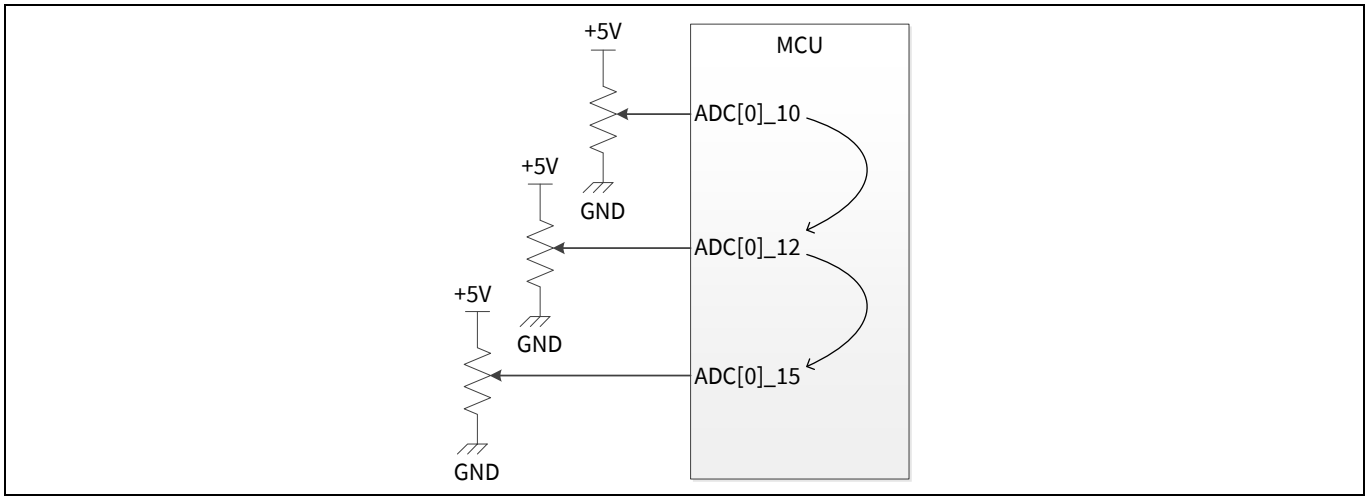


Figure 7 Example of Group Conversion Connection

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

4.1 Logical Channel Settings for Group Procedure

Figure 8 shows the example of logical channel setting for a group procedure in CYT2 series. 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 **Related Documents**.

Group Procedure

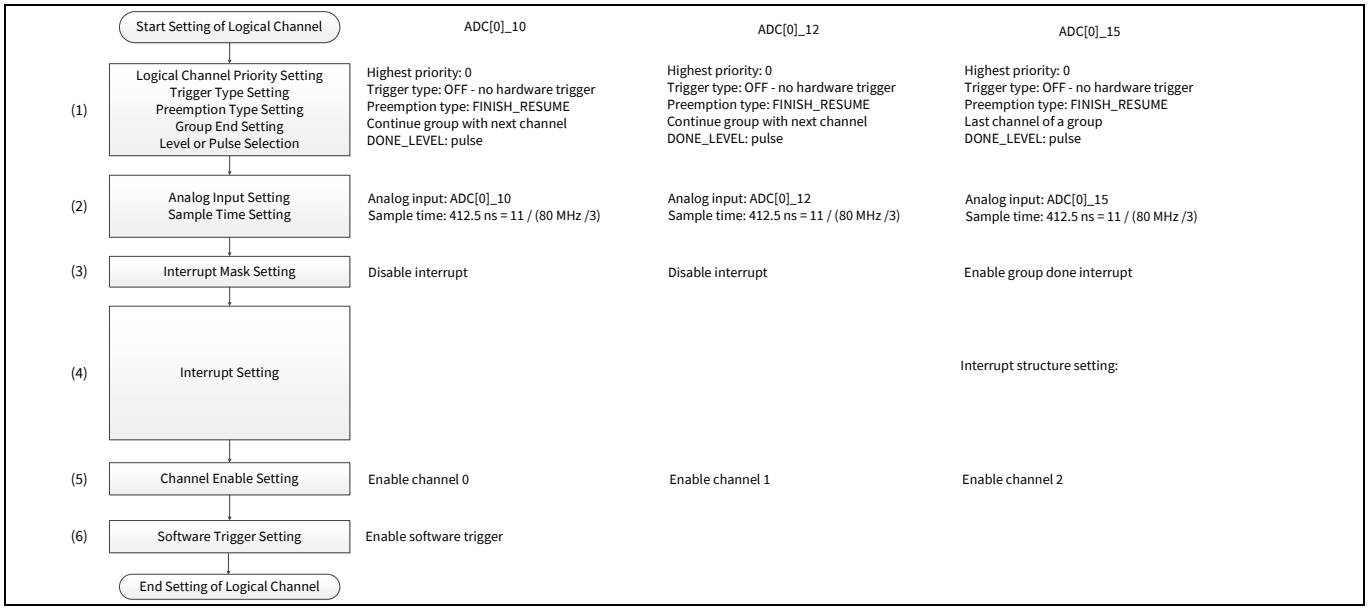


Figure 8 Example of Logical Channel Setting for Group in CYT2 Series

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 Logical Channel Setting for Group.

- 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]_10	Continue group with next channel	Disable
CH1	ADC[0]_12	Continue group with next channel	Disable
CH2	ADC[0]_15	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
- Mask Channel: Not Range/Not Pulse/Not Overflow

Group Procedure

4.1.2 Configuration

Table 11 lists the parameters and Table 12 lists the functions of Logical Channel Setting for Group in CYT2 Series.

Table 11 List of Logical Channel Setting for Group in CYT2 Series Parameters

Parameters	Description	Value
ADC_LOGICAL_CHANNEL	ADC Logical Channel	0ul
ADC_GROUP_NUMBER_OF_CHANNELS	ADC Channel Number in the group	3ul
ADC_GROUP_FIRST_CHANNEL	First Channel of ADC Group	ADC_LOGICAL_CHANNEL
ADC_GROUP_LAST_CHANNEL	Last Channel of ADC Group	ADC_GROUP_FIRST_CHANNEL + ADC_GROUP_NUMBER_OF_CHANNELS - 1ul
BB_POTI_ANALOG_MACRO	Analog macro for the potentiometer on Traveo II Base Board	CY_ADC_POT_MACRO: PASS0_SAR0
adcChannelConfig.triggerSelection	Trigger OFF	0ul (See Table 3)
adcChannelConfig.channelPriority	Channel Priority	0ul
adcChannelConfig.preenptionType	Pre-emption Type	3ul (See Table 3)
adcChannelConfig.isGroupEnd	Is Group End?	CH0: false, CH1: false, CH2: true
adcChannelConfig.doneLevel	Done Level	0ul (See Table 3)
adcChannelConfig.pinAddress	Pin Address	BB_POTI_ANALOG_INPUT_NO
adcChannelConfig.portAddress	Port Address	0ul (See Table 3)
adcChannelConfig.extMuxSelect	External MUX Select	0ul
adcChannelConfig.extMuxEnable	Enable External MUX	true
adcChannelConfig.preconditionMode	Pre-condition Mode	0ul (See Table 3)
adcChannelConfig.overlapDiagMode	Overlap Diagnostics Mode	0ul (See Table 3)
adcChannelConfig.sampleTime	Sample Time	samplingCycle (Calculated Value)
adcChannelConfig.calibrationValueSelect	Calibration Value Select	0ul (See Table 3)
adcChannelConfig.postProcessingMode	Post Processing Mode	0ul (See Table 3)
adcChannelConfig.resultAlignment	Result Alignment	0ul (See Table 3)
adcChannelConfig.signExtention	Sign Extension	0ul (See Table 3)
adcChannelConfig.averageCount	Average Count	0ul
adcChannelConfig.rightShift	Right Shift	0ul
adcChannelConfig.rangeDetectionMode	Range Detection Mode	2ul (See Table 3)
adcChannelConfig.rangeDetectionLoThreshold	Range Detection Low Threshold	0x0000ul

Group Procedure

Parameters	Description	Value
adcChannelConfig.rangeDetectionHiThreshold	Range Detection High Threshold	0x0FFFul
adcChannelConfig.mask.grpDone	Mask Group Done	CH0: false, CH1: false, CH2: true
adcChannelConfig.mask.grpCancelled	Mask Group Cancelled	false
adcChannelConfig.mask.grpOverflow	Mask Group Overflow	false
adcChannelConfig.mask.chRange	Mask Channel Range	false
adcChannelConfig.mask.chPulse	Mask Channel Pulse	false
adcChannelConfig.mask.chOverflow	Mask Channel Overflow	false

Table 12 List of Logical Channel Setting for Group in CYT2 Series Functions

Functions	Description	Value
Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)	Initialize the ADC channel	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig
Cy_SysInt_InitIRQ(Config)	Initialize the referenced system interrupt by setting the interrupt vector	Config = irq_cfg
Cy_Adc_Channel_Enable(SARchannel)	Enable the corresponding channel	SARchannel = PASS_SAR_CH
Cy_Adc_Channel_SoftwareTrigger(PASS SARchannel)	Issue a software start trigger	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

4.1.3 Sample Code

See **Code Listing 22** for sample code for initial configuration of logical channel settings for group procedure settings.

Code Listing 22 Logical Channel Settings for Group Procedure Settings

```
#define BB_POTI_ANALOG_PCLK          CY_ADC_POT_PCLK
#define BB_POTI_ANALOG_INPUT_NO     ((cy_en_adc_pin_address_t)CY_ADC_POT_IN_NO)
:
#define ADC_GROUP_FIRST_CHANNEL          ADC_LOGICAL_CHANNEL
#define ADC_GROUP_LAST_CHANNEL           (ADC_GROUP_FIRST_CHANNEL + ADC_GROUP_NUMBER_OF_CHANNELS - 1u)

uint16_t               resultBuff[ADC_GROUP_NUMBER_OF_CHANNELS][16];
cy_stc_adc_ch_status_t statusBuff[ADC_GROUP_NUMBER_OF_CHANNELS][16];
```

Group Procedure

Code Listing 22 Logical Channel Settings for Group Procedure Settings

```
uint8_t                 resultIdx;

:
int main(void)
{
:
    __enable_irq(); /* Enable global interrupts. */

    /* Initialize ADC */
    {
:
        cy_stc_adc_config_t adcConfig =
        {
            .preconditionTime    = 0ul,
            .powerupTime         = 0ul,
            .enableIdlePowerDown = false,
            .msbStretchMode      = CY_ADC_MSB_STRETCH_MODE_1CYCLE,
            .enableHalfLsbConv   = 0ul,
            .sarMuxEnable        = true,
            .adcEnable           = true,
            .sarIpEnable         = true,
        };
        cy_stc_adc_channel_config_t adcChannelConfig =
        {
            .triggerSelection          = CY_ADC_TRIGGER_OFF,
            .channelPriority           = 0ul,
            .preenptionType            = CY_ADC_PREEMPTION_FINISH_RESUME,
            .isGroupEnd                = false,
            .doneLevel                 = CY_ADC_DONE_LEVEL_PULSE,
            .pinAddress                = BB_POTI_ANALOG_INPUT_NO,
            .portAddress               = CY_ADC_PORT_ADDRESS_SARMUX0,
            .extMuxSelect              = 0ul,
            .extMuxEnable              = true,
            .preconditionMode          = CY_ADC_PRECONDITION_MODE_OFF,
            .overlapDiagMode           = CY_ADC_OVERLAP_DIAG_MODE_OFF,
            .sampleTime                = samplingCycle,
            .calibrationValueSelect    = CY_ADC_CALIBRATION_VALUE_REGULAR,
            .postProcessingMode        = CY_ADC_POST_PROCESSING_MODE_NONE,
            .resultAlignment           = CY_ADC_RESULT_ALIGNMENT_RIGHT,
            .signExtention             = CY_ADC_SIGN_EXTENTION_UNSIGNED,
            .averageCount              = 0u,
            .rightShift                = 0u,
            .rangeDetectionMode        = CY_ADC_RANGE_DETECTION_MODE_INSIDE_RANGE,
            .rangeDetectionLoThreshold = 0x0000ul,
            .rangeDetectionHiThreshold = 0x0FFFul,
```

ADC Configuration.

(1) Trigger Type Setting
Preemption Type Setting
Group End Setting
Level or Pulse Selection

(2) Analog Input Setting
Sample Time Setting

Group Procedure

Code Listing 22 Logical Channel Settings for Group Procedure Settings

```
            .mask.grpDone               = false,
            .mask.grpCancelled          = false,
            .mask.grpOverflow           = false,
            .mask.chRange               = false,
            .mask.chPulse               = false,
            .mask.chOverflow            = false,
        };

        Cy_Adc_Init(BB_POTI_ANALOG_MACRO, &adcConfig);

        adcChannelConfig.isGroupEnd   = false;
        adcChannelConfig.mask.grpDone = false;
        for (uint8_t ch = ADC_GROUP_FIRST_CHANNEL; ch < (ADC_GROUP_FIRST_CHANNEL +
ADC_GROUP_NUMBER_OF_CHANNELS); ch++)
        {
            uint8_t i = ch - ADC_GROUP_FIRST_CHANNEL; // i is 0-based for array indexing

            if(ch == ADC_GROUP_LAST_CHANNEL)
            {
                adcChannelConfig.isGroupEnd   = true;
                adcChannelConfig.mask.grpDone = true;
            }
            adcChannelConfig.pinAddress = aenAnalogInp

            Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ch], &adcChannelConfig);
        }
:
        Cy_SysInt_InitIRQ(&irq_cfg);
:
    /* Enable ADC ch. */
    for (uint8_t ch = ADC_GROUP_FIRST_CHANNEL; ch < (ADC_GROUP_FIRST_CHANNEL +
ADC_GROUP_NUMBER_OF_CHANNELS); ch++)
    {
        Cy_Adc_Channel_Enable(&CY_ADC_POT_MACRO->CH[ch]);
    }
:
    /* Issue SW trigger */
    Cy_Adc_Channel_SoftwareTrigger(&CY_ADC_POT_MACRO->CH[ADC_GROUP_FIRST_CHANNEL]);
:
    for(;;)
    {
    }
}
```

(3) Interrupt Mask Setting

ADC Configuration setting. See Code Listing 2.

ADC Channel Initialize. See **Code Listing 5**.

(4) Initialize Interrupt Request. See **Code Listing 7**.

(5) Enable ADC Channel. See **Code Listing 8**.

(6) Software Trigger Setting. See **Code Listing 9**.

4.2 A/D Conversion ISR for Group

Figure 9 shows the example of ADC ISR for the group. For details on CPU interrupt handling, see the Architecture TRM mentioned in the **Related Documents**.

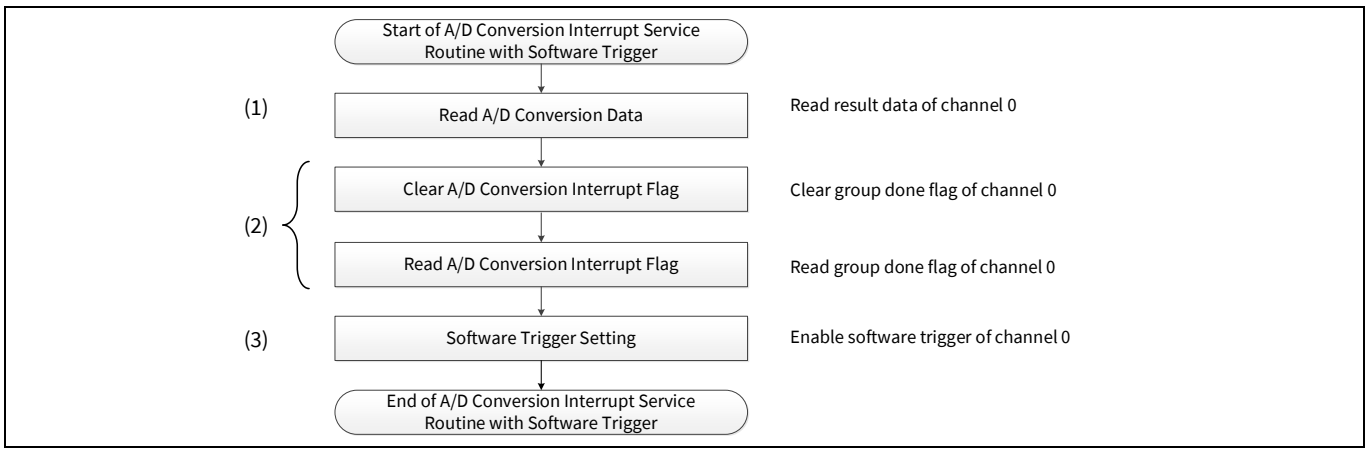


Figure 9 Example of ADC ISR for Group

4.2.1 Use Case

See Section **4.1.1**.

4.2.2 Configuration

Table 13 lists the parameters and **Table 14** lists the functions of the configuration part of in SDL for ADC ISR for Group settings.

Table 13 List of ADC ISR for Group Settings Parameters

Parameters	Description	Value
`resultBuff`	Conversion Result Buffer	- (Calculated Value)
`statusBuff`	Status Result Buffer	- (Calculated Value)
`resultIdx`	Index Result	- (Calculated Value)
`intrSource`	Interrupt Source	- (Calculated Value)

Table 14 List of ADC ISR for Group Settings Functions

Functions	Description	Value
`Cy_Adc_Channel_GetResult(SAR Channel,Status)`	Get conversion result and status.	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Status = resultBuff[resultIdx]
`Cy_Adc_Channel_ClearInterruptStatus(SAR Channel,Source)`	Clear the corresponding channel interrupt status.	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Source = intrSource
`Cy_Adc_Channel_SoftwareTrigger(SAR Channel)`	Issue a software start trigger.	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

4.2.3 Sample Code

See **Code Listing 23** for sample code for initial configuration of ADC ISR for group settings.

Code Listing 23 ADC ISR for Group Settings

```
#define BB_POTI_ANALOG_MACRO        CY_ADC_POT_MACRO
:
#define ADC_GROUP_FIRST_CHANNEL         ADC_LOGICAL_CHANNEL
#define ADC_GROUP_LAST_CHANNEL          (ADC_GROUP_FIRST_CHANNEL + ADC_GROUP_NUMBER_OF_CHANNELS - 1u)

uint16_t               resultBuff[ADC_GROUP_NUMBER_OF_CHANNELS][16];
cy_stc_adc_ch_status_t statusBuff[ADC_GROUP_NUMBER_OF_CHANNELS][16];
uint8_t                resultIdx;

void AdcIntHandler(void)
{
    cy_stc_adc_interrupt_source_t intrSource = { false };
:
    {
:
        /* Get the result(s) */
        for(uint8_t ch = ADC_GROUP_FIRST_CHANNEL; ch < (ADC_GROUP_FIRST_CHANNEL +
ADC_GROUP_NUMBER_OF_CHANNELS); ch++)
        {
            uint8_t i = ch - ADC_GROUP_FIRST_CHANNEL; // i is 0-based for array indexing
            Cy_Adc_Channel_GetResult(&BB_POTI_ANALOG_MACRO->CH[ch], &resultBuff[i][resultIdx],
&statusBuff[i][resultIdx]);
            printf("AN%d = %04d, ", aenAnalogInput[i], resultBu
        }

        /* Increment result idx */
        resultIdx = (resultIdx + 1ul) % (sizeof(resultBuff[0]) / sizeof(resultBuff[0][0]));

        /* Clear interrupt source(s) (Only last channel is required) */
        Cy_Adc_Channel_ClearInterruptStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_GROUP_LAST_CHANNEL],
&intrSource);

        /* Trigger next conversion */
        Cy_Adc_Channel_SoftwareTrigger(&BB_POTI_ANALOG_MACRO->CH[ADC_GROUP_FIRST_CHANNEL]);
    }
    else
    {
        // Unexpected interrupt
        CY_ASSERT(false);
    }
}
```

(1) Read A/D conversion data. See Code Listing 11.

(2) Clear and read A/D conversion flag. See Code Listing 12.

(3) Software Trigger Setting. See **Code Listing 9**.

5 Averaging Procedure

Averaging is fully configured per channel by SARn_CHx_POST_CTL register.

The number of samples averaged is up to 256.

This section shows an example application that converts 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 Settings**, **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 Averaging Settings

Figure 10 shows the example of averaging setting.

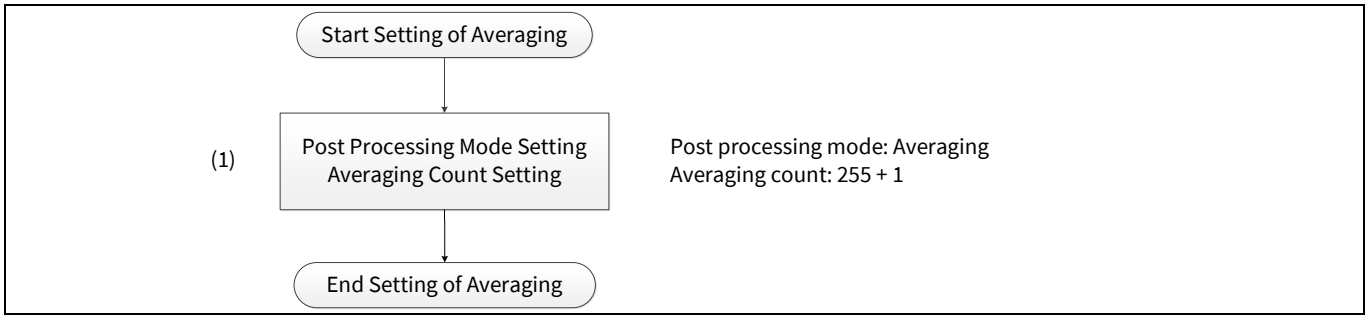


Figure 10 Example of Averaging Setting

5.1.1 Use Case

The following use case is an example of Averaging Setting.

- Post Processing Mode: Averaging
- Averaging Count: 255 + 1 times
- Other use case items are same as Section **2.3.1**.

5.1.2 Configuration

Table 15 lists the parameters and **Table 16** lists the functions of the configuration part of in SDL for Averaging Settings.

Table 15 List of Averaging Settings Parameters

Parameters	Description	Value
BB_POTI_ANALOG_MACRO	Analog macro for the potentiometer on Traveo II Base Board	CY_ADC_POT_MACRO: PASS0_SAR0
adcChannelConfig.triggerSelection	Trigger OFF	0ul (See **Table 3**)
adcChannelConfig.channelPriority	Channel Priority	0ul
adcChannelConfig.preenptionType	Pre-emption Type	3ul (See **Table 3**)

Averaging Procedure

Parameters	Description	Value
`adcChannelConfig.isGroupEnd`	Is Group End?	true
`adcChannelConfig.doneLevel`	Done Level	0ul (See **Table 3**)
`adcChannelConfig.pinAddress`	Pin Address	BB_POTI_ANALOG_INPUT_NO
`adcChannelConfig.portAddress`	Port Address	0ul (See **Table 3**)
`adcChannelConfig.extMuxSelect`	External MUX Select	0ul
`adcChannelConfig.extMuxEnable`	Enable External MUX	true
`adcChannelConfig.preconditionMode`	Pre-condition Mode	0ul (See **Table 3**)
`adcChannelConfig.overlapDiagMode`	Overlap Diagnostics Mode	0ul (See **Table 3**)
`adcChannelConfig.sampleTime`	Sample Time	0ul
`adcChannelConfig.calibrationValueSelect`	Calibration Value Select	0ul (See **Table 3**)
`adcChannelConfig.postProcessingMode`	Post Processing Mode	1ul (See **Table 3**)
`adcChannelConfig.resultAlignment`	Result Alignment	0ul (See **Table 3**)
`adcChannelConfig.signExtention`	Sign Extension	0ul (See **Table 3**)
`adcChannelConfig.averageCount`	Average Count	255ul
`adcChannelConfig.rightShift`	Right Shift	0ul
`adcChannelConfig.rangeDetectionMode`	Range Detection Mode	0ul (See **Table 3**)
`adcChannelConfig.rangeDetectionLoThreshold`	Range Detection Low Threshold	0x0000ul
`adcChannelConfig.rangeDetectionHiThreshold`	Range Detection High Threshold	0x0FFFul
`adcChannelConfig.mask.grpDone`	Mask Group Done	true
`adcChannelConfig.mask.grpCancelled`	Mask Group Cancelled	false
`adcChannelConfig.mask.grpOverflow`	Mask Group Overflow	false
`adcChannelConfig.mask.chRange`	Mask Channel Range	false
`adcChannelConfig.mask.chPulse`	Mask Channel Pulse	false
`adcChannelConfig.mask.chOverflow`	Mask Channel Overflow	false

Table 16 List of Averaging Settings Functions

Functions	Description	Value
`Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)`	Initialize the ADC channel	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig

5.1.3 Sample Code

See **Code Listing 24** for sample code for initial configuration of averaging settings.

Averaging Procedure

Code Listing 24 Average Settings

```
:
/* ADC Channel Configuration */
cy_stc_adc_channel_config_t adcChannelConfig =
{
    .triggerSelection          = CY_ADC_TRIGGER_OFF,
    .channelPriority           = 0ul,
    .preenptionType            = CY_ADC_PREEMPTION_FINISH_RESUME,
    .isGroupEnd                = true,
    .doneLevel                 = CY_ADC_DONE_LEVEL_PULSE,
    .pinAddress                = BB_POTI_ANALOG_INPUT_NO,
    .portAddress               = CY_ADC_PORT_ADDRESS_SARMUX0,
    .extMuxSelect              = 0ul,
    .extMuxEnable              = true,
    .preconditionMode          = CY_ADC_PRECONDITION_MODE_OFF,
    .overlapDiagMode           = CY_ADC_OVERLAP_DIAG_MODE_OFF,
    .sampleTime                = 0ul,
    .calibrationValueSelect    = CY_ADC_CALIBRATION_VALUE_REGULAR,
    .postProcessingMode        = CY_ADC_POST_PROCESSING_MODE_AVG, /* Averaging setting */
    .resultAlignment           = CY_ADC_RESULT_ALIGNMENT_RIGHT,
    .signExtention             = CY_ADC_SIGN_EXTENTION_UNSIGNED,
    .averageCount              = 255ul, /* 255+1 times*/
    .rightShift                = 0ul,
    .rangeDetectionMode        = CY_ADC_RANGE_DETECTION_MODE_BELOW_LO,
    .rangeDetectionLoThreshold = 0x0000ul,
    .rangeDetectionHiThreshold = 0x0FFFul,
    .mask.grpDone              = true,
    .mask.grpCancelled         = false,
    .mask.grpOverflow          = false,
    .mask.chRange              = false,
    .mask.chPulse              = false,
    .mask.chOverflow           = false,
};
:
int main(void)
{
    SystemInit();

    /* Enable global interrupts. */
    __enable_irq();

    Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &adcChannelConfig);
:
    for(;;);
}
```

(1) Averaging Settings

ADC Channel Initialize. See **Code Listing 5**.

6 Range Detection Procedure

The 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 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 lesser than '1500', it generates an interrupt to read the value as shown in **Figure 11**. Analog-to-digital conversion is repeated at regular timing by 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 Settings** and **Logical Channel Setting with Hardware Trigger** sections.

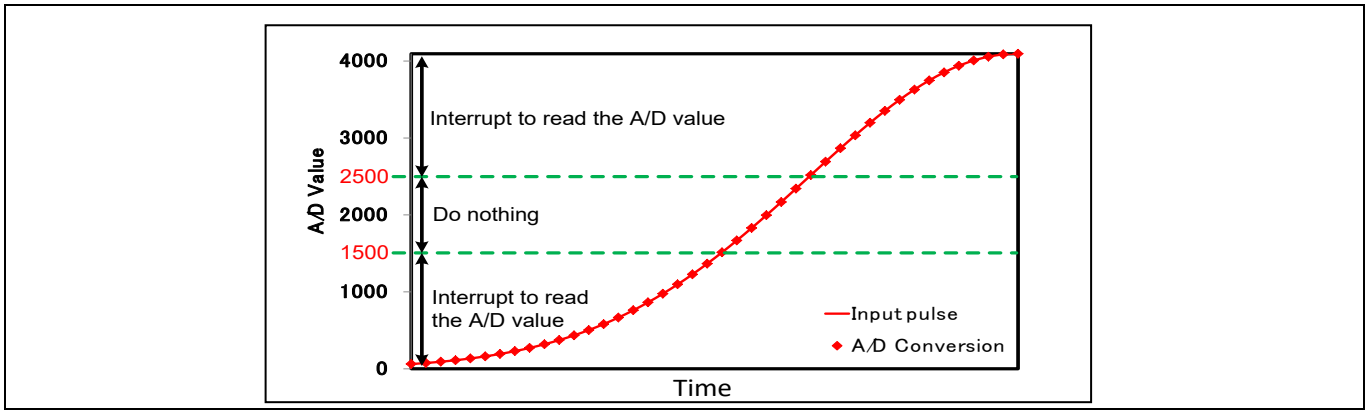


Figure 11 Example Behavior of Range Detection Application

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

6.1 Range Detection Settings

Figure 12 shows the example of Range Detection Setting.

Range Detection Procedure

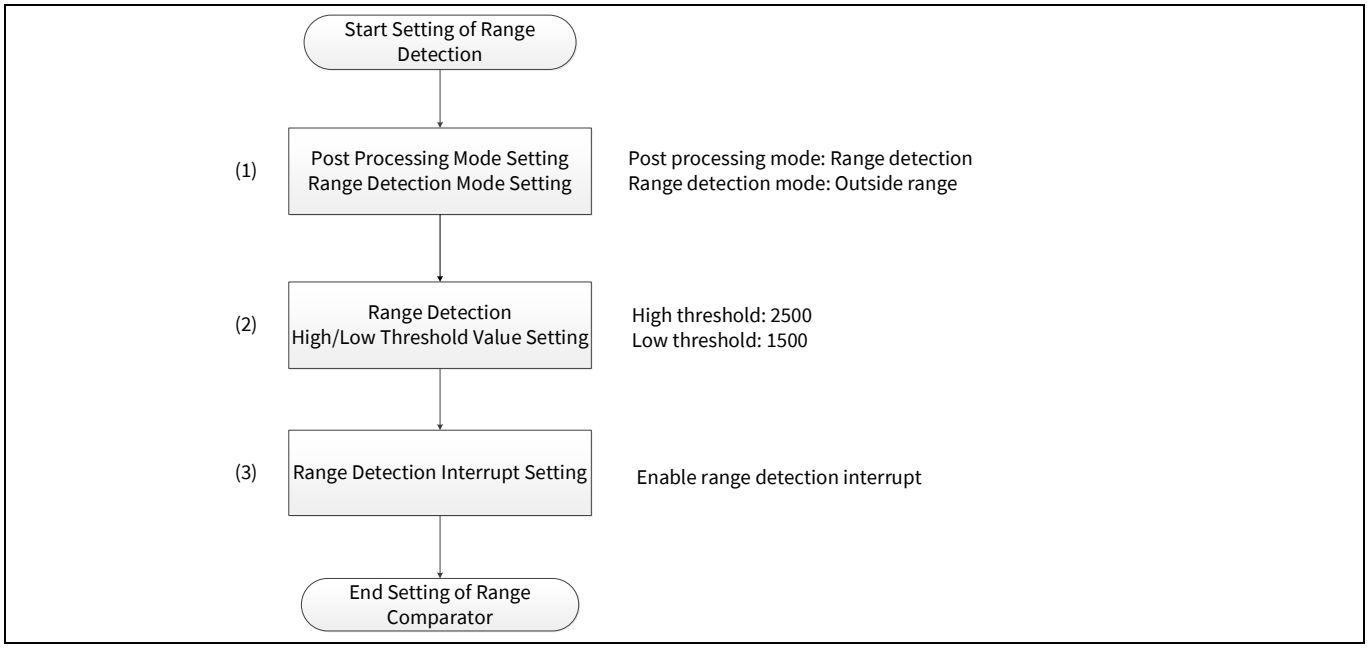


Figure 12 Example of Range Detection Setting

6.1.1 Use Case

The following use case is an example of Averaging Setting.

- 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 17 lists the parameters and **Table 18** lists the functions of the configuration part of in SDL for Range Detection Settings.

Table 17 List of Range Detection Settings Parameters

Parameters	Description	Value
`BB_POTI_ANALOG_MACRO`	Analog macro for the potentiometer on Traveo II Base Board	CY_ADC_POT_MACRO: PASS0_SAR0
`BB_POTI_ANALOG_INPUT_NO`	Analog Input Number for the potentiometer on Traveo II Base Board	CH0 (CH[ADC_LOGICAL_CHANNEL])
`adcChannelConfig.triggerSelection`	Trigger OFF	0ul (See **Table 3**)
`adcChannelConfig.channelPriority`	Channel Priority	0ul
`adcChannelConfig.preenptionType`	Pre-emption Type	3ul (See **Table 3**)
`adcChannelConfig.isGroupEnd`	Is Group End?	true
`adcChannelConfig.doneLevel`	Done Level	0ul (See **Table 3**)
`adcChannelConfig.pinAddress`	Pin Address	BB_POTI_ANALOG_INPUT_NO
`adcChannelConfig.portAddress`	Port Address	0ul (See **Table 3**)

Range Detection Procedure

Parameters	Description	Value
adcChannelConfig.extMuxSelect	External MUX Select	0ul
adcChannelConfig.extMuxEnable	Enable External MUX	true
adcChannelConfig.preconditionMode	Pre-condition Mode	0ul (See **Table 3**)
adcChannelConfig.overlapDiagMode	Overlap Diagnostics Mode	0ul (See **Table 3**)
adcChannelConfig.sampleTime	Sample Time	0ul
adcChannelConfig.calibrationValueSelect	Calibration Value Select	0ul (See **Table 3**)
adcChannelConfig.postProcessingMode	Post Processing Mode	3ul (See **Table 3**)
adcChannelConfig.resultAlignment	Result Alignment	0ul (See **Table 3**)
adcChannelConfig.signExtention	Sign Extension	0ul (See **Table 3**)
adcChannelConfig.averageCount	Average Count	0ul
adcChannelConfig.rightShift	Right Shift	0ul
adcChannelConfig.rangeDetectionMode	Range Detection Mode	3ul (See **Table 3**)
adcChannelConfig.rangeDetectionLoThreshold	Range Detection Low Threshold	0x09C4ul
adcChannelConfig.rangeDetectionHiThreshold	Range Detection High Threshold	0x05DCul
adcChannelConfig.mask.grpDone	Mask Group Done	false
adcChannelConfig.mask.grpCancelled	Mask Group Cancelled	false
adcChannelConfig.mask.grpOverflow	Mask Group Overflow	false
adcChannelConfig.mask.chRange	Mask Channel Range	true
adcChannelConfig.mask.chPulse	Mask Channel Pulse	false
adcChannelConfig.mask.chOverflow	Mask Channel Overflow	false

Table 18 List of Range Detection Settings Functions

Functions	Description	Value
Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)	Initialize the ADC channel	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig
Cy_SysInt_InitIRQ(Config)	Initialize the referenced system interrupt by setting the interrupt vector	Config = irq_cfg
Cy_Adc_Channel_Enable(PASS SARchannel)	Enable the corresponding channel	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]
Cy_Adc_Channel_SoftwareTrigger(PASS SARchannel)	Issue a software start trigger	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

6.1.3 Sample Code

See **Code Listing 25** for sample code for initial configuration of range detection settings.

Code Listing 25 Range Detection Settings

```
:
#define BB_POTI_ANALOG_INPUT_NO    ((cy_en_adc_pin_address_t)CY_ADC_POT_IN_NO)
 :
/* ADC Channel Configuration */
cy_stc_adc_channel_config_t adcChannelConfig =
{
    .triggerSelection          = CY_ADC_TRIGGER_GENERIC0,
    .channelPriority           = 0ul,
    .preenptionType            = CY_ADC_PREEMPTION_FINISH_RESUME,
    .isGroupEnd                = true,
    .doneLevel                 = CY_ADC_DONE_LEVEL_PULSE,
    .pinAddress                = BB_POTI_ANALOG_INPUT_NO,
    .portAddress               = CY_ADC_PORT_ADDRESS_SARMUX0,
    .extMuxSelect              = 0ul,
    .extMuxEnable              = true,
    .preconditionMode          = CY_ADC_PRECONDITION_MODE_OFF,
    .overlapDiagMode           = CY_ADC_OVERLAP_DIAG_MODE_OFF,
    .sampleTime                = 0ul, /* It will be update */
    .calibrationValueSelect    = CY_ADC_CALIBRATION_VALUE_REGULAR,
    .postProcessingMode        = CY_ADC_POST_PROCESSING_MODE_RANGE, /* Range detection */
    .resultAlignment           = CY_ADC_RESULT_ALIGNMENT_RIGHT,
    .signExtention             = CY_ADC_SIGN_EXTENTION_UNSIGNED,
    .averageCount              = 0ul,
    .rightShift                = 0ul,
    .rangeDetectionMode        = CY_ADC_RANGE_DETECTION_MODE_OUTSIDE_RANGE,/* Outside Mode */
    .rangeDetectionLoThreshold = 0x09C4ul, /* 2500 */
    .rangeDetectionHiThreshold = 0x05DCul, /* 1500 */
    .mask.grpDone              = false,
    .mask.grpCancelled         = false,
    .mask.grpOverflow          = false,
    .mask.chRange              = true, /* Range detection interrupt */
    .mask.chPulse              = false,
    .mask.chOverflow           = false,
};
:
int main(void)
{
    SystemInit();

    __enable_irq(); /* Enable global interrupts. */
 :
    /* Initialize ADC */
    uint32_t samplingCycle = (uint32_t)DIV_ROUND_UP((ANALOG_IN_SAMPLING_TIME_MIN_IN_NS *
(uint64_t)actualAdcOperationFreq), 1000000000ull);
```

(1) Post Processing Mode Setting.

(2) Range Detection Value Setting.

(3) Range Detection Interrupt Setting.

Initialize ADC Channel. See **Code Listing 5**.

Code Listing 25 Range Detection Settings

```
    adcChannelConfig.sampleTime = samplingCycle;
 :
    Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &adcChannelConfig);
 :
    /* Register ADC interrupt handler and enable interrupt */
    cy_stc_sysint_irq_t irq_cfg;
    irq_cfg = (cy_stc_sysint_irq_t){
            .sysIntSrc  = (cy_en_intr_t)((uint32_t)CY_ADC_POT_IRQN + ADC_LOGICAL_CHANNEL),
            .intIdx     = CPUIntIdx3_IRQn,
            .isEnabled  = true,
    };
    Cy_SysInt_InitIRQ(&irq_cfg);
 :
    /* Enable ADC ch. */
    Cy_Adc_Channel_Enable(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);
    /* Issue SW trigger */
    Cy_Adc_Channel_SoftwareTrigger(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);
:

    for(;;);
}
```

Initialize Interrupt Request. See **Code Listing 7**.

Enable ADC Channel. See **Code Listing 8**.

Software Trigger Setting. See **Code Listing 9**.

6.2 A/D Conversion ISR for Range Detection

Figure 13 shows the example of ADC ISR for a range detection. For details on CPU interrupt handing, see the Architecture TRM mentioned in **Related Documents**.

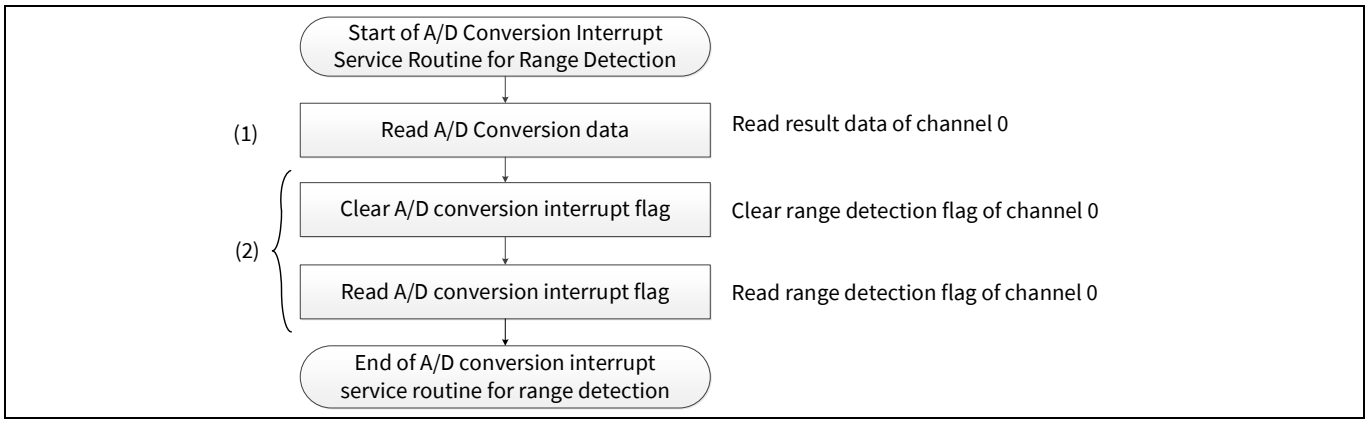


Figure 13 Example of ADC ISR for Range Detection

6.2.1 Configuration

Table 19 lists the parameters and Table 20 lists the functions of the configuration part of in SDL for ADC ISR for Range Detection settings.

Table 19 List of ADC ISR for Range Detection Settings Parameters

Parameters	Description	Value
`BB_POTI_ANALOG_MACRO`	Analog macro for the potentiometer on Traveo II Base Board	CY_ADC_POT_MACRO: PASS0_SAR0
`intrSource`	Interrupt Source	- (Calculated Value)
`resultIdx`	Index Result	- (Calculated Value)

Table 20 List of ADC ISR for Range Detection Settings Functions

Functions	Description	Value
`Cy_Adc_Channel_GetInterruptMaskedStatus(PASS SARchannel,INTR Source)`	Return the interrupt status	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] INTR Source = &intrSource
`Cy_Adc_Channel_GetResult(SAR Channel,Status)`	Get the conversion result and status.	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Status = resultBuff[resultIdx]
`Cy_Adc_Channel_ClearInterruptStatus(SAR Channel,Source)`	Clear the corresponding channel interrupt status.	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Source = intrSource

6.2.2 Sample Code

See **Code Listing 26** for sample code for initial configuration of ADC ISR for range detection settings.

Code Listing 26 ADC ISR for Range Detection Settings

```
:
/* ADC Interrupt Hanlder */
void AdcIntHandler(void)
{
    cy_stc_adc_interrupt_source_t intrSource = { false };

    /* Get interrupt source */
    Cy_Adc_Channel_GetInterruptMaskedStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&intrSource);

    if(intrSource.chRange)
    {
        /* Get the result(s) */
        Cy_Adc_Channel_GetResult(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&resultBuff[resultIdx], &statusBuff[resultIdx]);

        /* Increment result idx */
        resultIdx = (resultIdx + 1) % (sizeof(resultBuff) / sizeof(resultBuff[0]));

        /* Clear interrupt source */
        Cy_Adc_Channel_ClearInterruptStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&intrSource);

    }
    else
    {
        /* Unexpected interrupt */
        CY_ASSERT(false);
    }
}
:
```

Get Interrupt Masked Status. See **Code Listing 14**.

(1) Read A/D conversion data. See Code Listing 11.

(2) Clear and read A/D conversion flag. See Code Listing 12.

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 the range detection.

This function is usually used to avoid misdetections of abnormal voltage caused by noise and so on when monitoring voltage.

For parameters described in the **Range Detection Settings**, an analog-to-digital conversion result greater than or equal to 2500 or lesser than 1500 is a positive event; one greater than or equal to 1500 and lesser 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 14**.

The continuous number of positive event is cancelled when negative event occurs twice in a row, that is, the count of positive event continues even if negative event has occurred just once as shown in **Figure 15**.

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 Settings** and **Logical Channel Setting with Hardware Trigger** sections.

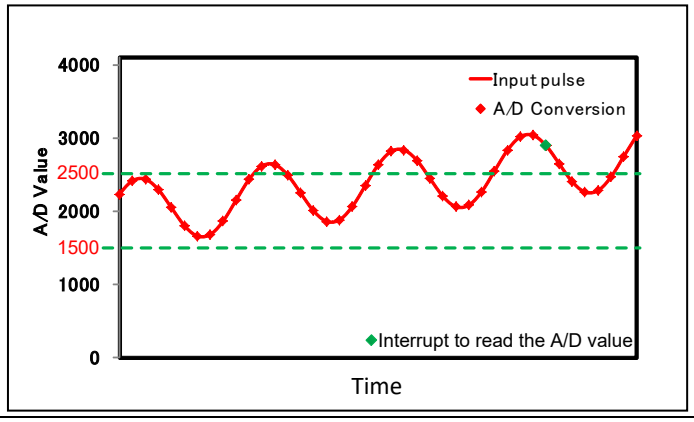


Figure 14 Example Behavior of Pulse Detection Application 1

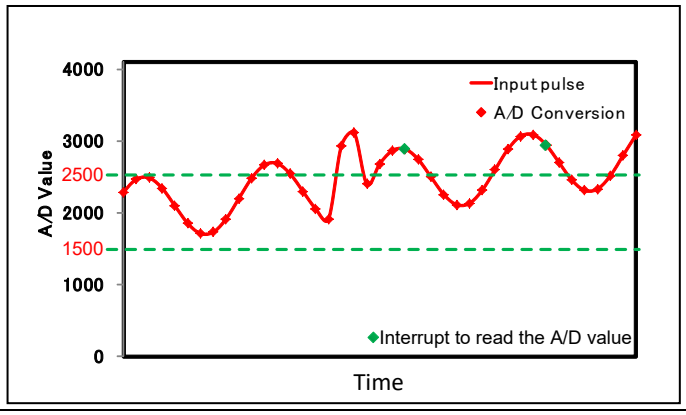


Figure 15 Example Behavior of Pulse Detection Application 2

Pulse Detection Procedure

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

7.1 Pulse Detection Settings

Figure 16 shows an example of pulse detection setting.

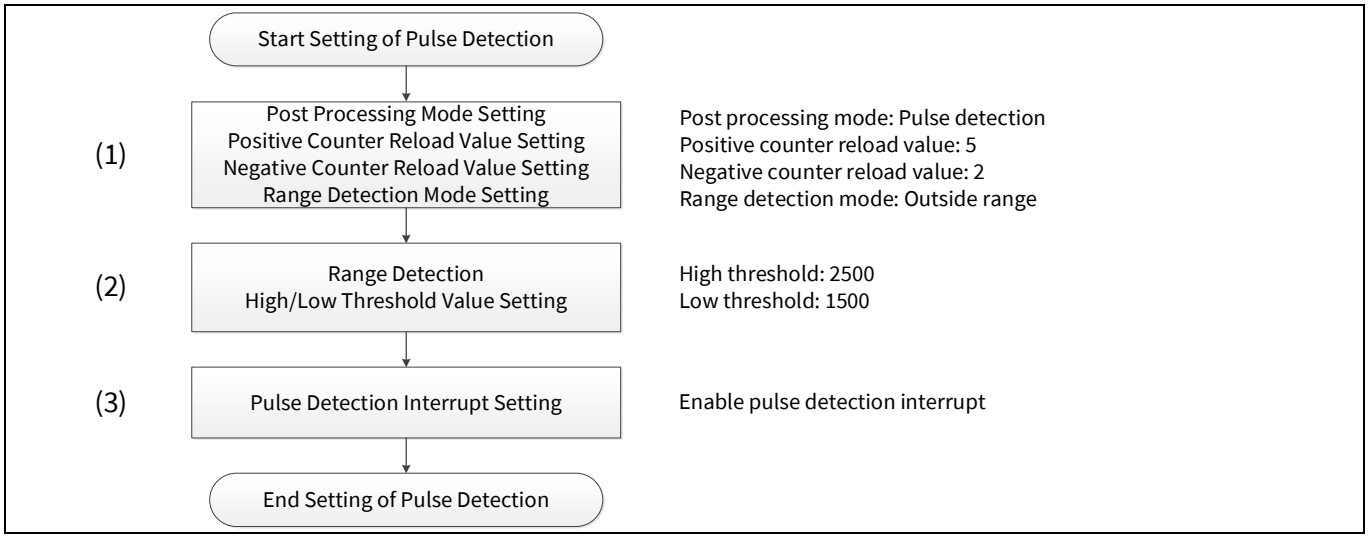


Figure 16 Example of Pulse Detection Setting

7.1.1 Use Case

The following use case is an example of Pulse Detection Settings.

- 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 21 lists the parameters and **Table 22** lists the functions of the configuration part of in SDL for Pulse Detection Settings.

Table 21 List of Pulse Detection Settings Parameters

Parameters	Description	Value
`BB_POTI_ANALOG_MACRO`	Analog macro for the potentiometer on Traveo II Base Board	CY_ADC_POT_MACRO: PASS0_SAR0
`BB_POTI_ANALOG_INPUT_NO`	Analog Input Number for the potentiometer on Traveo II Base Board	CH0 (CH[ADC_LOGICAL_CHANNEL])
`adcChannelConfig.triggerSelection`	Trigger TCPWM	1ul (See **Table 3**)
`adcChannelConfig.channelPriority`	Channel Priority	0ul

Pulse Detection Procedure

Parameters	Description	Value
adcChannelConfig.preenptionType	Pre-emption Type	3ul (See **Table 3**)
adcChannelConfig.isGroupEnd	Is Group End?	true
adcChannelConfig.doneLevel	Done Level	0ul (See **Table 3**)
adcChannelConfig.pinAddress	Pin Address	BB_POTI_ANALOG_INPUT_NO
adcChannelConfig.portAddress	Port Address	0ul (See **Table 3**)
adcChannelConfig.extMuxSelect	External MUX Select	0ul
adcChannelConfig.extMuxEnable	Enable External MUX	true
adcChannelConfig.preconditionMode	Pre-condition Mode	0ul (See **Table 3**)
adcChannelConfig.overlapDiagMode	Overlap Diagnostics Mode	0ul (See **Table 3**)
adcChannelConfig.sampleTime	Sample Time	0ul
adcChannelConfig.calibrationValueSelect	Calibration Value Select	0ul (See **Table 3**)
adcChannelConfig.postProcessingMode	Post Processing Mode	4ul (See **Table 3**)
adcChannelConfig.resultAlignment	Result Alignment	0ul (See **Table 3**)
adcChannelConfig.signExtention	Sign Extension	0ul (See **Table 3**)
adcChannelConfig.averageCount	Average Count is the Positive Counter Reload Value in Pulse Detection Mode	5ul
adcChannelConfig.rightShift	Right Shift is the Negative Counter Reload Value in Pulse Detection Mode	2ul
adcChannelConfig.rangeDetectionMode	Range Detection Mode	3ul (See **Table 3**)
adcChannelConfig.rangeDetectionLoThreshold	Range Detection Low Threshold	0x05DCul
adcChannelConfig.rangeDetectionHiThreshold	Range Detection High Threshold	0x09C4ul
adcChannelConfig.mask.grpDone	Mask Group Done	false
adcChannelConfig.mask.grpCancelled	Mask Group Cancelled	false
adcChannelConfig.mask.grpOverflow	Mask Group Overflow	false
adcChannelConfig.mask.chRange	Mask Channel Range	false
adcChannelConfig.mask.chPulse	Mask Channel Pulse	true
adcChannelConfig.mask.chOverflow	Mask Channel Overflow	false

Table 22 List of Pulse Detection Settings Functions

Functions	Description	Value
Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)	Initialize the ADC channel	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig

Pulse Detection Procedure

Functions	Description	Value
`Cy_SysInt_InitIRQ(Config)`	Initialize the referenced system interrupt by setting the interrupt vector	Config = irq_cfg
`Cy_Adc_Channel_Enable(PASS SARchannel)`	Enable the corresponding channel	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]
`Cy_Tcpwm_Pwm_Enable(Group Counter)`	De-initialize the TCPWM block	Group Counter = TCPWMx_GRPx_CNTx_PWM
`Cy_Tcpwm_TriggerStart(Group Counter)`	Trigger a software start on the selected TCPWMs	Group Counter = TCPWMx_GRPx_CNTx_PWM

7.1.3 Sample Code

See **Code Listing 27**.

Code Listing 27 Pulse Detection Settings

```
 :
#define BB_POTI_ANALOG_INPUT_NO     0
:
/* ADC Configuration */
:
cy_stc_adc_channel_config_t adcChannelConfig =
{
    .triggerSelection          = CY_ADC_TRIGGER_TCPWM,
    .channelPriority           = 0ul,
    .preenptionType            = CY_ADC_PREEMPTION_FINISH_RESUME,
    .isGroupEnd                = true,
    .doneLevel                 = CY_ADC_DONE_LEVEL_PULSE,
    .pinAddress                = BB_POTI_ANALOG_INPUT_NO,
    .portAddress               = CY_ADC_PORT_ADDRESS_SARMUX0,
    .extMuxSelect              = 0ul,
    .extMuxEnable              = true,
    .preconditionMode          = CY_ADC_PRECONDITION_MODE_OFF,
    .overlapDiagMode           = CY_ADC_OVERLAP_DIAG_MODE_OFF,
    .sampleTime                = 0ul,
    .calibrationValueSelect    = CY_ADC_CALIBRATION_VALUE_REGULAR,
    .postProcessingMode        = CY_ADC_POST_PROCESSING_MODE_RANGE_PULSE, /* Pulse Mode setting */
    .resultAlignment           = CY_ADC_RESULT_ALIGNMENT_RIGHT,
    .signExtention             = CY_ADC_SIGN_EXTENTION_UNSIGNED,
    .averageCount              = 5ul, /* Positive counter reload value */
    .rightShift                = 2ul, /* Negative counter reload value */
    .rangeDetectionMode        = CY_ADC_RANGE_DETECTION_MODE_OUTSIDE_RANGE, /* Outside Mode */
    .rangeDetectionLoThreshold = 0x05DCul, /* 1500 */
    .rangeDetectionHiThreshold = 0x09C4ul, /* 2500 */
    .mask.grpDone              = false,
    .mask.grpCancelled         = false,
```

(1) Post Processing Mode Setting.

(2) Range Detection Value Setting.

Pulse Detection Procedure

Code Listing 27 Pulse Detection Settings

```
    .mask.grpOverflow          = false,
    .mask.chRange              = false,
    .mask.chPulse              = true, /* Enable pulse detection interrupt */
    .mask.chOverflow           = false,
};
:
int main(void)
{
    SystemInit();

    __enable_irq(); /* Enable global interrupts. */
:
    /* Initialize ADC */
    uint32_t samplingCycle = (uint32_t)DIV_ROUND_UP((ANALOG_IN_SAMPLING_TIME_MIN_IN_NS *
(uint64_t)actualAdcOperationFreq), 1000000000ull);
    adcChannelConfig.sampleTime = samplingCycle;
:
    Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &adcChannelConfig);

    /* Register ADC interrupt handler and enable interrupt */
    cy_stc_sysint_irq_t irq_cfg;
    irq_cfg = (cy_stc_sysint_irq_t){
            .sysIntSrc  = pass_0_interrupts_sar_0_IRQn,
            .intIdx     = CPUIntIdx3_IRQn,
            .isEnabled  = true,
        };
    Cy_SysInt_InitIRQ(&irq_cfg);
:
    /* Clock Configuration for TCPWMs */
    uint32_t sourceFreq = 80000000ul;
    uint32_t targetFreq = 2000000ul;
    uint32_t divNum_PWM = (sourceFreq / targetFreq);
    CY_ASSERT((sourceFreq % targetFreq) == 0ul); // targe
:
    /* Enable ADC ch. and PWM */
    Cy_Adc_Channel_Enable(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);
    Cy_Tcpwm_Pwm_Enable(TCPWMx_GRPx_CNTx_PWM);
    Cy_Tcpwm_TriggerStart(TCPWMx_GRPx_CNTx_PWM);
:
    for(;;);
}
```

(3) Pulse Detection Interrupt Setting.

ADC Channel Initialize. See **Code Listing 5**.

Initialize Interrupt Request. See **Code Listing 7**.

Enable ADC Channel. See **Code Listing 8**.

TCPWM PWM Enable. See **Code Listing 18**.

TCPWM Trigger Start Select. See **Code Listing 19**.

7.2 A/D Conversion ISR for Pulse Detection

Figure 17 shows the example of ADC ISR for a pulse detection. For details on CPU interrupt handling, see the Architecture Technical Reference Manual mentioned in **Related Documents**.

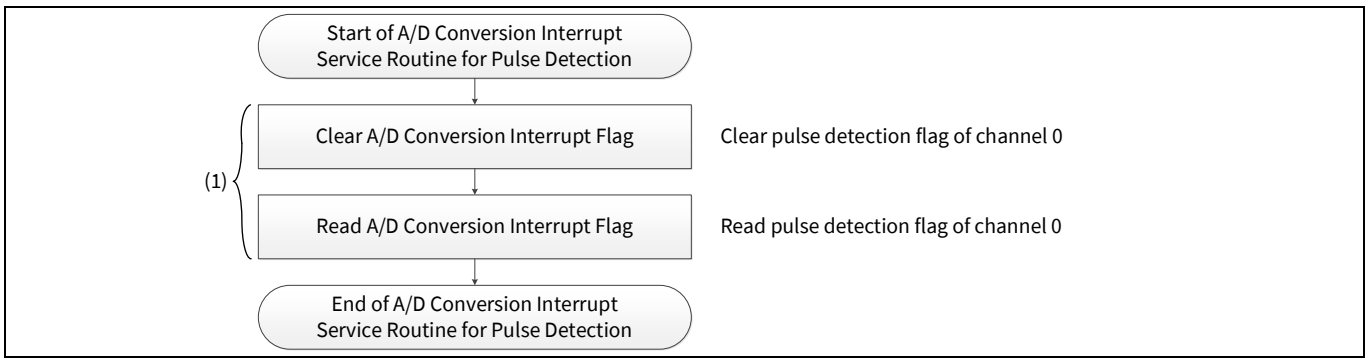


Figure 17 Example of ADC ISR for Pulse Detection

7.2.1 Configuration

Table 23 lists the parameters and **Table 24** lists the functions of the configuration part of in SDL for ADC Global settings.

Table 23 List of ADC ISR for Pulse Detection Settings Parameters

Parameters	Description	Value
`BB_POTI_ANALOG_MACRO`	Analog macro for the potentiometer on Traveo II Base Board	CY_ADC_POT_MACRO: PASS0_SAR0
`intrSource`	Interrupt Source	- (Calculated Value)
`resultIdx`	Index Result	- (Calculated Value)

Table 24 List of ADC ISR for Pulse Detection Settings Functions

Functions	Description	Value
`Cy_Adc_Channel_GetInterruptMaskedStatus(PASS SARchannel,INTR Source)`	Return the interrupt status	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] INTR Source = &intrSource
`Cy_Adc_Channel_ClearInterruptStatus(SAR Channel,Source)`	Clear the corresponding channel interrupt status	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Source = intrSource

7.2.2 Sample Code

See **Code Listing 28** for initial configuration of ADC ISR for pulse detection settings.

Code Listing 28 ADC ISR for Pulse Detection Settings

```
:
/* ADC Interrupt Hanlder */
void AdcIntHandler(void)
{
    cy_stc_adc_interrupt_source_t intrSource = { false };

    /* Get interrupt source */
    Cy_Adc_Channel_GetInterruptMaskedStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&intrSource);

    if(intrSource.chPulse)
    {
:
        /* Clear interrupt source */
        Cy_Adc_Channel_ClearInterruptStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&intrSource);
    }
    else
    {

        CY_ASSERT(false);
    }
}
```

Get Interrupt Masked Status. See **Code Listing 14**.

(1) Clear and read A/D conversion flag. See Code Listing 12.

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 18**.

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.

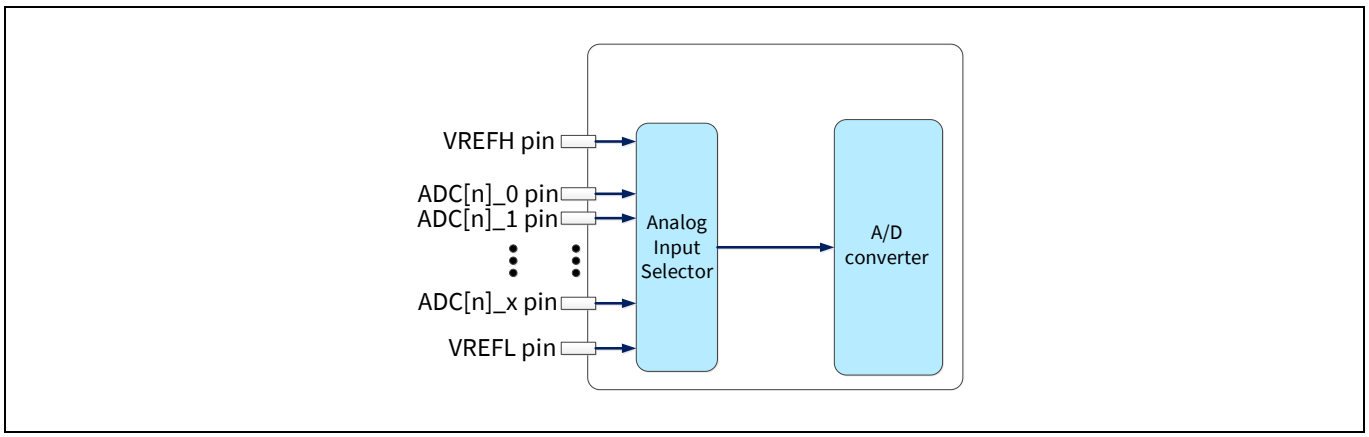


Figure 18 VREFH and VREFL as Analog Input

8.1 Check VZT and VFST

To conduct ADC diagnosis, see the datasheet mentioned in **Related Documents** 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 19 shows the example of ADC diagnosis flowchart. 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 **Related Documents**.

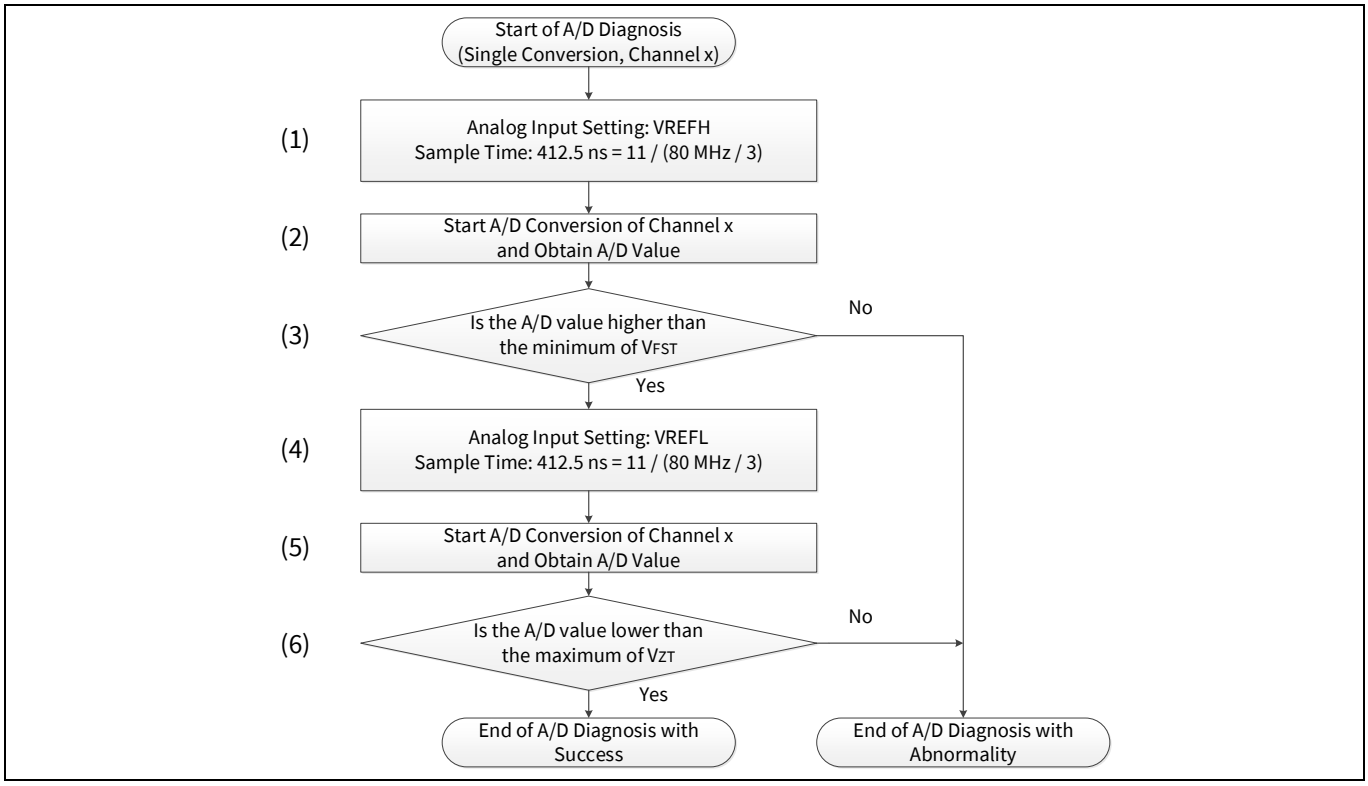


Figure 19 Example of ADC Diagnosis Flowchart

8.2.1 Use Case

The following use case is an example of ADC Diagnosis Settings.

- Analog Input Setting:
  - VREFH
  - VREFL
- Sample Time: 412.5 ns

8.2.2 Configuration

Table 25 lists the parameters and **Table 26** lists the functions of the configuration part of in SDL for ADC Diagnosis settings.

Table 25 List of ADC Diagnosis Settings Parameters

Parameters	Description	Value
`adcChannelConfig.triggerSelection`	Trigger OFF	0ul (See **Table 3**)
`adcChannelConfig.channelPriority`	Channel Priority	0ul
`adcChannelConfig.preenptionType`	Pre-emption Type	3ul (See **Table 3**)
`adcChannelConfig.isGroupEnd`	Is Group End?	true
`adcChannelConfig.doneLevel`	Done Level	0ul (See **Table 3**)
`adcChannelConfig.pinAddress`	Pin Address	When setting VREFH: CY_ADC_PIN_ADDRESS_VREF_H

Diagnosis Function

Parameters	Description	Value
		When setting VREFL: CY_ADC_PIN_ADDRESS_VREF_L
adcChannelConfig.portAddress	Port Address	0ul (See **Table 3**)
adcChannelConfig.extMuxSelect	External MUX Select	0ul
adcChannelConfig.extMuxEnable	Enable External MUX	false
adcChannelConfig.preconditionMode	Pre-condition Mode	0ul (See **Table 3**)
adcChannelConfig.overlapDiagMode	Overlap Diagnostics Mode	0ul (See **Table 3**)
adcChannelConfig.sampleTime	Sample Time	0ul
adcChannelConfig.calibrationValueSelect	Calibration Value Select	0ul (See **Table 3**)
adcChannelConfig.postProcessingMode	Post Processing Mode	0ul (See **Table 3**)
adcChannelConfig.resultAlignment	Result Alignment	0ul (See **Table 3**)
adcChannelConfig.signExtention	Sign Extension	0ul (See **Table 3**)
adcChannelConfig.averageCount	Average Count	0ul
adcChannelConfig.rightShift	Right Shift	0ul
adcChannelConfig.rangeDetectionMode	Range Detection Mode	1ul (See **Table 3**)
adcChannelConfig.rangeDetectionLoThreshold	Range Detection Low Threshold	0ul
adcChannelConfig.rangeDetectionHiThreshold	Range Detection High Threshold	0ul
adcChannelConfig.mask.grpDone	Mask Group Done	false
adcChannelConfig.mask.grpCancelled	Mask Group Cancelled	false
adcChannelConfig.mask.grpOverflow	Mask Group Overflow	false
adcChannelConfig.mask.chRange	Mask Channel Range	false
adcChannelConfig.mask.chPulse	Mask Channel Pulse	false
adcChannelConfig.mask.chOverflow	Mask Channel Overflow	false

Table 26 List of ADC Diagnosis Settings Functions

Functions	Description	Value
Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)	Initialize the ADC channel	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig
Cy_Adc_Channel_Enable(PASS SARchannel)	Enable the corresponding channel	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]
Cy_Adc_Channel_SoftwareTrigger(PASS SARchannel)	Issue a software start trigger	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

Diagnosis Function

Functions	Description	Value
`Cy_Adc_Channel_GetGroupStatus(PASS SARchannel, GroupStatus)`	Return the group conversion status	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] GroupStatus = false
`Cy_Adc_Channel_GetResult(SAR Channel,Status)`	Get the conversion result and status	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Status = resultBuff[resultIdx]

8.2.3 Sample Code

See **Code Listing 29** to **Code Listing 30** for sample code for for initial configuration of ADC diagnosis settings.

Code Listing 29 ADC Diagnosis Settings

```
:
/* ADC logical channel to be used */
:
#define BB_POTI_ANALOG_PCLK          CY_ADC_POT_PCLK
#define BB_POTI_ANALOG_INPUT_NO     ((cy_en_adc_pin_address_t)CY_ADC_POT_IN_NO)
:
/* ADC Channel Configuration */
cy_stc_adc_channel_config_t adcChannelConfig =
{
    .triggerSelection          = CY_ADC_TRIGGER_OFF,
    .channelPriority           = 0ul,
    .preenptionType            = CY_ADC_PREEMPTION_FINISH_RESUME,
    .isGroupEnd                = true,
    .doneLevel                 = CY_ADC_DONE_LEVEL_PULSE,
    .pinAddress                = CY_ADC_PIN_ADDRESS_VREF_H, /* Analog input setting */
    .portAddress               = CY_ADC_PORT_ADDRESS_SARMUX0,
    .extMuxSelect              = 0ul,
    .extMuxEnable              = false,
    .preconditionMode          = CY_ADC_PRECONDITION_MODE_OFF,
    .overlapDiagMode           = CY_ADC_OVERLAP_DIAG_MODE_OFF,
    .sampleTime                = 0ul,
    .calibrationValueSelect    = CY_ADC_CALIBRATION_VALUE_REGULAR,
    .postProcessingMode        = CY_ADC_POST_PROCESSING_MODE_NONE,
    .resultAlignment           = CY_ADC_RESULT_ALIGNMENT_RIGHT,
    .signExtention             = CY_ADC_SIGN_EXTENTION_UNSIGNED,
    .averageCount              = 0ul,
    .rightShift                = 0ul,
    .rangeDetectionMode        = CY_ADC_RANGE_DETECTION_MODE_INSIDE_RANGE,
    .rangeDetectionLoThreshold = 0ul,
    .rangeDetectionHiThreshold = 0ul,
    .mask.grpDone              = false,
    .mask.grpCancelled         = false,
    .mask.grpOverflow          = false,
```

(1) Analog Input Setting = VREFH.

Diagnosis Function

Code Listing 29 ADC Diagnosis Settings

```
    .mask.chRange               = false,
    .mask.chPulse               = false,
    .mask.chOverflow            = false,
};
:
/* Code Example for ADC Diagnosis  */

int main(void)
{
    SystemInit();

    __enable_irq(); /* Enable global interrupts. */
:
    /* Initialize ADC */
    uint32_t samplingCycle = (uint32_t)DIV_ROUND_UP((ANALOG_IN_SAMPLING_TIME_MIN_IN_NS *
(uint64_t)actualAdcOperationFreq), 1000000000ull);
    adcChannelConfig.sampleTime = samplingCycle;

:
    Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &adcChannelConfig);

    /* Enable ADC ch. */
    Cy_Adc_Channel_Enable(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);
    /* Issue SW trigger */
    Cy_Adc_Channel_SoftwareTrigger(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);


    for(;;)
    {
        status = DiagnosisProcedure();
        if(Test_Completed == 1 && status == CY_SUCCESS)
        {
            while(1);
        }
    }

}
uint32_t DiagnosisProcedure(void)
{
    status = CY_BAD_PARAM;

    /* Wait Group Done */
    cy_stc_adc_group_status_t Groupstatus = { fa
    Cy_Adc_Channel_GetGroupStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],&Groupstatus);
    while(Groupstatus.grpComplete == false);

    /* Get the result(s) */
```

Initialize ADC Channel. See **Code Listing 5**.

(2) Enable ADC Channel. See **Code Listing 8**.

Software Trigger Setting. See **Code Listing 9**.

Returns group conversion status. See **Code Listing 30**.

Diagnosis Function

Code Listing 29 ADC Diagnosis Settings

```
    Cy_Adc_Channel_GetResult(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &resultBuff[resultIdx],
&statusBuff[resultIdx]);

    /* Get the AD value */
    uint16_t AD_Value = resultBuff[resultIdx];

    /* Increment result idx */
    resultIdx = (resultIdx + 1) % (sizeof(resultBuff) / sizeof(resultBuff[0]));

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

            status = CY_SUCCESS;
            return(status);
        }
        else
        {
            /*AD Value Higher than the Maximum of Vzt */
            return(CY_BAD_PARAM);
        }
    }

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

        adcChannelConfig.pinAddress = CY_ADC_PIN_ADDRESS_VREF_L;

        Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC
        /* Enable ADC ch. */
        Cy_Adc_Channel_Enable(&BB_POTI_ANALOG_MACRO->CH[
        /* Trigger next conversion */
        Cy_Adc_Channel_SoftwareTrigger(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);
    }
    else
    {
        /* A/D value lower than the Minimum of Vfst */
        return(CY_BAD_PARAM);
    }
}
```

Read A/D conversion data. See **Code Listing 11**.

(6) Is the A/D value lower than the Maximum of VZT.

(3) Is the A/D value higher than the Minimum of VFST.

(4) Analog Input Setting = VREFL.

ADC Channel Initialize. See **Code Listing 5**.

(5) Enable ADC Channel. See **Code Listing 8**.

Software Trigger Setting. See **Code Listing 9**.

Diagnosis Function

Code Listing 30 Cy_Adc_Channel_GetGroupStatus() Function

```
cy_en_adc_status_t Cy_Adc_Channel_GetGroupStatus(const volatile stc_PASS_SAR_CH_t * base,
cy_stc_adc_group_status_t * status)
{
     cy_en_adc_status_t ret = CY_ADC_SUCCESS;
     un_PASS_SAR_CH_GRP_STAT_t unHwStat = { 0ul };

     if (NULL != status)
     {
         unHwStat.u32Register    = base->unGRP_STAT.u32Register;
         status->chOverflow      = (unHwStat.stcField.u1CH_OVERFLOW       != 0ul) ? true : false;
         status->chPulseComplete = (unHwStat.stcField.u1CH_PULSE_COMPLETE != 0ul) ? true : false;
         status->chRangeComplete = (unHwStat.stcField.u1CH_RANGE_COMPLETE != 0ul) ? true : false;
         status->grpBusy         = (unHwStat.stcField.u1GRP_BUSY          != 0ul) ? true : false;
         status->grpCancelled    = (unHwStat.stcField.u1GRP_CANCELLED     != 0ul) ? true : false;
         status->grpComplete     = (unHwStat.stcField.u1GRP_COMPLETE      != 0ul) ? true : false;
         status->grpOverflow     = (unHwStat.stcField.u1GRP_OVERFLOW      != 0ul) ? true : false;
    }
     else
     {
         ret = CY_ADC_BAD_PARAM;
     }
     return ret;
}
```

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 20**.

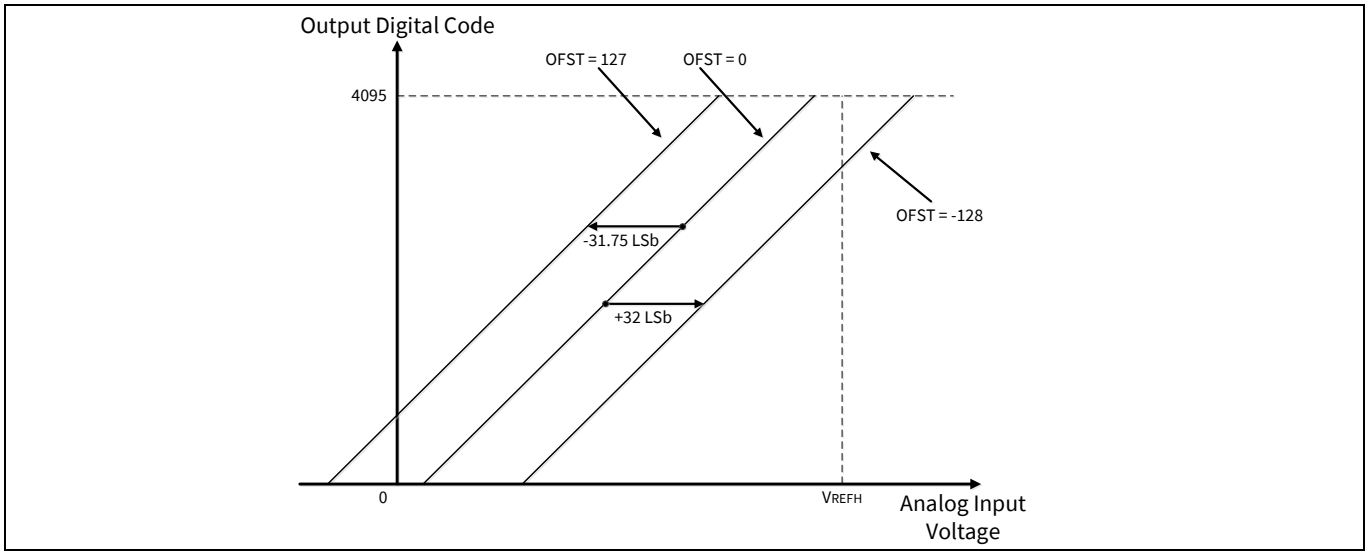


Figure 20 Relationship Between OFST and Offset Shift Direction

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]):

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

The relationship between the GAIN and the gain shift direction is shown in **Figure 21**.

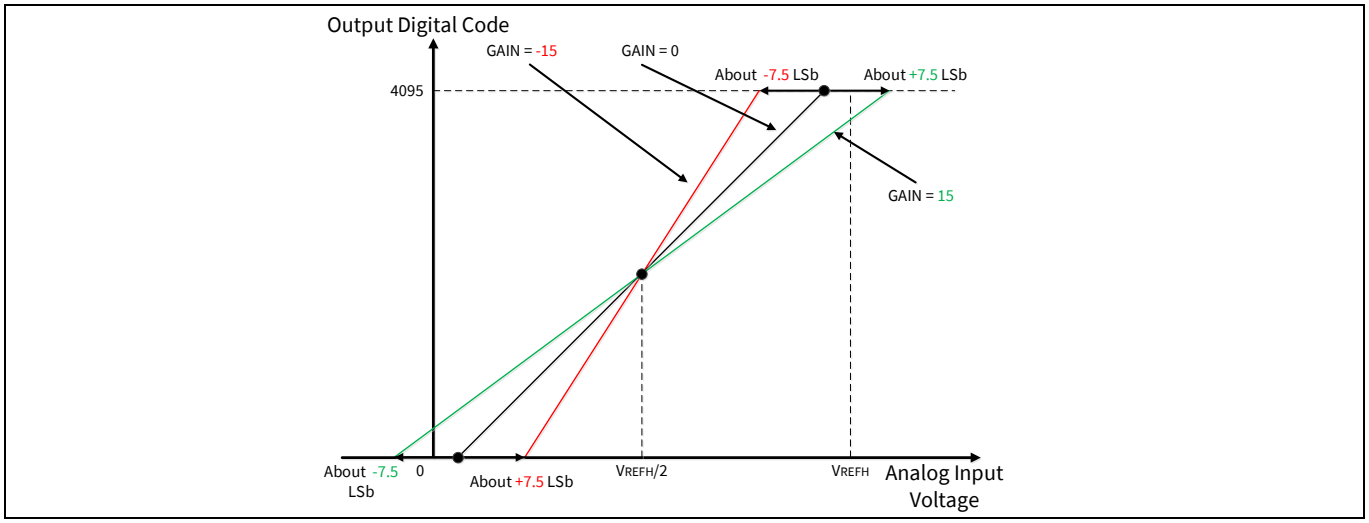


Figure 21 Relationship Between GAIN and Gain Shift Direction

9.3 Calibration Procedure

Figure 22 shows the example flowchart of ADC calibration. First adjust the offset, and next adjust the gain.

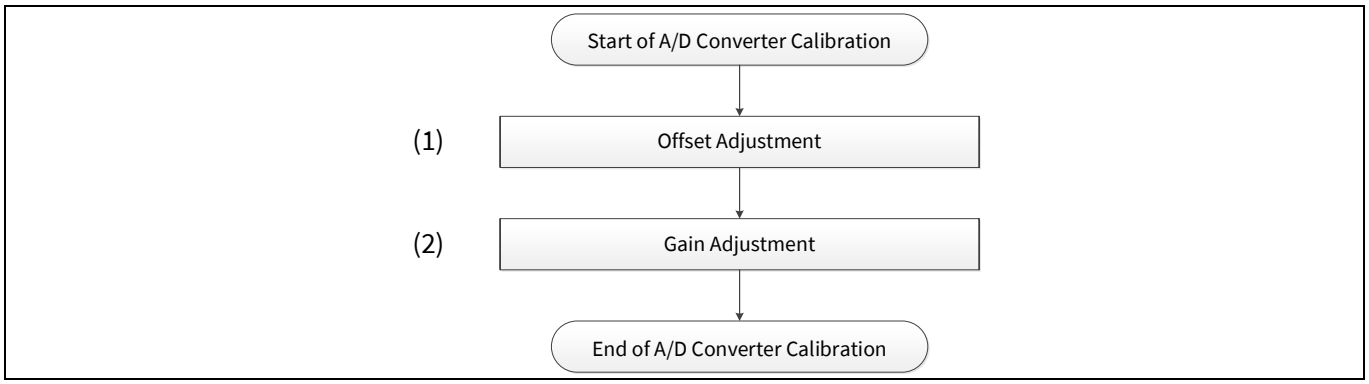


Figure 22 Example Flowchart of ADC Calibration

9.3.1 Configuration

Table 27 lists the parameters of the configuration part of in SDL for ADC Calibration Procedure settings.

Table 27 List of ADC Calibration Procedure Settings Parameters

Parameters	Description	Value
`.offset`	Offset calibration setting	127ul
`.gain`	Gain calibration setting	0ul

9.3.2 Sample Code

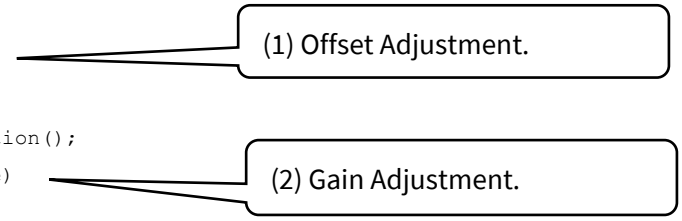
See **Code Listing 31** for a sample code for initial configuration of ADC calibration procedure settings.

Code Listing 31 ADC Calibration Procedure Settings

```
:
/* Calibration Setting */
cy_stc_adc_analog_calibration_conifg_t calibrationConfig =
{
    .offset = 127ul,/* Offset Calibration Setting */
    .gain   = 0ul,/* Gain Calibration Setting */
};
:
int main(void)
{
    SystemInit();

    __enable_irq(); /* Enable global interrupts. */

:
    if(Offset_Calibration() == true)
    {
        //Gain_Status = Gain_Calibration();
        if(Gain_Calibration() == true)
        {
            /* Test Completed */
            while(1);
        }
        else
        {
            /* Error */
            while(1);
        }
    }
    else
    {
        /* Error */
        while(1);
    }
}
```



9.4 Offset Adjustment Procedure

Figure 23 shows the example flowchart of offset adjustment.

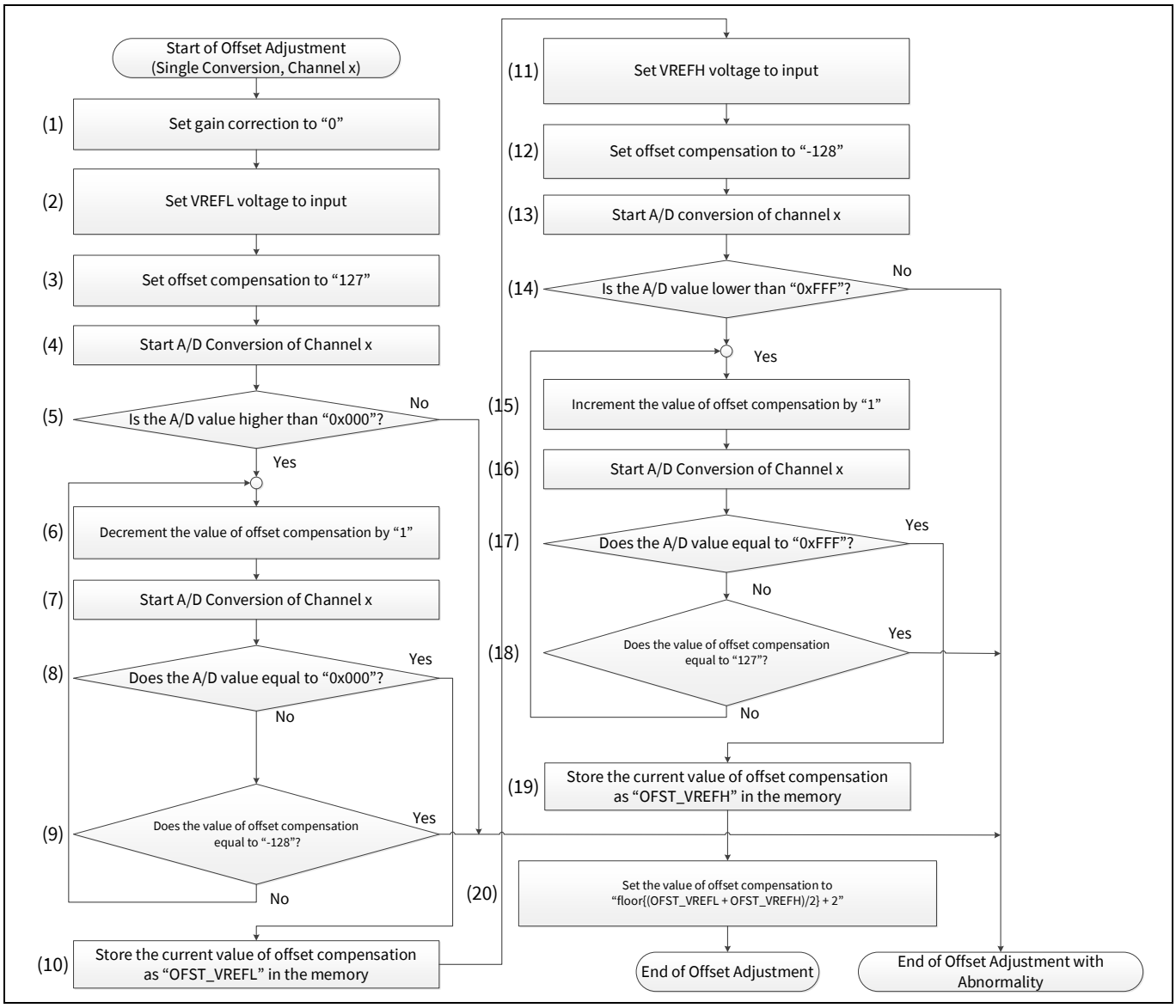


Figure 23 Example Flowchart of Offset Adjustment

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 28 lists the parameters and **Table 29** lists the functions of the configuration part of in SDL for Offset Adjustment Procedure settings.

Calibration Function

Table 28 List of Offset Adjustment Procedure Settings Parameters

Parameters	Description	Value
BB_POTI_ANALOG_MACRO	Analog macro for the potentiometer on Traveo II Base Board	CY_ADC_POT_MACRO: PASS0_SAR0
calibrationConfig.offset	Calibration Offset Value	-128 or more, 128 or less
resultBuff	Conversion Result Buffer	- (Calculated Value)

Table 29 List of Offset Adjustment Procedure Settings Functions

Functions	Description	Value
Cy_Adc_Channel_SoftwareTrigger(PASS SARchannel)	Issue a software start trigger	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]
Cy_Adc_Channel_GetGroupStatus(PASS SARchannel, GroupStatus)	Returns the group conversion status	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] GroupStatus = false
Cy_Adc_Channel_GetResult(SAR Channel,Status)	Get the conversion result and status	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Status = resultBuff[resultIdx]
Cy_Adc_SetAnalogCalibrationValue(PASS SARchannel,&calibrationConfig)	Set the analog calibration value	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] Calibration Configure = calibrationConfig
Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)	Initialize the ADC channel	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig
Cy_Adc_Channel_Enable(PASS SARchannel)	Enable the corresponding channel	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

9.4.3 Sample Code

See **Code Listing 32** to **Code Listing 33** for sample code snippets for initial configuration of offset adjustment procedure settings.

Code Listing 32 Offset Adjustment Procedure Settings

```
:
#define BB_POTI_ANALOG_MACRO        CY_ADC_POT_MACRO
```

Calibration Function

Code Listing 32 Offset Adjustment Procedure Settings

```
:
/* Calibration Setting */
cy_stc_adc_analog_calibration_conifg_t calibrationConfig =
{
    .offset = 127ul,/* Offset Calibration Setting */
    .gain   = 0ul,/* Gain Calibration Setting */
};
:
int main(void)
{
    SystemInit();

    __enable_irq(); /* Enable global interrupts. */

:
    /* Set Gain Coreection to 0" */
    calibrationConfig.gain = 0ul;
    /* Set VREFL Voltage to Input */
    adcChannelConfig.pinAddress = CY_ADC_PIN_ADDRESS_VREF_L;
    /* Set Offset Compensation to "127" */
    calibrationConfig.offset = 127ul;
:
     if(Offset_Calibration() == true)
    {
        //Gain_Status = Gain_Calibration();
        if(Gain_Calibration() == true)
        {
            /* Test Completed */
            while(1);
        }
        else
        {
            /* Error */
            while(1);
        }
    }
    else
    {
        /* Error */
        while(1);
    }
}


/*******************************************************************/
/* Function: GetAdcValue                                           */
/*******************************************************************/
uint16_t GetAdcValue(void)
```

(1) Set Gain Correction to “0”.

(2) Set VREFL Voltage to Input.

(3) Set offset compensation to “127”.

(4) Start A/D Conversion of Channel x.

Calibration Function

Code Listing 32 Offset Adjustment Procedure Settings

```
{
    /* trigger ADC */
    Cy_Adc_Channel_SoftwareTrigger(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);

    /* wait ADC completion */
    cy_stc_adc_group_status_t Groupstatus = { false };
    Cy_Adc_Channel_GetGroupStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],&Groupstatus);
    while(Groupstatus.grpComplete == false)
    {
        Cy_Adc_Channel_GetGroupStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],&Groupstatus);
    }
    /* Get the result(s) */
    Cy_Adc_Channel_GetResult(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &resultBuff[resultIdx],
&statusBuff[resultIdx]);
    /* Get the AD value */
    result = resultBuff[resultIdx];
    /* Increment result idx */
    resultIdx = (resultIdx + 1) % (sizeof(resultBuff) / sizeof(resultBuff[0]));

    return result ;
}

/*****************************************************************/
/* Function: OffsetCalibration                                   */
/*****************************************************************/
bool Offset_Calibration(void)
{
    /* Start A/D Conversion and get the A/D value */
    result = GetAdcValue();

    /* Is the A/D Value Higher than 0 ? */
    if(result <= 0)
    {
        /* Error */
        return false;
    }

    /* Decrment the Value of Offset Compensation by "1" */
    for(offset_VREFL = 127; offset_VREFL >= -128; offset_VREFL -= 1)
    {
        /* Set "offset_VREFL" to offset regist
        calibrationConfig.offset = offset_VREFL;
        Cy_Adc_SetAnalogCalibrationValue(BB_POTI_ANALOG_MACRO, &calibrationConfig);

        /* Get AD Value */
        result = GetAdcValue();

        /* Does the A/D value Equal to 0 ? */
```

Software Trigger Setting. See **Code Listing 9**.

Returns group conversion status. See **Code Listing 30**.

Read A/D conversion data. See **Code Listing 11**.

(5) Is the A/D value higher than “0x000” ?

(6) Decrement the value of offset compensation by “1”.

Analog Calibration Value Setting. See **Code Listing 33**.

(7) Start A/D Conversion of Channel x.

(8) Does the A/D value equal to “0x000” ?

Code Listing 32 Offset Adjustment Procedure Settings

```
        if(result == 0)
        {
            break;
        }
    }

    if(offset_VREFL == -129)
    {
        /* Error */
        return false;;
    }

    /*  Setting for VREFH */
    /* Set VREFH Voltage to Input */
    adcChannelConfig.pinAddress = CY_ADC_PIN_ADDRESS_VREF_H;
    /* Set Offset Compensation to "-128" */
    calibrationConfig.offset = -128;
    Cy_Adc_SetAnalogCalibrationValue(BB_POTI_ANALOG_MACRO, &calibrationConfig);
    Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &adcChannelConfig);
    /* Enable ADC ch. */
    Cy_Adc_Channel_Enable(&BB_POTI_ANALOG_MACRO->CH[ADC_LO
    /* Start A/D Conversion and get the A/D value */
    result = GetAdcValue();

    /* Is the AD value lower than 0xFFF */
    if(result > 0xFFF)
    {
        /* Error */
        return false;
    }

    /* Increment the Value of Offset Compensation by "1" */
    for(offset_VREFH = -128; offset_VREFH <= 127; offset_VREFH += 1)
    {
        /* Set "offset_VREFH" to offset regist value */
        calibrationConfig.offset = offset_VREFH;
        Cy_Adc_SetAnalogCalibrationValue(BB_POTI_ANALOG_MACRO, &calibrationConfig);

        /* Get AD Value */
        result = GetAdcValue();

        /* Does the A/D value Equal to 4095 ? */
        if(result == 0xFFF)
        {
            break;
        }
    }
```

(9) Does the value of offset compensation equal to “-128”?

(10) Store the current value of offset compensation as “OFST_VREFL” in the memory.

(11) Set VREFH voltage to input.

(12) Set offset compensation to “-128”.

Analog Calibration Value Setting. See **Code Listing 33**.

ADC Channel Initialize. See **Code Listing 5**.

Enable ADC Channel. See **Code Listing 8**.

(13) Start A/D Conversion of Channel x.

(14) Is the A/D value lower than “0xFFF”?.

(15) Increment the value of offset compensation by “1”

Analog Calibration Value Setting. See **Code Listing 33**.

(16) Start A/D Conversion of Channel x.

(17) Does the A/D value equal to “0xFFF” ?

(18) Does the value of offset compensation equal to “127”?

Calibration Function

Code Listing 32 Offset Adjustment Procedure Settings

```
    if(offset_VREFH == 128)
    {
        /* Error */
        return false;
    }

    calibrationConfig.offset = (floor((offset_VREFH+offset_VREFL)/2) + 2);

    return true;

}
```

(19) Store the current value of offset compensation as “OFST_VREFH” in the memory.

(20) Set the value of offset compensation to “floor{(OFST_VREFL + OFST_VREFH)/2} + 2”

Code Listing 33 Cy_Adc_SetAnalogCalibrationValue() Function

```
cy_en_adc_status_t Cy_Adc_SetAnalogCalibrationValue(volatile stc_PASS_SAR_t * base,
cy_stc_adc_analog_calibration_conifg_t * config)
{
    cy_en_adc_status_t    ret      = CY_ADC_SUCCESS;
    un_PASS_SAR_ANA_CAL_t unAnaCal = { 0ul };

    if (NULL != config)
    {
        unAnaCal.stcField.u8AOFFSET = config->offset;
        unAnaCal.stcField.u5AGAIN   = config->gain;
        base->unANA_CAL.u32Register = unAnaCal.u32Register;
    }
    else
    {
        ret = CY_ADC_BAD_PARAM;
    }
    return ret;
}
```

9.5 Gain Adjustment Procedure

Figure 24 shows the example flowchart of gain adjustment.

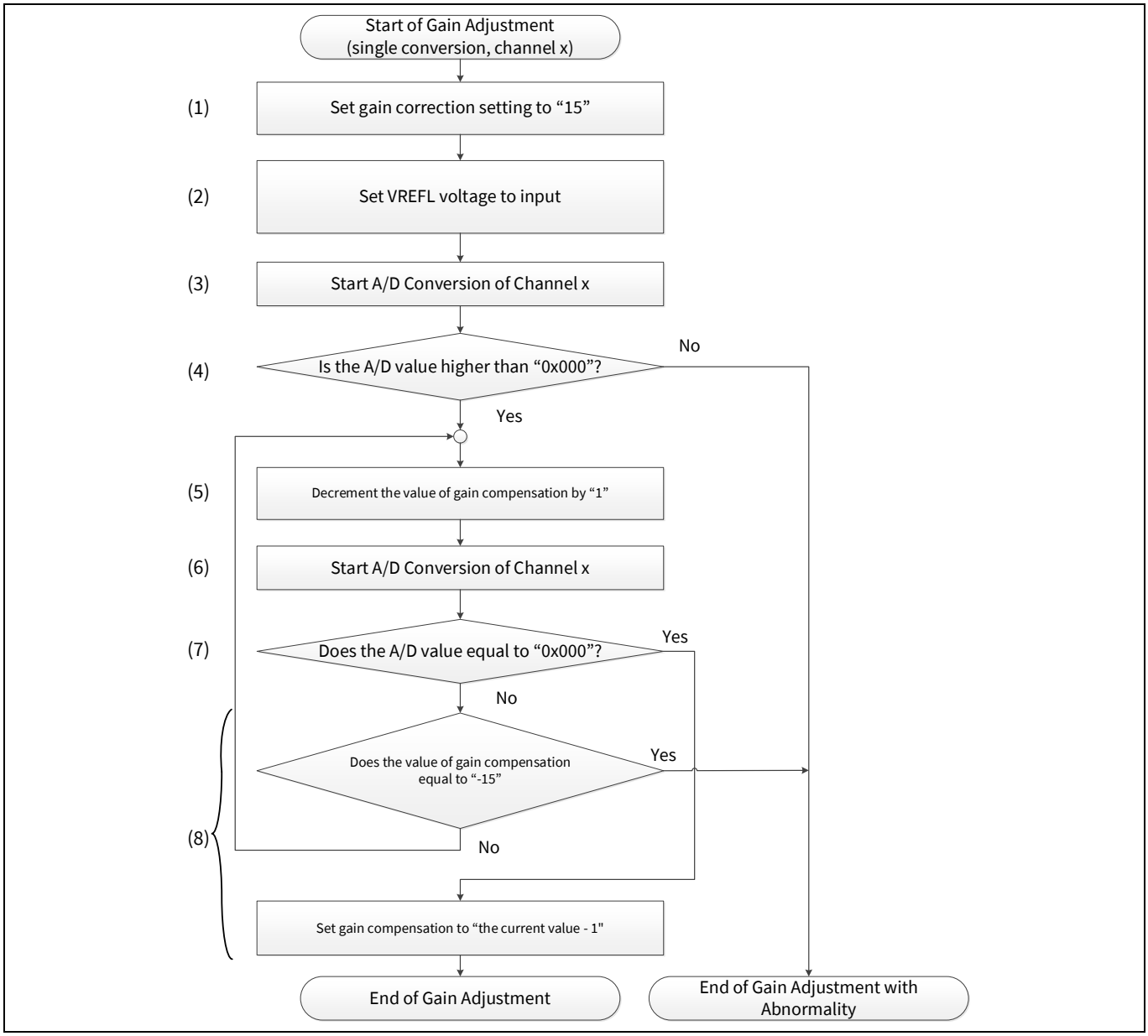


Figure 24 Example Flowchart of Gain Adjustment

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 30 lists the parameters and **Table 31** lists the functions of the configuration part of in SDL for ADC Global settings.

Table 30 List of Gain Adjustment Procedure Settings Parameters

Parameters	Description	Value
BB_POTI_ANALOG_MACRO	Analog macro for the potentiometer on Traveo II Base Board	CY_ADC_POT_MACRO: PASS0_SAR0

Calibration Function

Parameters	Description	Value
calibrationConfig.gain	Calibration Gain Value	-1 or more, 15 or less
resultBuff	Conversion Result Buffer	- (Calculated Value)

Table 31 List of Gain Adjustment Procedure Settings Functions

Functions	Description	Value
Cy_Adc_Channel_SoftwareTrigger(PASS SARchannel)	Issue a software start trigger	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]
Cy_Adc_Channel_GetGroupStatus(PASS SARchannel, GroupStatus)	Return the group conversion status	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] GroupStatus = false
Cy_Adc_Channel_GetResult(SAR Channel,Status)	Get the conversion result and status	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Status = resultBuff[resultIdx]
Cy_Adc_SetAnalogCalibrationValue(PASS SARchannel,&calibrationConfig)	Set the analog calibration value.	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] Calibration Configure = calibrationConfig
Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)	Initialize the ADC channel	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig
Cy_Adc_Channel_Enable(PASS SARchannel)	Enable the corresponding channel	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

9.5.3 Sample Code

See **Code Listing 34** for sample code for initial configuration of gain adjustment procedure settings

Code Listing 34 Gain Adjustment Procedure Settings

```
:
#define BB_POTI_ANALOG_MACRO        CY_ADC_POT_MACRO
:
/* Calibration Setting */
cy_stc_adc_analog_calibration_conifg_t calibrationConfig =
{
    .offset = 127ul,/* Offset Calibration Setting */
    .gain   = 0ul,/* Gain Calibration Setting */
};
:
```

Calibration Function

Code Listing 34 Gain Adjustment Procedure Settings

```
int main(void)
{
    SystemInit();

    __enable_irq(); /* Enable global interrupts. */

:
    if(Offset_Calibration() == true)
    {
        //Gain_Status = Gain_Calibration();
        if(Gain_Calibration() == true)
        {
            /* Test Completed */
            while(1);
        }
        else
        {
            /* Error */
            while(1);
        }
    }
    else
    {
        /* Error */
        while(1);
    }
}


/*******************************************************************/
/* Function: GetAdcValue                                           */
/*******************************************************************/
uint16_t GetAdcValue(void)
{
    /* trigger ADC */
    Cy_Adc_Channel_SoftwareTrigger(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);

    /* wait ADC completion */
    cy_stc_adc_group_status_t Groupstatus = { false };
    Cy_Adc_Channel_GetGroupStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],&Groupstatus);
    while(Groupstatus.grpComplete == false)
    {
        Cy_Adc_Channel_GetGroupStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],&Groupstatus);
    }
    /* Get the result(s) */
    Cy_Adc_Channel_GetResult(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &resultBuff[resultIdx],
&statusBuff[resultIdx]);
    /* Get the AD value */
```

Software Trigger Setting. See **Code Listing 9**.

Returns group conversion status. See **Code Listing 30**.

Read A/D conversion data. See **Code Listing 11**.

Calibration Function

Code Listing 34 Gain Adjustment Procedure Settings

```
    result = resultBuff[resultIdx];
    /* Increment result idx */
    resultIdx = (resultIdx + 1) % (sizeof(resultBuff) / sizeof(resultBuff[0]));

    return result ;
}

/*******************************************************************/
/* Function: Gain_Calibration                                      */
/*******************************************************************/
bool Gain_Calibration(void)
{
    int16_t Gain;
    /* Set Gain Correction setting to "15" */
    calibrationConfig.gain = 15;
    /* Set VREFL Voltage to Input */
    adcChannelConfig.pinAddress = CY_ADC_PIN_ADDRESS_VREF_L;
    Cy_Adc_SetAnalogCalibrationValue(BB_POTI_ANALOG_MACRO, &calibrationConfig);
    Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &adcChannelConfig);

    /* Enable ADC ch. */
    Cy_Adc_Channel_Enable(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);
    /* Start A/D value Conversion */
    result = GetAdcValue();

    if(result < 0)
    {
        /* Error */
        return false;
    }

    for(Gain = 15; Gain >=-15; Gain -= 1)
    {
        /* Set gain to register */
        calibrationConfig.gain = Gain;
        Cy_Adc_SetAnalogCalibrationValue(BB_POTI_ANALOG_MACRO, &calibrationConfig);

        result = GetAdcValue();
        if(result == 0)
        {
            /* Set gain compension to "gain" -1 */
            calibrationConfig.gain = Gain - 1;
            Cy_Adc_SetAnalogCalibrationValue(BB_POTI_ANALOG_MACRO, &calibrationConfig);

            break;
        }
```

(1) Set Gain Correction to “15”.

(2) Set VREFL Voltage to Input.

Analog Calibration Value Setting. See **Code Listing 33**.

Initialize ADC Chan nel. See **Code Listing 5**.

Enable ADC Channel. See **Code Listing 8**.

(3) Start A/D Conversion of Channel x.

(4) Is the A/D value higher than “0x000”?

Analog Calibration Value Setting. See **Code Listing 33**.

(6) Start A/D Conversion of Channel x.

(7) Does the A/D value equal to “0x000”?

(8) Set gain compensation to “the current value - 1".

Analog Calibration Value Setting. See **Code Listing 33**.

Calibration Function

Code Listing 34 Gain Adjustment Procedure Settings

```
        if( Gain == -16)
        {
            /* Error */
            return false;
        }
    }
    return true;
}
```

(8) Does the value of gain compensation equal to “-15”

10 Glossary

Terms	Description
SAR ADC	Successive Approximation Register 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

11 Related Documents

The following are the Traveo II family series datasheets and Technical Reference Manuals. Contact **Technical Support** to obtain these documents.

- Device datasheet
  - CYT2B7 Datasheet 32-Bit Arm® Cortex®-M4F Microcontroller Traveo™ II Family
  - CYT2B9 Datasheet 32-Bit Arm® Cortex®-M4F Microcontroller Traveo™ II Family
  - CYT4BF Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo™ II Family
  - CYT4DN Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo™ II Family
  - CYT3BB/4BB Datasheet 32-Bit Arm® Cortex®-M7 Microcontroller Traveo™ II Family
- Body Controller Entry Family
  - Traveo™ II Automotive Body Controller Entry Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B7
  - Traveo™ II Automotive Body Controller Entry Registers Technical Reference Manual (TRM) for CYT2B9
- Body Controller High Family
  - Traveo™ II Automotive Body Controller High Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Body Controller High Registers Technical Reference Manual (TRM) for CYT4BF
  - Traveo™ II Automotive Body Controller High Registers Technical Reference Manual (TRM) for CYT3BB/4BB
- Cluster 2D Family
  - Traveo™ II Automotive Cluster 2D Family Architecture Technical Reference Manual (TRM)
  - Traveo™ II Automotive Cluster 2D Registers Technical Reference Manual (TRM)

12 Other References

A Sample Driver Library (SDL) including startup as sample software to access various peripherals is provided. SDL also serves as a reference to customers for drivers that are not covered by the official AUTOSAR products. The SDL cannot be used for production purposes because it does not qualify to automotive standards. The code snippets in this application note are part of the SDL. Contact **Technical Support** to obtain the SDL.

Revision history

Document version	Date of release	Description of changes
**	03/09/2018	New application note.
*A	12/06/2018	Changed target part number (CYT2B series). Added Glossary section.
*B	02/28/2019	Added target part number (CYT4B series).
*C	10/02/2019	Added target part number (CYT4D series).
*D	03/02/2020	Changed target parts number (CYT2/ CYT4 series). Added target parts number (CYT3 series).
*E	2021-02-12	Added flowchart and example codes. Moved to Infineon template.

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

Edition 2021-02-12

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about this document?

Go to www.cypress.com/support

Document reference

002-19755 Rev. *E

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.

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

WARNINGS

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

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