TRAVEO™ T2G 車載マイクロコントローラーでの SAR ADC の使用

本書について

適用範囲と目的

このアプリケーションノートは、ソフトウェアトリガ, ハードウェアトリガ, グループ処理, 平均化, レンジ検出, パルス検出, 診断, およびキャリブレーションを使用して TRAVEO™ T2G 自動車用マイクロコントローラに搭載される SAR ADC の設定方法, 使い方について説明します。

対象者

このドキュメントは、TRAVEO™ T2G 自動車用マイクロコントローラの SAR ADC を使用するユーザーを対象とします。

関連製品ファミリ

TRAVEO™ T2G ファミリ

目次

目次

目次

1 はじめに

このアプリケーションノートでは、インフィニオン TRAVEO™ T2G 自動車用マイクロコントローラの SAR ADC の使い方を説明します。SAR ADC はアナログ入力電圧をデジタル値に変換します。アナログチャネルは個別またはグループとして使用できます。それぞれのチャネルは、ソフトウェアまたはハードウェアによって起動できます。SAR ADC には平均化, レンジ検出, パルス検出, 診断, キャリブレーションの機能があります。

一般的に、ADC 結果は、電源電圧, 基準電圧, 入力アナログ電圧, 温度, およびノイズなどの環境に影響されます。したがって、環境の影響を軽減するために、キャリブレーションと複数の ADC 結果の平均化を推奨します。

このアプリケーションノートで説明されている機能と使用されている用語を理解するためには、アーキテクチャ テクニカル リファレンス マニュアル (TRM) の「SAR ADC」セクションを参照してください。

2 ソフトウェアトリガ処理

図 1 に、MCU の ADC[0]_0 端子に入力された電圧をデジタル値に変換するアプリケーション例を示します。A/D 変換は、割込み処理ルーチン内でソフトウェアトリガされることで、繰り返し行われます。

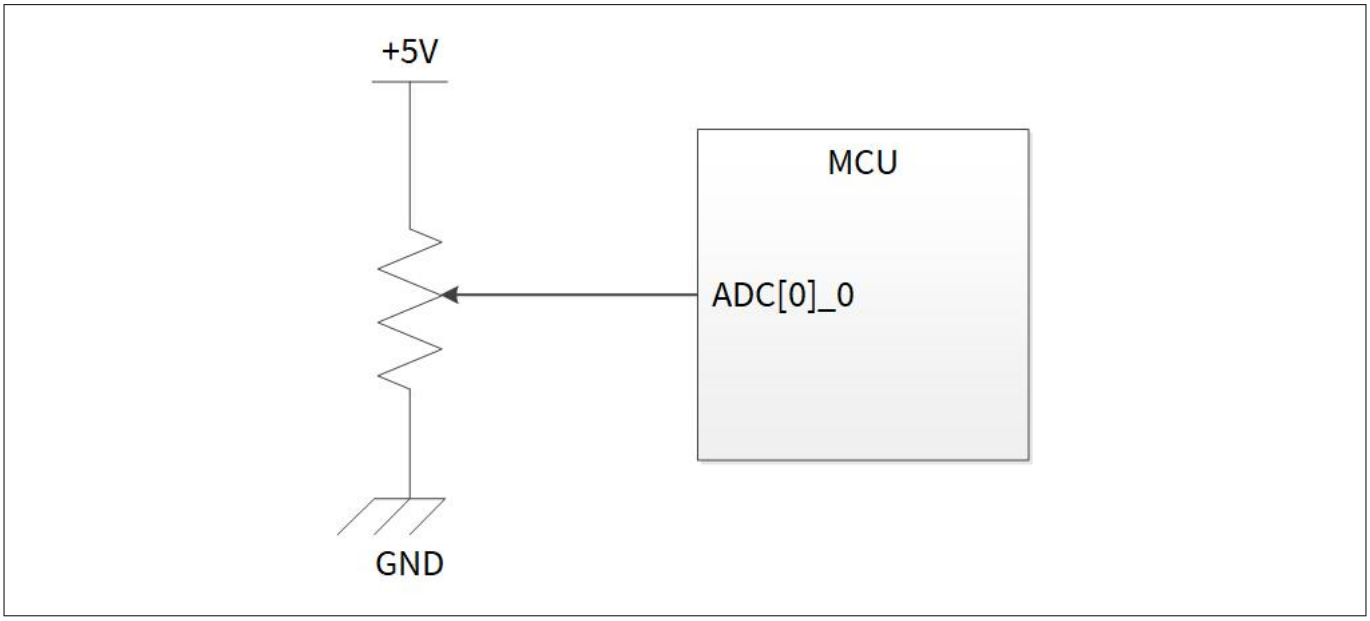


図 1 A/D 変換接続例

このアプリケーションを実装するためには、ADC チャネルの設定手順を記述する以下のセクションを活用してください。これらのセクションでは、ソフトウェアトリガの使用例も示します。

2.1 基本的な ADC グローバル設定

ここでは、サンプルドライバライブラリ (SDL) を使用して、ユースケースに基づいて ADC を設定する方法について説明します。このアプリケーションノートのコードは SDL の一部です。詳細については、その他の参考資料を参照してください。

SDL は、設定部とドライバ部があります。設定部は、主に目的の操作のためのパラメータ値を設定します。ドライバ部は、設定部のパラメータを各レジスタに設定します。システムに応じて設定部を変更できます。

2.2 ADC グローバル設定

以下は、各チャネル共通の ADC 設定手順です。

- SARn_CTL レジスタによる ADC イネーブルおよび Auto Idle Power Down 設定
- PASS_PASS_CTL レジスタによる Debug Freeze 設定

図 2 に、ADC グローバル設定例を示します。

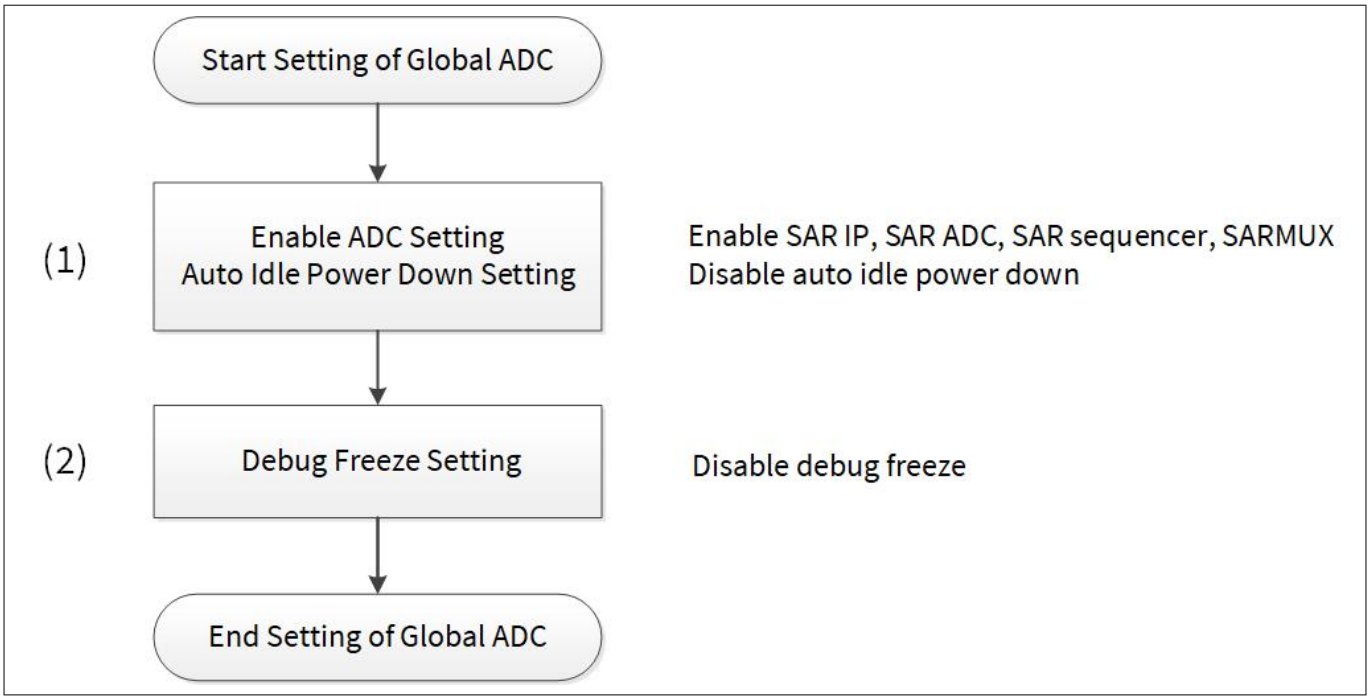


図 2　　ADC グローバル設定例

2.2.1　　ユースケース

以下のユースケースは ADC グローバル設定の例を示します。

- ADC ロジカルチャネル: 0
- ADC 動作周波数: 26.67 MHz
- オートアイドルパワーダウン: 0 (無効)
- パワーアップ時間: 0 (無効)
- SAR IP: 1 (有効)
- SAR ADC と SARSEQ: 1 (有効)
- SARMUX: 1 (有効)

2.2.2　　コンフィグレーション

A/D 変換グローバル設定の SDL の設定部のパラメータを表 1 に、関数を表 2 に示します。

表 1　　ADC グローバルパラメーター覧

パラメータ	説明	値
BB_POTI_ANALOG_MACRO	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログマクロ	CY_ADC_POT_MACRO: PASS0_SAR0
CY_ADC_BAD_PARAM	不正なパラメータを渡されました	0x01 ul
FreezeConfig.enableFreezeAdc0	デバッグモードで ADC0 にフリーズ	0 ul
FreezeConfig.enableFreezeAdc1	デバッグモードで ADC1 にフリーズ	0 ul
FreezeConfig.enableFreezeAdc2	デバッグモードで ADC2 にフリーズ	0 ul
FreezeConfig.enableFreezeAdc3	デバッグモードで ADC3 にフリーズ	0 ul

(続く)

表 1 (続き) ADC グローバルパラメーター覧

パラメータ	説明	値
adcConfig.preconditionTime	プリコンディション時間	0 ul
adcConfig.powerupTime	パワーアップ時間	0 ul
adcConfig.enableIdlePowerDown	アイドルパワーダウン有効	false
adcConfig.msbStretchMode	MSB ストレッチモード 0: CY_ADC_MSB_STRETCH_MODE_1CYCLE 1: CY_ADC_MSB_STRETCH_MODE_2CYCLE	1 ul
adcConfig.enableHalfLsbConv	ハーフ LSB 変換有効	0 ul
adcConfig.sarMuxEnable	SAR MUX 有効	true
adcConfig.adcEnable	ADC 有効	true
adcConfig.sarIpEnable	SAR IP 有効	true

表 2 ADC グローバル設定関数一覧

関数	説明	値
Cy_Adc_Init(PASS SAR,ADC Configure)	ADC モジュール初期化	PASS SAR = BB_POTI_ANALOG_MACRO ADC Configure = adcConfig

2.2.3 サンプルコード

ここでは、ADC グローバル設定の初期設定のサンプルコードを示します。次の説明は、SDL のドライバ部のレジスタ表記を理解するのに役立ちます。

- base-> unENABLE.stcField.u1CHAN_EN は、レジスタ TRM に記載される PASS0_SAR0_CH0_ENABLE.CHAN_EN です。他のレジスタも同じように記述されます。
- パフォーマンス改善策: レジスタ設定のパフォーマンスを向上させるために、SDL は完全な 32 ビットデータをレジスタに書き込みます。各ビットフィールドは、ビット書き込み可能なバッファで事前に生成され、最終的な 32 ビットデータとしてレジスタに書き込まれます。

```
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;
```

レジスタの共用体と構造体表現についての詳細は、hdr/rev_x/ip 配下の cyip_pass.h を参照してください。

2 ソフトウェアトリガ処理

Code listing 1 ADC グローバル設定の一般的な設定

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

   /* Initialize ADC */
    {
        cy_stc_adc_config_t adcConfig =  /* ADC Configuration.*/
        {
            .preconditionTime    = 0ul,
            .powerupTime         = 0ul,
            .enableIdlePowerDown = false,
            .msbStretchMode      = CY_ADC_MSB_STRETCH_MODE_2CYCLE,
            .enableHalfLsbConv   = 0ul,
            .sarMuxEnable        = true,
            .adcEnable           = true,
            .sarIpEnable         = true,
        };
        Cy_Adc_Init(BB_POTI_ANALOG_MACRO, &adcConfig);  /* ADC Initialize. See Code Listing 2.
*/
        Cy_Adc_SetDebugFreezeMode(PASS0_EPASS_MMIO,&FreezeConfig);  /* Set Debug freeze mode.
See Code Listing 3. */
:
   }
:
    for(;;)
    {
    }
}
```

2 ソフトウェアトリガ処理

Code listing 2 Cy_Adc_Init() 機能

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

        /* CTL register setting */  /* (1) Enable DC Setting. Auto Idle Power Down 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;
    }
    else
    {
        ret = CY_ADC_BAD_PARAM;
    }

    return ret;
}
```

Code listing 3 Cy_Adc_SetDebugFreezeMode () 機能

```
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;  /* (2) Debug Freeze disabled by
default. */
    }
    else
    {
        ret = CY_ADC_BAD_PARAM;
    }
    return ret;
}
```

2.3 ソフトウェアトリガを使用する場合のロジカルチャネルの設定

各ロジカルチャネルは、1 つの A/D 変換結果レジスタがあります。ロジカルチャネルは SARn_CHx_SAMPLE_CTL レジスタにより、任意のアナログ入力 (ADC[n]_i) に割り当てられます。

図 3 はソフトウェアトリガを使用する場合のロジカルチャネルの設定例です。この例では、最小のサンプル時間が設定されます。システムに最適なサンプル時間を検討するためには、関連ドキュメントに記載されているデータシートを参照してください。

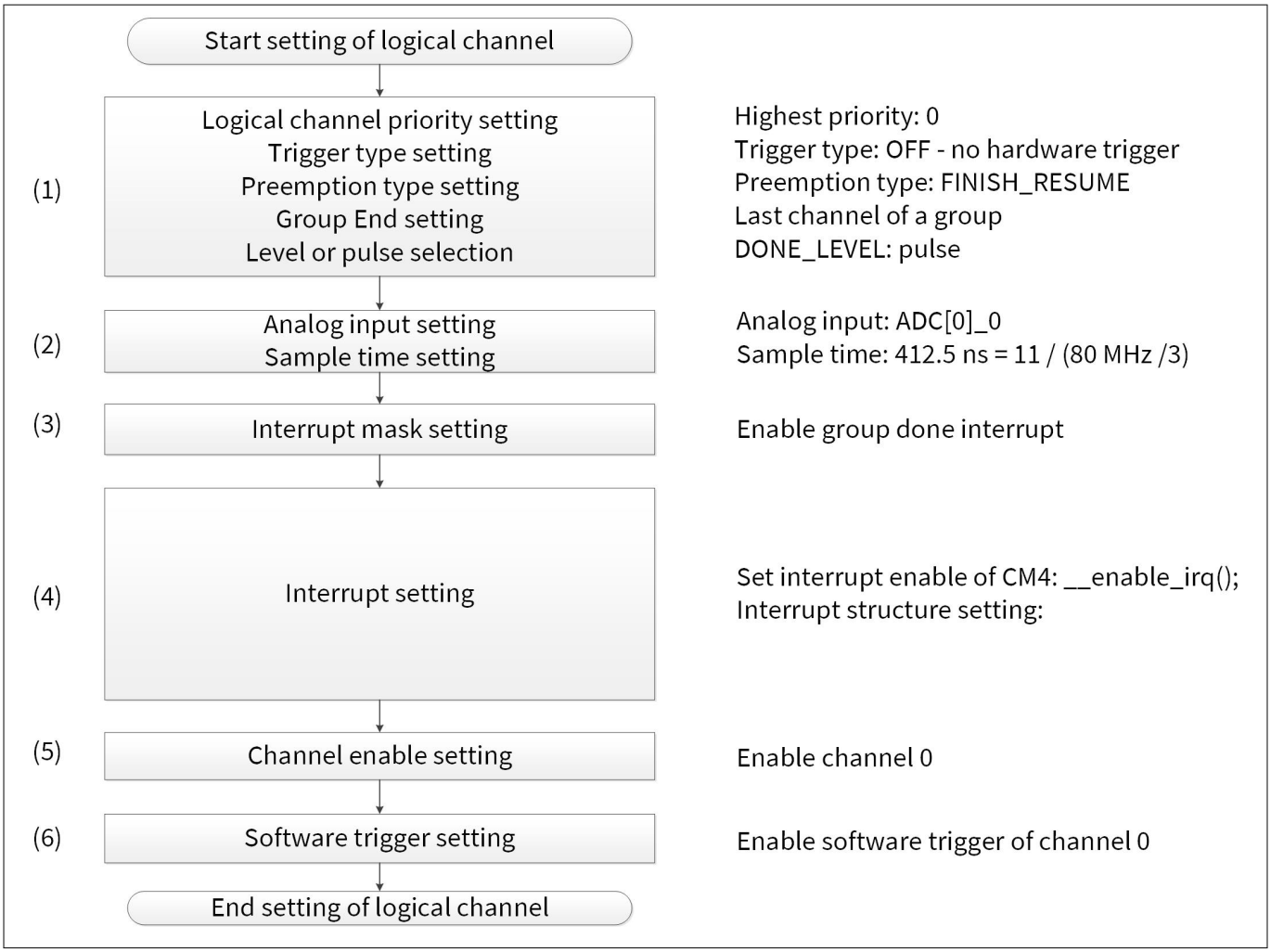


図 3　　ソフトウェアトリガを使用する場合のロジカルチャネルの設定例

この例ではロジカルチャネル 0 が使用されます。所望のアナログ入力の設定をすれば、任意のロジカルチャネルで、このアプリケーションを動作させられます。

2.3.1　　ユースケース

以下のユースケースは、ソフトウェアトリガにおけるロジカルチャネルの設定例です。

- アナログ入力: ADC[0]_0
- サンプル時間: 412.5 ns = 11 / (80 MHz /3)
- ADC トリガ選択: OFF
- チャネル優先度: 0 (最高)
- チャネル Pre-emption タイプ: Finish resume
- チャネルグループ: 最後のチャネル
- ADC Done レベル: パルス
- ADC ピン/ポートアドレス: ADC[0]_0
- 外部アナログ MUX: 0, 有効
- プレコンディショニングモード: OFF
- オーバラップ診断モード: OFF
- キャリブレーション値の選択: 通常

- ポストプロセッシングモード: なし
- 結果の配置: 右
- 符号拡張: 符号なし
- 平均カウント: 0
- 右シフト: 0
- マスクグループ: 完了/キャンセルなし/オーバフローなし
- マスクチャネル: 範囲外/パルスなし/オーバフローなし

2.3.2 コンフィグレーション

ソフトウェアトリガを使用したロジカルチャネル設定の SDL の設定部のパラメータを表 3 に、関数を表 4 に示します。

表 3 ソフトウェアトリガを使用したロジカルチャネル設定パラメータ

パラメータ	説明	値
BB_POTI_ANALOG_MACRO	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログマクロ	CY_ADC_POT_MACRO: PASS0_SAR0
BB_POTI_ANALOG_INPUT_NO	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログ入力数	CH0 (CH[ADC_LOGICAL_CHANNEL])
adcConfig.msbStretchMode	MSB ストレッチモード	1ul (表 1 参照)
adcChannelConfig.triggerSelection	トリガ 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	チャネル優先度	0ul
adcChannelConfig.preenptionType	Pre-emption タイプ 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	グループ終了？	true
adcChannelConfig.doneLevel	Done レベル 0: CY_ADC_DONE_LEVEL_PULSE 1: CY_ADC_DONE_LEVEL_LEVEL	0ul
adcChannelConfig.pinAddress	ピンアドレス	BB_POTI_ANALOG_INPUT_NO

(続く)

表 3 (続き) ソフトウェアトリガを使用したロジカルチャネル設定パラメータ

パラメータ	説明	値
adcChannelConfig.portAddress	ポートアドレス 0: CY_ADC_PORT_ADDRESS_SARMUX0 1: CY_ADC_PORT_ADDRESS_SARMUX1 2: CY_ADC_PORT_ADDRESS_SARMUX2 3: CY_ADC_PORT_ADDRESS_SARMUX3	0ul
adcChannelConfig.extMuxSelect	外部 MUX 選択	0ul
adcChannelConfig.extMuxEnable	外部 MUX 有効	true
adcChannelConfig.preconditionMode	プリコンディションモード 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	オーバラップ診断モード 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	サンプル時間	samplingCycle (計算値)
adcChannelConfig.calibrationValueSelect	キャリブレーション値選択 0: CY_ADC_CALIBRATION_VALUE_REGULAR 1: CY_ADC_CALIBRATION_VALUE_ALTERNATE	0ul
adcChannelConfig.postProcessingMode	ポストプロセッシングモード 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	結果の配置 0: CY_ADC_RESULT_ALIGNMENT_RIGHT 1: CY_ADC_RESULT_ALIGNMENT_LEFT	0ul

(続く)

表 3 (続き) ソフトウェアトリガを使用したロジカルチャネル設定パラメータ

パラメータ	説明	値
adcChannelConfig.signExtention	符号拡張 0: CY_ADC_SIGN_EXTENTION_UNSIGNED 1: CY_ADC_SIGN_EXTENTION_SIGNED	0ul
adcChannelConfig.averageCount	平均カウント	0ul
adcChannelConfig.rightShift	右シフト	0ul
adcChannelConfig.rangeDetectionMode	レンジ検出モード 0: CY_ADC_RANGE_DETECTION_MODE_BELOW_LO 1: CY_ADC_RANGE_DETECTION_MODE_INSIDE_RANGE 2: CY_ADC_RANGE_DETECTION_MODE_ABOVE_HI 3: CY_ADC_RANGE_DETECTION_MODE_OUTSIDE_RANGE	1ul
adcChannelConfig.rangeDetectionLoThreshold	レンジ検出閾値 (下限)	0x0000ul
adcChannelConfig.rangeDetectionHiThreshold	レンジ検出閾値 (上限)	0x0FFFul
adcChannelConfig.mask.grpDone	マスクグループ Done	true
adcChannelConfig.mask.grpCancelled	マスクグループキャンセル	false
adcChannelConfig.mask.grpOverflow	マスクグループオーバフロー	false
adcChannelConfig.mask.chRange	マスクチャネルレンジ	false
adcChannelConfig.mask.chPulse	マスクチャネルパルス	false
adcChannelConfig.mask.chOverflow	マスクチャネルオーバフロー	false

表 4 ソフトウェアトリガを使用したロジカルチャネル設定関数

関数	説明	値
Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)	ADC チャネルの初期化	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig
Cy_Adc_Channel_SetInterruptMask(PASS SARchannel, INTR Source)	ADC チャネル割込みマスク設定	PASS SARchannel = base INTR Source = mask

(続く)

表 4 (続き) ソフトウェアトリガを使用したロジカルチャネル設定関数

関数	説明	値
`Cy_SysInt_InitIRQ(Config)`	割込みベクタを設定し参照されるシステム割込みを初期化	Config = irq_cfg
`Cy_Adc_Channel_Enable(SARchannel)`	対応チャネル有効	SARchannel = PASS_SAR_CH
`Cy_Adc_Channel_SoftwareTrigger(PASS SARchannel)`	ソフトウェア開始トリガ発行	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

2.3.3 サンプルコード

ソフトウェアトリガー設定による論理チャネル設定の初期設定のサンプルコードについては、Code listing 4～Code listing 9 を参照してください。

Code listing 4 ADC グローバル設定の一般的な設定

```
:
#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 =  /* ADC Configuration.*/
        {
            .preconditionTime    = 0ul,
            .powerupTime         = 0ul,
            .enableIdlePowerDown = false,
            .msbStretchMode      = CY_ADC_MSB_STRETCH_MODE_2CYCLE,
            .enableHalfLsbConv   = 0ul,
            .sarMuxEnable        = true,
            .adcEnable           = true,
            .sarIpEnable         = true,
        };
        cy_stc_adc_channel_config_t adcChannelConfig =
        {
            .triggerSelection          = CY_ADC_TRIGGER_OFF,  /* Trigger type setting */
            .channelPriority           = 0ul,  /* (1) Logical channel priority setting */
            .preenptionType            = CY_ADC_PREEMPTION_FINISH_RESUME,  /* Preemption type
setting */
            .isGroupEnd                = true,  /* Group End setting */
            .doneLevel                 = CY_ADC_DONE_LEVEL_PULSE,  /* Level or pulse selection
*/
            .pinAddress                = BB_POTI_ANALOG_INPUT_NO,  /* (2) Analog input setting,
Sample time setting */
            .portAddress               = CY_ADC_PORT_ADDRESS_SARMUX0,  /* (2) Analog input
setting, Sample time setting */
            .extMuxSelect              = 0ul,  /* (2) Analog input setting, Sample time setting
*/
            .extMuxEnable              = true,  /* (2) Analog input setting, Sample time
setting */
            .preconditionMode          = CY_ADC_PRECONDITION_MODE_OFF,  /* (2) Analog input
setting, Sample time setting */
            .overlapDiagMode           = CY_ADC_OVERLAP_DIAG_MODE_OFF,  /* (2) Analog input
setting, Sample time setting */
            .sampleTime                = 0ul,   /* (2) Analog input setting, Sample time
setting */
            .calibrationValueSelect    = CY_ADC_CALIBRATION_VALUE_REGULAR,  /* (2) Analog input
setting, Sample time setting */
            .postProcessingMode        = CY_ADC_POST_PROCESSING_MODE_NONE,  /* (2) Analog input
setting, Sample time setting */
```

2 ソフトウェアトリガ処理

```
            .resultAlignment           = CY_ADC_RESULT_ALIGNMENT_RIGHT,  /* (2) Analog input
setting, Sample time setting */
            .signExtention             = CY_ADC_SIGN_EXTENTION_UNSIGNED,  /* (2) Analog input
setting, Sample time setting */
            .averageCount              = 0ul,  /* (2) Analog input setting, Sample time setting
*/
            .rightShift                = 0ul,  /* (2) Analog input setting, Sample time setting
*/
            .rangeDetectionMode        = CY_ADC_RANGE_DETECTION_MODE_INSIDE_RANGE,
            .rangeDetectionLoThreshold = 0x0000ul,  /* Continuation of (2) */
            .rangeDetectionHiThreshold = 0x0FFFul,  /* Continuation of (2) */
            .mask.grpDone              = true,  /* (3) Interrupt mask setting */
            .mask.grpCancelled         = false,  /* (3) Interrupt mask setting */
            .mask.grpOverflow          = false,  /* (3) Interrupt mask setting */
            .mask.chRange              = false,  /* (3) Interrupt mask setting */
            .mask.chPulse              = false,  /* (3) Interrupt mask setting */
            .mask.chOverflow           = false,  /* (3) Interrupt mask setting */
        };



        Cy_Adc_Init(BB_POTI_ANALOG_MACRO, &adcConfig);  /* ADC Initialize. See Code Listing 2 */
        Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&adcChannelConfig);  /* Initialize ADC Channel. See Code Listing 5. */
    }

:
        Cy_SysInt_InitIRQ(&irq_cfg);  /* Initialize Interrupt Request. See Code Listing 7 */

:
    /* Enable ADC ch. */  /* Enable ADC Channel. See Code Listing 8 */
    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]);  /*
Software Trigger Setting. See Code Listing 9 */
    for(;;)
    {
    }
}
```

2 ソフトウェアトリガ処理

Code listing 5 Cy_Adc_Channel_Init() 機能

```
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;
        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);  /* ADC Channel Set Interrupt
Mask. See Code Listing 6 */
    }
    else
    {
        ret = CY_ADC_BAD_PARAM;
    }
    return ret;
}
```

Code listing 6 Cy_Adc_Channel_SetInterruptMask() 機能

```
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;
    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;
}
```

2 ソフトウェアトリガ処理

Code listing 7 Cy_SysInt_InitIRQ() 機能

```
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
            #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;  /* (4) Set Interrupt Enable of CM4, Interrupt structure setting,
Priority setting, Clear pending status and Enable IRQ. */
            #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->intIdx;
                    unIntCtl0.stcField.u1CPU_INT_VALID = config->isEnabled ? 1ul : 0ul;
                    CPUSS->unCM7_0_SYSTEM_INT_CTL[config->sysIntSrc].u32Register =
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);
    }
```

Code listing 8 Cy_Adc_Channel_Enable() 機能

```
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() 機能

```
void Cy_Adc_Channel_SoftwareTrigger(volatile stc_PASS_SAR_CH_t * base)
{
    base->unTR_CMD.stcField.u1START = 1ul;  /* (6) Enable Software Trigger. */
}
```

2.4 ソフトウェアトリガを使用する場合の A/D 変換割込み処理

図 4 に、ソフトウェアトリガを使用する場合の A/D 変換割込み処理例を示します。CPU 割込み処理の詳細は、関連ドキュメントのアーキテクチャ TRM を参照してください。

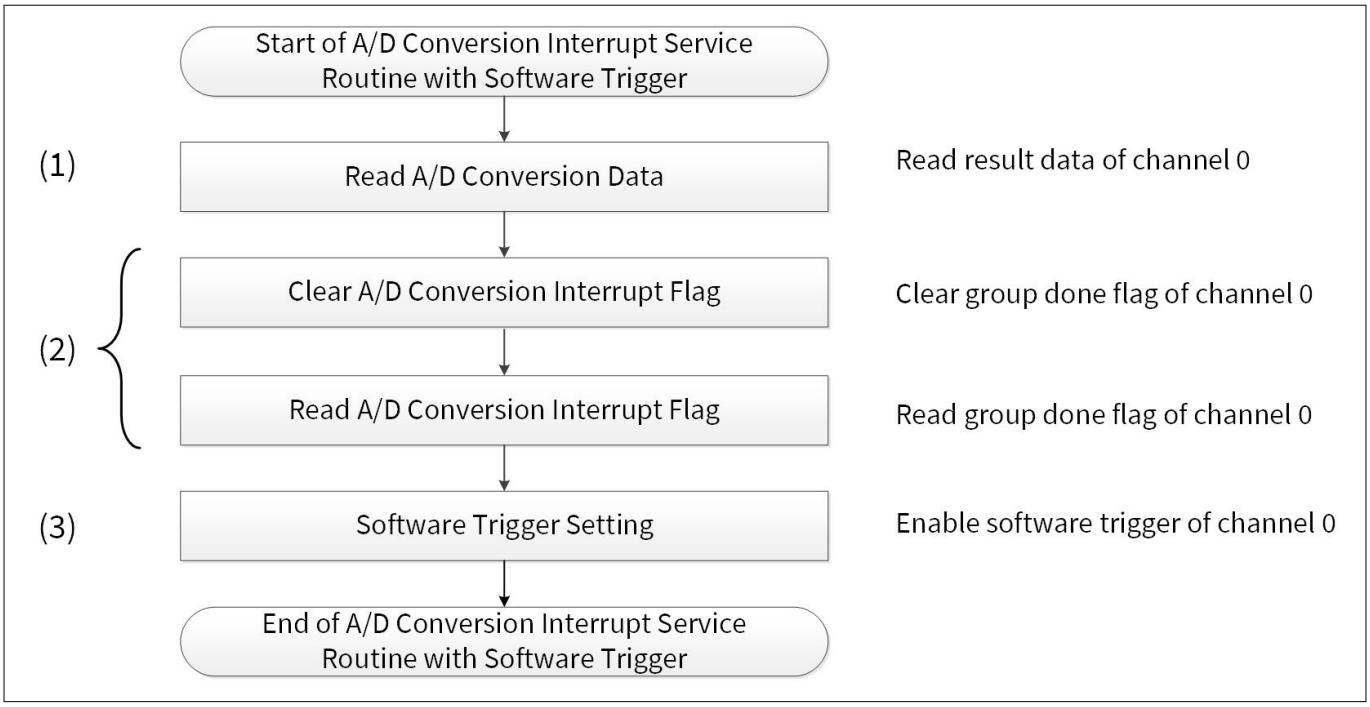


図 4　　ソフトウェアトリガを使用する場合の A/D 変換割込み処理例

2.4.1　　ユースケース

セクション 2.3.1 を参照してください。

2.4.2　　コンフィグレーション

ADC グローバル設定における SDL の設定部のパラメータを表 5 に、関数を表 6 に示します。

表 5　　ソフトウェアトリガを使用した ADC 割込み処理パラメータ

パラメータ	説明	値
BB_POTI_ANALOG_MACRO	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログマクロ	CY_ADC_POT_MACRO: PASS0_SAR0
resultBuff	変換結果バッファ	- (計算値)
statusBuff	変換結果ステータス	- (計算値)
resultIdx	インデックス結果	- (計算値)
intrSource	割込みソース	- (計算値)

表 6　　ソフトウェアトリガを使用した ADC 割込み処理関数

関数	説明	値
Cy_Adc_Channel_GetResult(SAR Channel,Result,Status)	変換結果とステータスの取得	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Result = resultBuff[resultIdx], Status = statusBuff[resultIdx]

(続く)

表 6 (続き) ソフトウェアトリガを使用した ADC 割込み処理関数

関数	説明	値
`Cy_Adc_Channel_ClearInterruptStatus(SAR Channel,Source)`	対応するチャネル割込みステータスのクリア	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Source = intrSource
`Cy_Adc_Channel_SoftwareTrigger(SAR Channel)`	ソフトウェア開始トリガ発行	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

2.4.3 サンプルコード

ソフトウェアトリガを使用する場合の ADC ISR の初期設定のサンプルコードについては、Code listing 10〜 Code listing 12 を参照してください。

2 ソフトウェアトリガ処理

Code listing 10 ソフトウェアトリガを使用する場合の ADC ISR

```
#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) */  /* (1) Read A/D conversion data. See Code Listing 11. */
        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 */  /* (2) Clear and read A/D conversion flag. See Code
Listing 12. */
        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]);  /* (3)
Software Trigger Setting. See Code Listing 9. */
    }
    else
    {
        // Unexpected interrupt
        CY_ASSERT(false);
    }
}
```

Code listing 11 Cy_Adc_Channel_GetResult() 機能

```
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() 機能

```
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 ハードウェアトリガ処理

TCPWM, GPIO, およびイベントジェネレータのような他のハードウェア関連機能のトリガによって SAR A/D 変換が開始できます。

ここでは、ADC[0]_0 端子に入力された電圧値をデジタル値に変換するアプリケーション例を紹介します。A/D 変換は、対応する TCPWM のハードウェアトリガにより一定の時間間隔で繰り返されます。

このアプリケーションの物理設定は図 1 に示されるものと同じです。

基本的な ADC グローバル設定に記載されるようにパラメータを設定していることを確認してください。

このアプリケーションを実装するためには、ADC チャネル設定手順とハードウェアトリガの使用例に従ってください。このセクションの例では、対応する TCPWM をトリガに使用します。

3.1 ハードウェアトリガを使用する場合のロジカルチャネル設定

図 5 に、CYT2 シリーズのハードウェアトリガを使用する場合のロジカルチャネルの設定例を示します。この例では、最小のサンプル時間が設定されます。システムに最適なサンプル時間を検討するためには、関連資料のデータシートを参照してください。

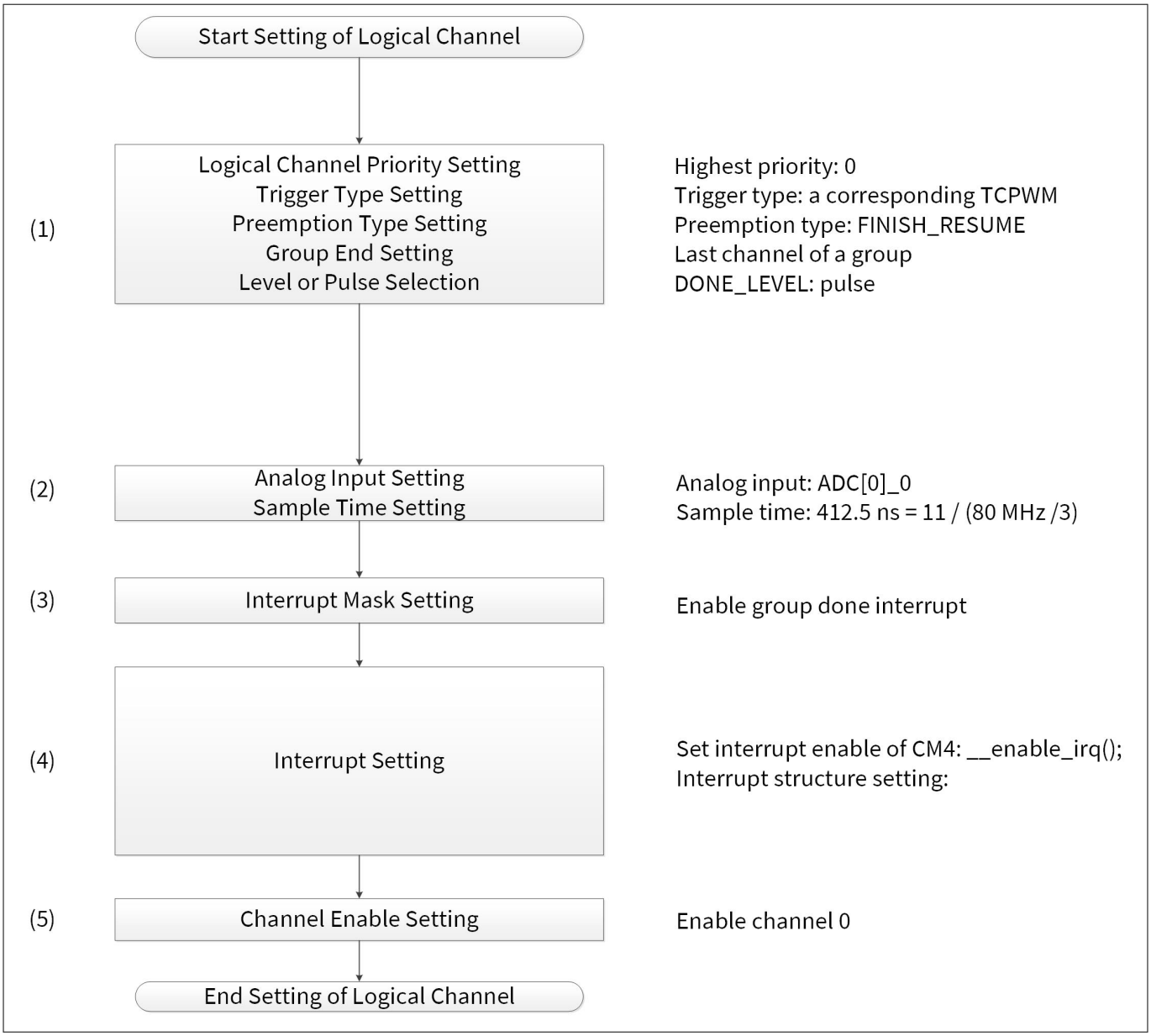


図 5　　CYT2 シリーズのハードウェアトリガを使用する場合のロジカルチャネル設定例

この例ではロジカルチャネル 0 が使用されます。所望のアナログ入力の設定をすれば、任意のロジカルチャネルで、このアプリケーションを動作させられます。このほかに、TCPWM とトリガマルチプレクサについても設定が必要です。

3.1.1　　ユースケース

以下のユースケースは、ハードウェアトリガにおけるロジカルチャネルの設定例です。

- アナログ入力: ADC[0]_0
- サンプル時間: 412.5 ns = 11 / (80 MHz /3)
- トリガ選択: TCPWM
- Pre-emption タイプ: FINISH_RESUME
- GROUP_END: グループの最後のチャネル
- 外部アナログ MUX: 0, 有効
- プレコンディショニングモード: OFF

- オーバラップ診断モード: OFF
- キャリブレーション値の選択: 通常
- ポストプロセッシングモード: なし
- 結果の配置: 右
- 符号拡張: 符号なし
- 平均カウント: 0
- 右シフト: 0
- マスクグループ: 完了/キャンセルなし/オーバフローなし
- マスクチャネル: 範囲外/パルスなし/オーバフローなし

3.1.2 コンフィグレーション

CYT2 シリーズのハードウェアトリガにおけるロジカルチャネル設定の SDL の設定部のパラメータを表 7 に、関数を表 8 に示します。

表 7 CYT2 シリーズのハードウェアトリガにおけるロジカルチャネル設定パラメータ

パラメータ	説明	値
BB_POTI_ANALOG_MACRO	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログマクロ	CY_ADC_POT_MACRO: PASS0_SAR0
CY_GPIO_DM_STRONG_IN_OFF	ストロングドライブ，入力バッファオフ	0x06ul
pin_cfg1.outVal	ピン出力状態	0ul
pin_cfg1.driveMode	ドライブモード	CY_GPIO_DM_STRONG_IN_OFF
pin_cfg1.hsiom	高速 I/O マトリクス選択	TCPWMx_LINEx_MUX
pin_cfg1.intEdge	割込みエッジタイプ	0ul
pin_cfg1.intMask	割込み有効マスク	0ul
pin_cfg1.vtrip	入力バッファ電圧トリップタイプ	0ul
pin_cfg1.slewRate	出力バッファスルーレート	0ul
pin_cfg1.driveSel	ドライブ強度	0ul
CY_GPIO_DM_ANALOG	アナログ High-Z，入力バッファオフ	0x00ul
adcPinConfig.outVal	ピン出力状態	0ul
adcPinConfig.driveMode	ドライブモード	CY_GPIO_DM_ANALOG
adcPinConfig.hsiom	高速 I/O マトリクス選択	P6_0_GPIO
adcPinConfig.intEdge	割込みエッジタイプ	0ul
adcPinConfig.intMask	割込み有効マスク	0ul
adcPinConfig.vtrip	入力バッファ電圧トリップタイプ	0ul
adcPinConfig.slewRate	出力バッファスルーレート	0ul

(続く)

表 7 (続き) CYT2 シリーズのハードウェアトリガにおけるロジカルチャネル設定パラメータ

パラメータ	説明	値
adcPinConfig.driveSel	ドライブ強度	0ul
adcConfig.preconditionTime	プリコンディション時間	0ul
adcConfig.powerupTime	パワーアップ時間	0ul
adcConfig.enableIdlePowerDown	アイドルパワーダウン有効	false
adcConfig.msbStretchMode	MSB ストレッチモード	1ul (表 1 参照)
adcConfig.enableHalfLsbConv	ハーフ LSB 変換有効	0ul
adcConfig.sarMuxEnable	SAR MUX 有効	true
adcConfig.adcEnable	ADC 有効	true
adcConfig.sarIpEnable	SAR IP 有効	true
adcChannelConfig.triggerSelection	トリガ選択	1ul (表 3 参照)
adcChannelConfig.channelPriority	チャネル優先度	0ul
adcChannelConfig.preenptionType	Pre-emption タイプ	3ul (表 3 参照)
adcChannelConfig.isGroupEnd	グループ終了？	true
adcChannelConfig.doneLevel	Done レベル	0ul (表 3 参照)
adcChannelConfig.pinAddress	ピンアドレス	BB_POTI_ANALOG_INPUT_NO
adcChannelConfig.portAddress	ポートアドレス	0ul (表 3 参照)
adcChannelConfig.extMuxSelect	外部 MUX 選択	0ul
adcChannelConfig.extMuxEnable	外部 MUX 有効	true
adcChannelConfig.preconditionMode	プリコンディションモード	0ul (表 3 参照)
adcChannelConfig.overlapDiagMode	オーバラップ診断モード	0ul (表 3 参照)
adcChannelConfig.sampleTime	サンプル時間	0ul
adcChannelConfig.calibrationValueSelect	キャリブレーション値選択	0ul (表 3 参照)
adcChannelConfig.postProcessingMode	ポストプロセッシングモード	0ul (表 3 参照)
adcChannelConfig.resultAlignment	結果の配置	0ul (表 3 参照)
adcChannelConfig.signExtention	符号拡張	0ul (表 3 参照)
adcChannelConfig.averageCount	平均カウント	0ul
adcChannelConfig.rightShift	右シフト	0ul
adcChannelConfig.rangeDetectionMode	レンジ検出モード	1ul (表 3 参照)
adcChannelConfig.rangeDetectionLoThreshold	レンジ検出閾値 (下限)	0x0000ul
adcChannelConfig.rangeDetectionHiThreshold	レンジ検出閾値 (上限)	0x0FFFul
adcChannelConfig.mask.grpDone	マスクグループ Done	true
adcChannelConfig.mask.grpCancelled	マスクグループキャンセル	false

(続く)

表 7 (続き) CYT2 シリーズのハードウェアトリガにおけるロジカルチャネル設定パラメータ

パラメータ	説明	値
adcChannelConfig.mask.grpOverflow	マスクグループオーバフロー	false
adcChannelConfig.mask.chRange	マスクチャネルレンジ	false
adcChannelConfig.mask.chPulse	マスクチャネルパルス	false
adcChannelConfig.mask.chOverflow	マスクチャネルオーバフロー	false

表 8 CYT2 シリーズのハードウェアトリガにおけるロジカルチャネル設定関数

関数	説明	値
Cy_Adc_Channel_GetInterruptMaskedStatus(PASS SARchannel,INTR Source)	割込みステータス取得	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] INTR Source = &intrSource
Cy_SysInt_SetSystemIrqVector(Source,INTR handler)	システム割込みのユーザー割込みサービスルーチンベクタの変更	Source = sysIntSrc INTR handler = AdcIntHandler
Cy_GPIO_Pin_Init(Port Number,Pin Number,gpio_pin_config)	すべてのピン構成設定の初期化	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)	PWM 動作用に TCPWM を初期化	Group Counter = TCPWMx_GRPx_CNTx_PWM TCPWM PWM config = &MyPWM_config
Cy_Tcpwm_Pwm_Enable(Group Counter)	TCPWM ブロックを初期化解除	Group Counter = TCPWMx_GRPx_CNTx_PWM
Cy_Tcpwm_TriggerStart(Group Counter)	選択した TCPWM でソフトウェアの起動トリガをかける	Group Counter = TCPWMx_GRPx_CNTx_PWM
Cy_TrigMux_Connect1To1(Trigger MUX,Invert,Trigger Type,Debug Freeze Enable)	入力トリガソースと出力トリガを接続	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 サンプルコード

CYT2 シリーズのハードウェアトリガを使用したロジカルチャネル設定の初期設定のサンプルコードについては、Code listing 13 ～ Code listing 20 を参照してください。

3 ハードウェアトリガ処理

Code listing 13 CYT2 シリーズのハードウェアトリガを使用したロジカルチャネル設定

```
#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 =  /* GPIO Pin Configuration */
{
      .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 =  /* TCPWM Configuration */
{
      .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,
      .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 =  /* ADC Pin Configuration. */
{
      .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 =  /* ADC Configuration. See Code Listing 2. */
{
     .preconditionTime    = 0ul,
      .powerupTime         = 0ul,
      .enableIdlePowerDown = false,
      .msbStretchMode      = CY_ADC_MSB_STRETCH_MODE_2CYCLE,
      .enableHalfLsbConv   = 0ul,
     .sarMuxEnable        = true,
      .adcEnable           = true,
      .sarIpEnable         = true,
};
```

```
cy_stc_adc_channel_config_t adcChannelConfig =  /* ADC Channel Configuration. See Code Listing
5. */
{
     /* Trigger type: Trigger from corresponding TCPWM channel */
.triggerSelection          = CY_ADC_TRIGGER_TCPWM,  /* Trigger Type Setting */
     .channelPriority           = 0ul,  /* (1) Logical Channel Priority Setting */
     .preenptionType            = CY_ADC_PREEMPTION_FINISH_RESUME,  /* Preemption Type Setting
*/
.isGroupEnd                = true,  /* Group End Setting */
     /* CY_ADC_DONE_LEVEL_PULSE = 0 */
.doneLevel                 = CY_ADC_DONE_LEVEL_PULSE,   /* Level or Pulse Selection  */
     .pinAddress                = BB_POTI_ANALOG_INPUT_NO,  /* (2) Analog Input Setting, Sample
Time Setting */
     .portAddress               = CY_ADC_PORT_ADDRESS_SARMUX0,  /* (2) Analog Input Setting,
Sample Time Setting */
     .extMuxSelect              = 0ul,  /* (2) Analog Input Setting, Sample Time Setting */
     .extMuxEnable              = true,  /* (2) Analog Input Setting, Sample Time Setting */
     .preconditionMode          = CY_ADC_PRECONDITION_MODE_OFF,  /* (2) Analog Input Setting,
Sample Time Setting */
     .overlapDiagMode           = CY_ADC_OVERLAP_DIAG_MODE_OFF,  /* (2) Analog Input Setting,
Sample Time Setting */
     .sampleTime                = 0ul,  /* (2) Analog Input Setting, Sample Time Setting */
    .calibrationValueSelect    = CY_ADC_CALIBRATION_VALUE_REGULAR,  /* (2) Analog Input
Setting, Sample Time Setting */
     .postProcessingMode        = CY_ADC_POST_PROCESSING_MODE_NONE,  /* (2) Analog Input
Setting, Sample Time Setting */
     .resultAlignment           = CY_ADC_RESULT_ALIGNMENT_RIGHT,  /* (2) Analog Input Setting,
Sample Time Setting */
    .signExtention             = CY_ADC_SIGN_EXTENTION_UNSIGNED,  /* (2) Analog Input Setting,
Sample Time Setting */
     .averageCount              = 0ul,  /* (2) Analog Input Setting, Sample Time Setting */
     .rightShift                = 0ul,  /* (2) Analog Input Setting, Sample Time Setting */
     .rangeDetectionMode        = CY_ADC_RANGE_DETECTION_MODE_INSIDE_RANGE,  /* (2) Analog
Input Setting, Sample Time Setting */
     .rangeDetectionLoThreshold = 0x0000ul,  /* (2) Analog Input Setting, Sample Time Setting */
     .rangeDetectionHiThreshold = 0x0FFFul,  /* (2) Analog Input Setting, Sample Time Setting */
     .mask.grpDone              = true,  /* (3) Interrupt Mask Setting */
     .mask.grpCancelled         = false,  /* (3) Interrupt Mask Setting */
     .mask.grpOverflow          = false,  /* (3) Interrupt Mask Setting */
     .mask.chRange              = false,  /* (3) Interrupt Mask Setting */
     .mask.chPulse              = false,  /* (3) Interrupt Mask Setting */
     .mask.chOverflow           = false,  /* (3) Interrupt Mask Setting */
};
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);  /* Get Interrupt Masked Status. See Code Listing 14. */
```

```
    if(intrSource.grpDone)
    {
        /* Get the result(s) */
      Cy_Adc_Channel_GetResult(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&resultBuff[resultIdx], &statusBuff[resultIdx]);  /* Read A/D conversion data. See Code Listing
11. */


        /* 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);  /* Clear and read A/D conversion flag. See Code Listing 12. */
    }
    else
    {
        /* Unexpected interrupt */
        CY_ASSERT(false);
    }
}

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

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

    SystemInit();

:
    /* 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);  /* ADC Initialize. See Code Listing 2 */
    Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&adcChannelConfig);  /* ADC Channel Initialize. See Code Listing 5. */

    /* 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);  /* (4) Initialize Interrupt Request. See Code Listing 7. */
    Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, AdcIntHandler);  /* Initialize Interrupt
Request. See Code Listing 15. */
    NVIC_SetPriority(irq_cfg.intIdx, 0ul);
    NVIC_EnableIRQ(irq_cfg.intIdx);
```

```
  :


    /* Port Configuration for TCPWM */  /* Initialize GPIO Pin. See Code Listing 16. */
   Cy_GPIO_Pin_Init(TCPWMx_LINEx_PORT, TCPWMx_LINEx_PIN, &pin_cfg1);

    /* Initialize TCPWM0_GRPx_CNTx_PWM_PR as PWM Mode & Enable */
    Cy_Tcpwm_Pwm_Init(TCPWMx_GRPx_CNTx_PWM, &MyPWM_config);  /* Initialize TCPWM PWM. See Code
Listing 17. */




    /* Enable ADC ch. and PWM */
    Cy_Adc_Channel_Enable(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);  /* (5) Enable ADC
channel. See Code Listing 8. */

    Cy_Tcpwm_Pwm_Enable(TCPWMx_GRPx_CNTx_PWM);  /* TCPWM PWM Enable. See Code Listing 18. */

   Cy_Tcpwm_TriggerStart(TCPWMx_GRPx_CNTx_PWM);  / * TCPWM Trigger Start Select. See Code
Listing 19 */


    /* Trigger MUX */
    Cy_TrigMux_Connect1To1(TRIG_IN_1TO1_1_TCPWM_TO_PASS_CH_TR0,
                           0ul,
                           TRIGGER_TYPE_PASS_TR_SAR_CH_IN__EDGE,
                           0ul);  /* Connects an input trigger source and output trigger. See
Code Listing 20. */

    for(;;);
}
```

Code listing 14 Cy_Adc_Channel_GetInterruptMaskedStatus() 機能

```
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;
}
```

Code listing 15 Cy_SysInt_SetSystemIrqVector() 機能

```
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() 機能

```
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() 機能

```
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))
    {

        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;
        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() 機能

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

Code listing 19 Cy_Tcpwm_TriggerStart() 機能

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

Code listing 20 Cy_TrigMux_Connect1To1() 機能

```
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 ハードウェアトリガを使用する場合の A/D 変換割込み処理

図 6 に、ハードウェアトリガを使用する場合の A/D 変換割込み処理例を示します。CPU 割込み処理の詳細は、関連資料のアーキテクチャ TRM を参照してください。

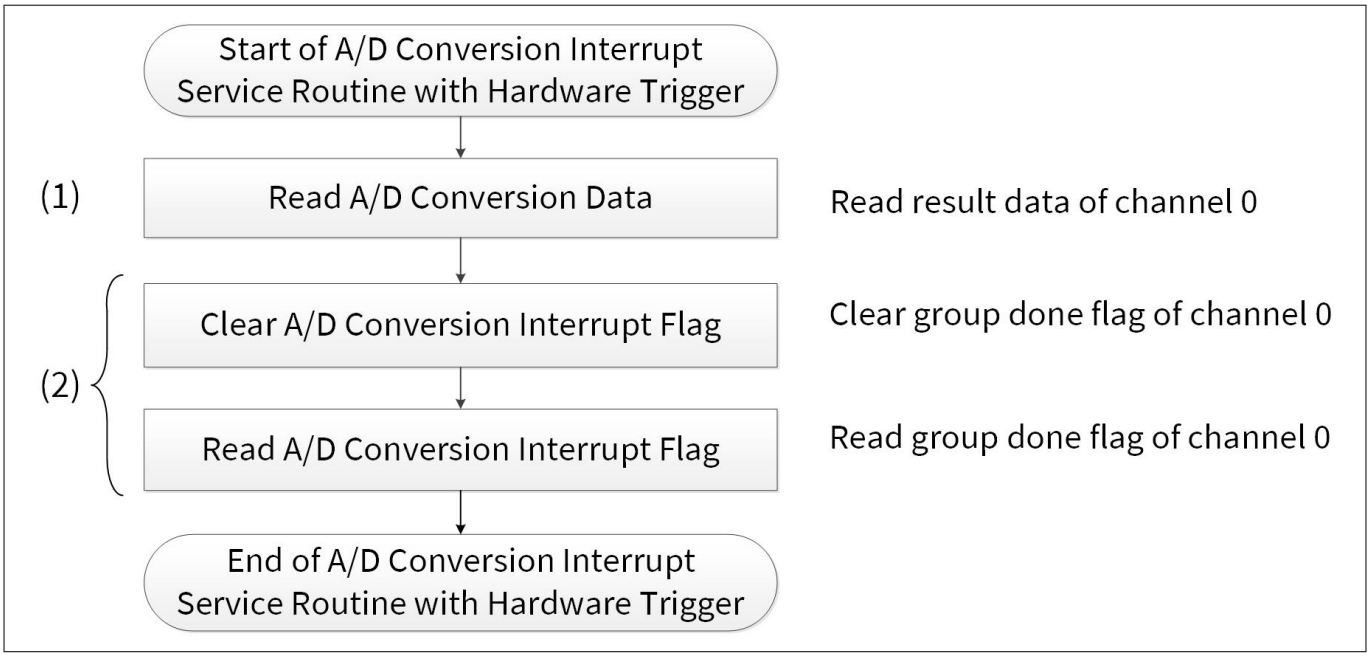


図 6　　ハードウェアトリガを使用する場合の A/D 変換割込み処理例

3.2.1　　ユースケース

セクション 3.1.1 を参照してください。

3.2.2　　コンフィグレーション

ハードウェアトリガを使用する場合の A/D 変換割込み処理の SDL の設定部のパラメータを表 9 に、関数を表 10 に示します。

表 9　　ハードウェアトリガを使用する場合の A/D 変換割込み処理のパラメータ

パラメータ	説明	値
BB_POTI_ANALOG_MACRO	TRAVEO™ T2G MCU ベースボード上のポテンショメータ用のアナログマクロ	CY_ADC_POT_MACRO: PASS0_SAR0
resultBuff	変換結果バッファ	- (計算値)
statusBuff	変換結果ステータス	- (計算値)
resultIdx	インデックス結果	- (計算値)
intrSource	割込みソース	- (計算値)

表 10　　ハードウェアトリガを使用する場合の A/D 変換割込み処理の関数

関数	説明	値
Cy_Adc_Channel_GetResult(SAR Channel,Result,Status)	変換結果とステータスの取得	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Result = resultBuff[resultIdx], Status = statusBuff[resultIdx]

(続く)

表 10 (続き) ハードウェアトリガを使用する場合の A/D 変換割込み処理の関数

関数	説明	値
`Cy_Adc_Channel_ClearInterruptStatus(SAR Channel,Source)`	対応するチャネル割込みステータスのクリア	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Source = intrSource

3.2.3 サンプルコード

ソフトウェアトリガを使用する場合の ADC ISR の初期設定のサンプルコードについては、Code listing 21 を参照してください。

Code listing 21 ソフトウェアトリガを使用する場合の ADC ISR

```
#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) */  /* (1) Read A/D conversion data. See Code Listing 11. */
        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);  */ (2) Clear and read A/D conversion flag. See Code Listing 21. */
 :
    }
    else
    {
        // Unexpected interrupt
        CY_ASSERT(false);
    }
}
```

4 グループ処理

SAR ADC には、1 度のトリガにより、複数端子を使用して、順番に変換を行う機能があります。使用する端子と変換する順番は、アナログ入力端子として設定可能なポートから任意に選択できます。

1 度のトリガで順番に変換される ADC のロジカルチャネルの集合を「グループ」と呼びます。

グループトリガは、グループの最初のチャネルによって定義され、"OFF" (ハードウェアトリガなし), "TCPWM", "Generic 0～4", または"連続"のトリガタイプを持ちます。

SARn_CHxTR_CTL レジスタの GROUP_END ビットで「次チャネルでグループを続ける」オプションが設定されていない場合、そのグループは、1 チャネルだけで構成されます。そうでない場合は、グループは、そのビットが「グループの最後のチャネル」に設定された行の最後のチャネルまで続きます。

グループの最初のチャネルにトリガがかかり、かつ変換が終了した後、次のチャネルの変換が自動的に開始されます。これは、そのグループ内の全チャネルの変換が行われるまで繰り返されます。

図 7 に、1 度のソフトウェアトリガで ADC[0]_10, ADC[0]_12, ADC[0]_15 の順番で電圧値を変換するアプリケーション例を示します。

基本的な ADC グローバル設定に記載されるようにパラメータを設定していることを確認してください。

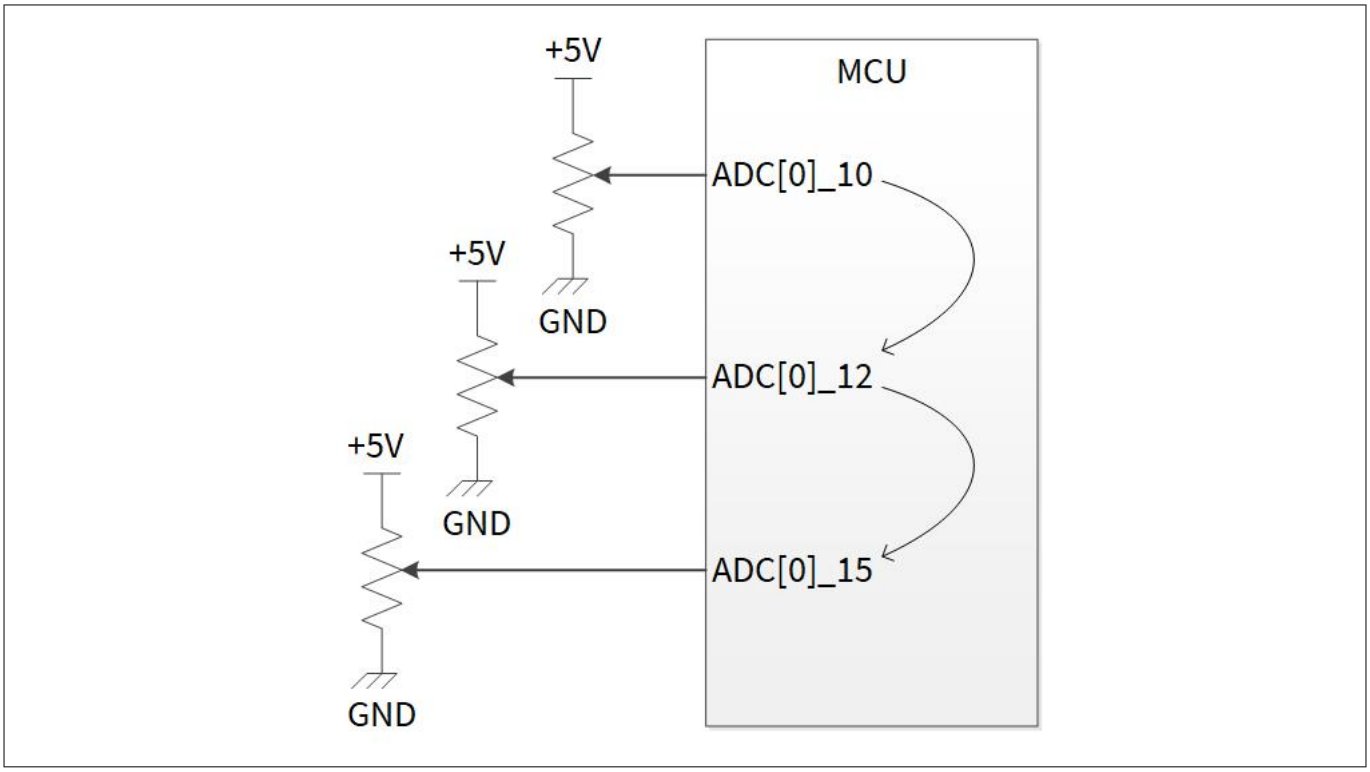


図 7 グループ変換接続例

このアプリケーションを実装するためには、以下のセクションの情報と例を使用してください。

4.1 グループを使用する場合のロジカルチャネルの設定

図 8 に、CYT2 シリーズのグループを使用する場合のロジカルチャネル設定例を示します。この例では、最小のサンプル時間が設定されます。システムに最適なサンプル時間を検討するためには、関連資料のデータシートを参照してください。

4 グループ処理

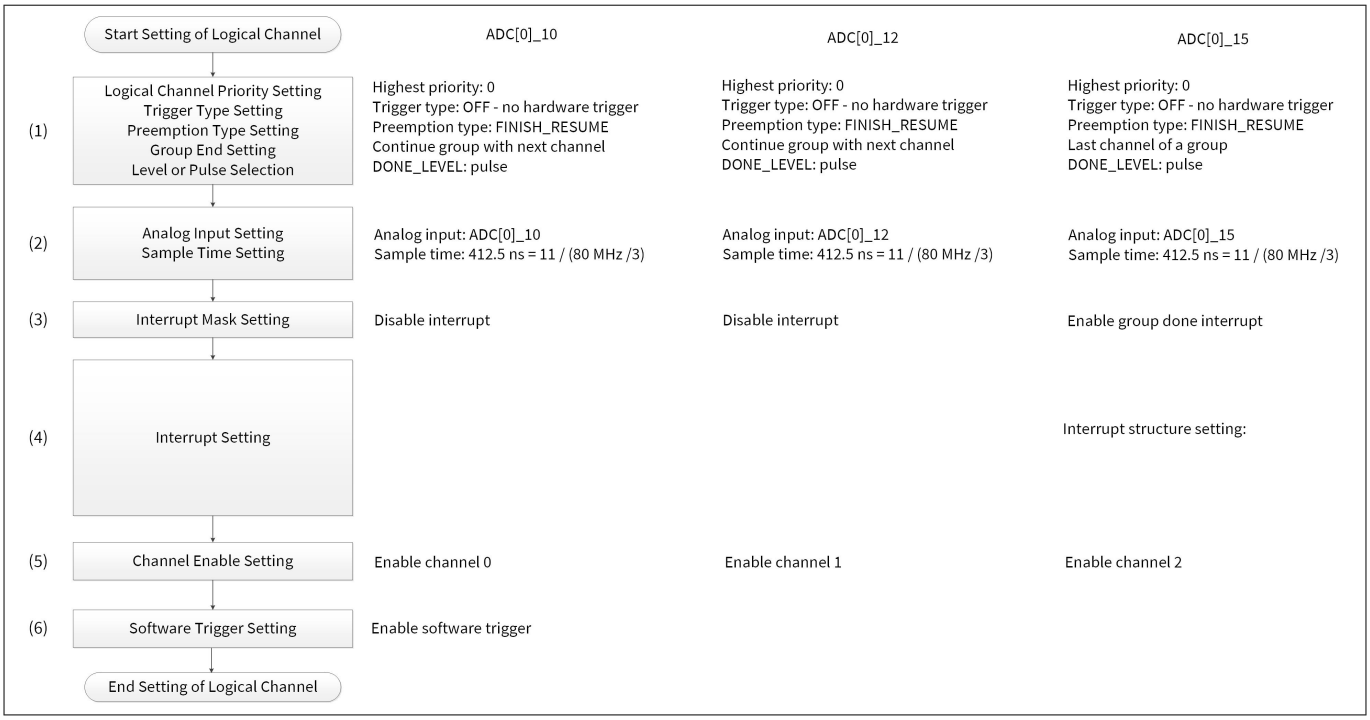


図 8　　CYT2 シリーズのグループを使用する場合のロジカルチャネル設定例

この例ではロジカルチャネル 0, 1, 2 を使用していますが、チャネル番号が連続していれば任意のチャネルを使用できます。

4.1.1　　ユースケース

以下のユースケースに、グループを使用する場合のロジカルチャネルの設定例を示します。

- ADC トリガ選択: OFF
- チャネル優先度: 0 (最高)
- チャネル Pre-emption タイプ: Finish resume
- アナログ入力, チャネルグループおよび割込み

チャネル	アナログ入力	チャネルグループ	割込み
CH0	ADC[0]_10	次のチャネルでグループ続行	無効
CH1	ADC[0]_12	次のチャネルでグループ続行	無効
CH2	ADC[0]_15	グループの最後のチャネル	グループ Done 割込み有効

- ADC Done レベル: パルス
- 外部アナログ MUX: 0, 有効
- プレコンディショニングモード: OFF
- オーバラップ診断モード: OFF
- キャリブレーション値の選択: 通常
- ポストプロセッシングモード: なし
- 結果の配置: 右
- 符号拡張: 符号なし
- 平均カウント: 0
- 右シフト: 0

- マスクグループ: 完了/キャンセルなし/オーバフローなし
- マスクチャネル: 範囲外/パルスなし/オーバフローなし

4.1.2 コンフィグレーション

CYT2 シリーズのグループのロジカルチャネル設定のパラメータを表 11 に、関数を表 12 に示します。

表 11 CYT2 シリーズのグループのロジカルチャネル設定パラメータ

パラメータ	説明	値
ADC_LOGICAL_CHANNEL	ADC ロジカルチャネル	0ul
ADC_GROUP_NUMBER_OF_CHANNELS	グループ内の ADC チャネル数	3ul
ADC_GROUP_FIRST_CHANNEL	ADC グループの最初のチャネル	ADC_LOGICAL_CHANNEL
ADC_GROUP_LAST_CHANNEL	ADC グループの最後のチャネル	ADC_GROUP_FIRST_CHANNEL + ADC_GROUP_NUMBER_OF_CHANNELS - 1ul
BB_POTI_ANALOG_MACRO	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログマクロ	CY_ADC_POT_MACRO: PASS0_SAR0
adcChannelConfig.triggerSelection	トリガ OFF	0ul (表 3 参照)
adcChannelConfig.channelPriority	チャネル優先度	0ul
adcChannelConfig.preenptionType	Pre-emption タイプ	3ul (表 3 参照)
adcChannelConfig.isGroupEnd	グループ終了？	CH0 = false, CH1 = false, CH2: true
adcChannelConfig.doneLevel	Done レベル	0ul (表 3 参照)
adcChannelConfig.pinAddress	ピンアドレス	BB_POTI_ANALOG_INPUT_NO
adcChannelConfig.portAddress	ポートアドレス	0ul (表 3 参照)
adcChannelConfig.extMuxSelect	外部 MUX 選択	0ul
adcChannelConfig.extMuxEnable	外部 MUX 有効	true
adcChannelConfig.preconditionMode	プリコンディションモード	0ul (表 3 参照)
adcChannelConfig.overlapDiagMode	オーバラップ診断モード	0ul (表 3 参照)
adcChannelConfig.sampleTime	サンプル時間	samplingCycle (計算値)
adcChannelConfig.calibrationValueSelect	キャリブレーション値選択	0ul (表 3 参照)
adcChannelConfig.postProcessingMode	ポストプロセッシングモード	0ul (表 3 参照)
adcChannelConfig.resultAlignment	結果の配置	0ul (表 3 参照)
adcChannelConfig.signExtention	符号拡張	0ul (表 3 参照)
adcChannelConfig.averageCount	平均カウント	0ul

(続く)

表 11 (続き) CYT2 シリーズのグループのロジカルチャネル設定パラメータ

パラメータ	説明	値
adcChannelConfig.rightShift	右シフト	0ul
adcChannelConfig.rangeDetectionMode	レンジ検出モード	2ul (表 3 参照)
adcChannelConfig.rangeDetectionLoThreshold	レンジ検出閾値 (下限)	0x0000ul
adcChannelConfig.rangeDetectionHiThreshold	レンジ検出閾値 (上限)	0x0FFFul
adcChannelConfig.mask.grpDone	マスクグループ Done	CH0 = false, CH1 = false, CH2: true
adcChannelConfig.mask.grpCancelled	マスクグループキャンセル	false
adcChannelConfig.mask.grpOverflow	マスクグループオーバフロー	false
adcChannelConfig.mask.chRange	マスクチャネルレンジ	false
adcChannelConfig.mask.chPulse	マスクチャネルパルス	false
adcChannelConfig.mask.chOverflow	マスクチャネルオーバフロー	false

表 12 CYT2 シリーズのグループのロジカルチャネル設定関数

関数	説明	値
Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)	ADC チャネルの初期化	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig
Cy_SysInt_InitIRQ(Config)	割込みベクタを設定し参照されるシステム割込みを初期化	Config = irq_cfg
Cy_Adc_Channel_Enable(SARchannel)	対応チャネル有効	SARchannel = PASS_SAR_CH
Cy_Adc_Channel_SoftwareTrigger(PASS SARchannel)	ソフトウェア開始トリガ発行	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

4.1.3 サンプルコード

グループ手順のロジカルチャネル設定の初期設定のサンプルコードについては、Code listing 22 を参照してください。

Code listing 22 グループ手順のロジカルチャネル設定

```
#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];
uint8_t                resultIdx;

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

    /* Initialize ADC */
    {
:
        cy_stc_adc_config_t adcConfig =  /* ADC Configuration. */
        {
            .preconditionTime    = 0ul,
            .powerupTime         = 0ul,
            .enableIdlePowerDown = false,
            .msbStretchMode      = CY_ADC_MSB_STRETCH_MODE_2CYCLE,


            .enableHalfLsbConv   = 0ul,
            .sarMuxEnable        = true,
            .adcEnable           = true,
            .sarIpEnable         = true,
        };
        cy_stc_adc_channel_config_t adcChannelConfig =
        {
            .triggerSelection          = CY_ADC_TRIGGER_OFF,  /* (1) Trigger Type Setting */
            .channelPriority           = 0ul,
            .preenptionType            = CY_ADC_PREEMPTION_FINISH_RESUME,  /* Preemption Type
Setting */
            .isGroupEnd                = false,  /* Group End Setting */
            .doneLevel                 = CY_ADC_DONE_LEVEL_PULSE,   /* Level or Pulse
Selection  */
            .pinAddress                = BB_POTI_ANALOG_INPUT_NO,  /* (2) Analog Input Setting,
Sample Time Setting */
            .portAddress               = CY_ADC_PORT_ADDRESS_SARMUX0,  /* (2) Analog Input
Setting, Sample Time Setting */
            .extMuxSelect              = 0ul,  /* (2) Analog Input Setting, Sample Time Setting
*/
            .extMuxEnable              = true,  /* (2) Analog Input Setting, Sample Time
Setting */
```

```
            .preconditionMode          = CY_ADC_PRECONDITION_MODE_OFF,  /* (2) Analog Input
Setting, Sample Time Setting */
            .overlapDiagMode           = CY_ADC_OVERLAP_DIAG_MODE_OFF,  /* (2) Analog Input
Setting, Sample Time Setting */
            .sampleTime                = samplingCycle,  /* (2) Analog Input Setting, Sample
Time Setting */
            .calibrationValueSelect    = CY_ADC_CALIBRATION_VALUE_REGULAR,  /* (2) Analog Input
Setting, Sample Time Setting */
            .postProcessingMode        = CY_ADC_POST_PROCESSING_MODE_NONE,  /* (2) Analog Input
Setting, Sample Time Setting */
            .resultAlignment           = CY_ADC_RESULT_ALIGNMENT_RIGHT,  /* (2) Analog Input
Setting, Sample Time Setting */
            .signExtention             = CY_ADC_SIGN_EXTENTION_UNSIGNED,  /* (2) Analog Input
Setting, Sample Time Setting */
            .averageCount              = 0u,  /* (2) Analog Input Setting, Sample Time Setting
*/
            .rightShift                = 0u,  /* (2) Analog Input Setting, Sample Time Setting
*/
            .rangeDetectionMode        = CY_ADC_RANGE_DETECTION_MODE_INSIDE_RANGE,  /* (2)
Analog Input Setting, Sample Time Setting */
            .rangeDetectionLoThreshold = 0x0000ul,  /* (2) Analog Input Setting, Sample Time
Setting */
            .rangeDetectionHiThreshold = 0x0FFFul,  /* (2) Analog Input Setting, Sample Time
Setting */



            .mask.grpDone              = false,  /* (3) Interrupt Mask Setting */
            .mask.grpCancelled         = false,  /* (3) Interrupt Mask Setting */
            .mask.grpOverflow          = false,  /* (3) Interrupt Mask Setting */
            .mask.chRange              = false,  /* (3) Interrupt Mask Setting */
            .mask.chPulse              = false,  /* (3) Interrupt Mask Setting */
            .mask.chOverflow           = false,  /* (3) Interrupt Mask Setting */
        };

        Cy_Adc_Init(BB_POTI_ANALOG_MACRO, &adcConfig);  /* ADC Configuration setting. See Code
Listing 2. */


        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 = aenAnalogInput[i];
```

```
            Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ch], &adcChannelConfig);  /* ADC
Channel Initialize. See Code Listing 5. */
        }
:
        Cy_SysInt_InitIRQ(&irq_cfg);  /* (4) Initialize Interrupt Request. See Code Listing 7.
*/
:
    /* 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]);  /* (5) Enable ADC Channel. See Code
Listing 8. */
    }
:



    /* Issue SW trigger */
    Cy_Adc_Channel_SoftwareTrigger(&CY_ADC_POT_MACRO->CH[ADC_GROUP_FIRST_CHANNEL]);  /* (6)
Software Trigger Setting. See Code Listing 9. */
:
    for(;;)
    {
    }
}
```

4.2 グループを使用する場合の A/D 変換割込み処理

図 9 に、グループを使用する場合の A/D 変換割込み処理例を示します。CPU 割込み処理の詳細は、関連資料のアーキテクチャ TRM を参照してください。

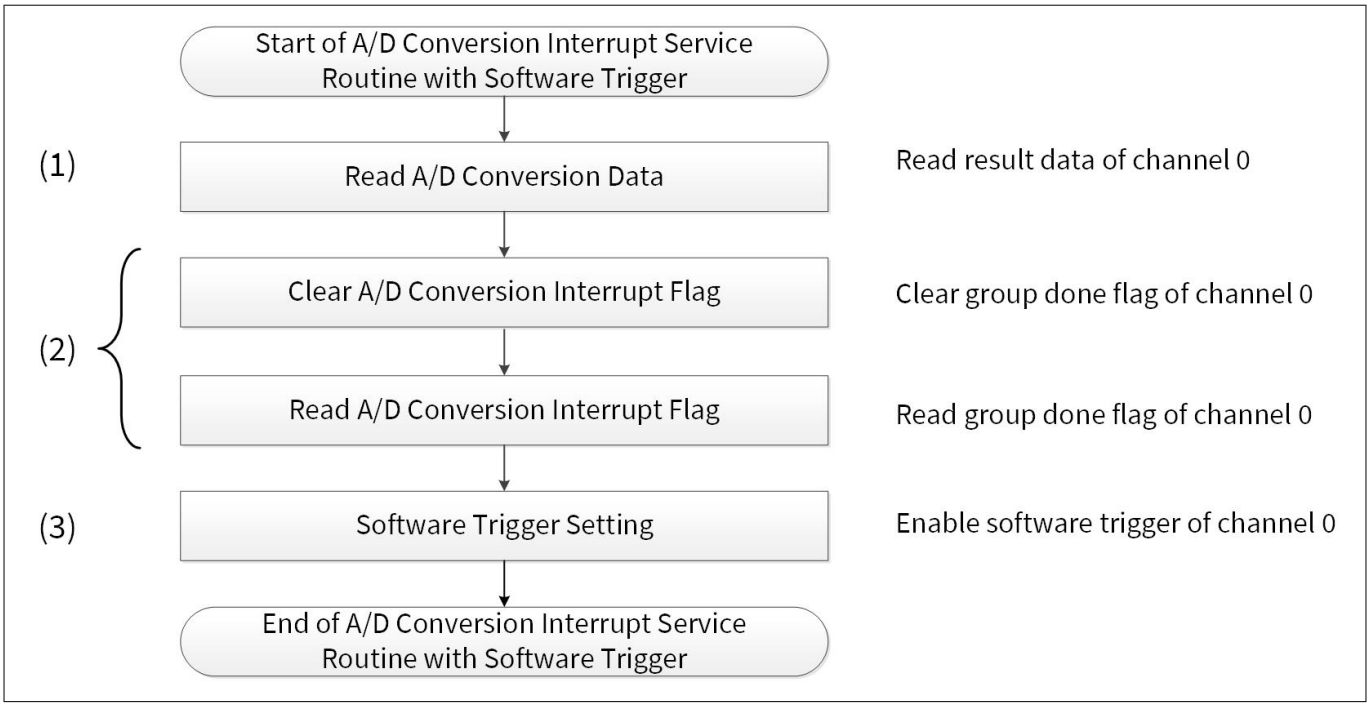


図 9　　グループを使用する場合の A/D 変換割込み処理例

4.2.1　ユースケース

セクション 4.1.1 を参照してください。

4.2.2　コンフィグレーション

グループを使用する場合の A/D 変換割込み処理の SDL の設定部のパラメータを表 13 に、関数を表 14 に示します。

表 13　　グループを使用する場合の A/D 変換割込み処理パラメータ

パラメータ	説明	値
`resultBuff`	変換結果バッファ	- (計算値)
`statusBuff`	変換結果ステータス	- (計算値)
`resultIdx`	インデックス結果	- (計算値)
`intrSource`	割込みソース	- (計算値)

表 14　　グループを使用する場合の A/D 変換割込み処理関数

関数	説明	値
`Cy_Adc_Channel_GetResult(SAR Channel,Result,Status)`	変換結果とステータスの取得	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Result = resultBuff[resultIdx], Status = statusBuff[resultIdx]

(続く)

表 14 (続き) グループを使用する場合の A/D 変換割込み処理関数

関数	説明	値
`Cy_Adc_Channel_ClearInterruptStatus(SAR Channel,Source)`	対応するチャネル割込みステータスのクリア	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Source = intrSource
`Cy_Adc_Channel_SoftwareTrigger(SAR Channel)`	ソフトウェア開始トリガ発行	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

4.2.3 サンプルコード

グループ設定を使用する場合の ADC ISR の初期設定のサンプルコードについては、Code listing 23 を参照してください。

Code listing 23 グループ設定を使用する場合の ADC ISR

```
#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]);  /* (1) Read A/D conversion data. See Code Listing 11. */
            printf("AN%d = %04d, ", aenAnalogInput[i], resultBuff[i][resultIdx]);
        }

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



        /* Trigger next conversion */
        Cy_Adc_Channel_SoftwareTrigger(&BB_POTI_ANALOG_MACRO->CH[ADC_GROUP_FIRST_CHANNEL]);  /*
(3) Software Trigger Setting. See Code Listing 9. */
    }
    else
    {
        // Unexpected interrupt
        CY_ASSERT(false);
    }
}
```

5 平均化処理

平均化処理は、SARn_CHx_POST_CTL レジスタによってチャネルごとに設定できます。

平均化されるサンプル数の最大は 256 です。

ここでは、MCU の ADC[0]_0 端子に入力された電圧を平均化したデジタル値に変換するアプリケーション例を紹介します。

このアプリケーションの物理設定は図 1 に示されるものと同じです。

基本的な ADC グローバル設定, ユースケース, およびソフトウェアトリガを使用する場合の A/D 変換割込み処理に記載される設定にしていることを確認してください。

このアプリケーションを実装するためには、以下のセクションの情報と例を使用してください。

5.1 平均化の設定

図 10 に、平均化の設定例を示します。設定された平均カウント分の ADC 結果が累積されます。その累積された結果は、設定された右シフトビット分、右にシフトされます。その後、それは結果レジスタに格納されます。真の平均化を行うには、平均カウントを 2 の累乗にし、それに対応した値を右シフトに設定する必要があります。2 の累乗でない平均カウントの場合、右シフトは要求された除算だけで概算します。このケースで真の平均化結果が必要な場合、ソフトウェアが除算を行う必要があります。

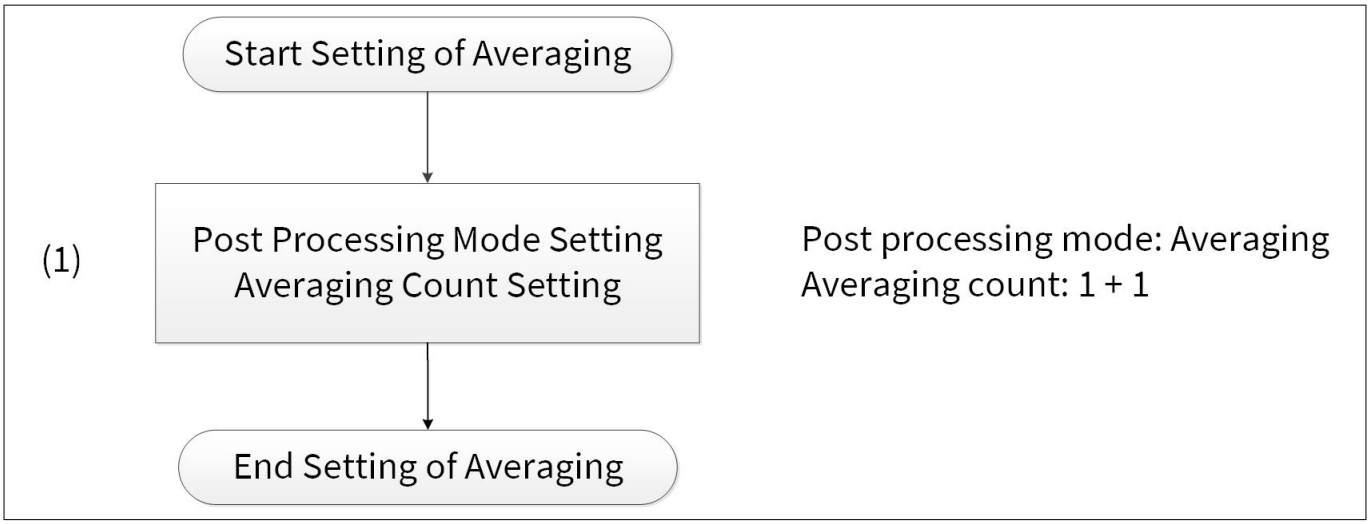


図 10 平均化の設定例

5.1.1 ユースケース

以下のユースケースは平均化の設定の例を示します。

- ポストプロセッシングモード: 平均化
- 平均カウント: 1 + 1 回
- 右シフト: 1
- その他のユースケースアイテムはセクション 2.3.1 と同様です。

5.1.2 コンフィグレーション

平均化処理を使用する場合の SDL の設定部のパラメータを表 15 に、関数を表 16 に示します。

5 平均化処理

表 15　　平均化処理のパラメータ

パラメータ	説明	値
BB_POTI_ANALOG_MACRO	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログマクロ	CY_ADC_POT_MACRO: PASS0_SAR0
adcChannelConfig.triggerSelection	トリガ OFF	0ul (表 3 参照)
adcChannelConfig.channelPriority	チャネル優先度	0ul
adcChannelConfig.preenptionType	Pre-emption タイプ	3ul (表 3 参照)
adcChannelConfig.isGroupEnd	グループ終了？	true
adcChannelConfig.doneLevel	Done レベル	0ul (表 3 参照)
adcChannelConfig.pinAddress	ピンアドレス	BB_POTI_ANALOG_INPUT_NO
adcChannelConfig.portAddress	ポートアドレス	0ul (表 3 参照)
adcChannelConfig.extMuxSelect	外部 MUX 選択	0ul
adcChannelConfig.extMuxEnable	外部 MUX 有効	true
adcChannelConfig.preconditionMode	プリコンディションモード	0ul (表 3 参照)
adcChannelConfig.overlapDiagMode	オーバラップ診断モード	0ul (表 3 参照)
adcChannelConfig.sampleTime	サンプル時間	0ul
adcChannelConfig.calibrationValueSelect	キャリブレーション値選択	0ul (表 3 参照)
adcChannelConfig.postProcessingMode	ポストプロセッシングモード	1ul (表 3 参照)
adcChannelConfig.resultAlignment	結果の配置	0ul (表 3 参照)
adcChannelConfig.signExtention	符号拡張	0ul (表 3 参照)
adcChannelConfig.averageCount	平均カウント	1ul
adcChannelConfig.rightShift	右シフト	1ul
adcChannelConfig.rangeDetectionMode	レンジ検出モード	0ul (表 3 参照)
adcChannelConfig.rangeDetectionLoThreshold	レンジ検出閾値 (下限)	0x0000ul
adcChannelConfig.rangeDetectionHiThreshold	レンジ検出閾値 (上限)	0x0FFFul
adcChannelConfig.mask.grpDone	マスクグループ Done	true
adcChannelConfig.mask.grpCancelled	マスクグループキャンセル	false
adcChannelConfig.mask.grpOverflow	マスクグループオーバフロー	false
adcChannelConfig.mask.chRange	マスクチャネルレンジ	false
adcChannelConfig.mask.chPulse	マスクチャネルパルス	false
adcChannelConfig.mask.chOverflow	マスクチャネルオーバフロー	false

表 16 平均化処理の関数

関数	説明	値
`Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)`	ADC チャネルの初期化	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig

5.1.3 サンプルコード

平均化処理設定の初期設定のサンプルコードについては、Code listing 24 を参照してください。

5 平均化処理

Code listing 24 平均化処理設定

```
:
/* 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              = 1ul, /* 1+1 times*/  /* (1) Averaging Settings */
    .rightShift                = 1ul,
    .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)
{
    /* Enable global interrupts. */
    __enable_irq();

    SystemInit();

    Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &adcChannelConfig);  /*
ADC Channel Initialize. See Code Listing 5. */
:
    for(;;);
}
```

6 レンジ検出処理

レンジ検出は、CPU の介在なしで、レジスタに設定した上限閾値と下限閾値に対する確認ができます。

各ロジカルチャネルは、個別にその検出範囲を設定できます。この機能は、電圧の異常を監視するために使用されます。

ここでは、MCU の ADC[0]_0 端子に入力された電圧をデジタル値に変換するアプリケーション例を紹介します。図 11 に示すように、A/D 変換値が"2500"以上、または"1500"未満の場合、その値を読み出すために割込みを発生します。A/D 変換は、対応している TCPWM のハードウェアトリガにより一定の時間間隔で繰り返されます。

このアプリケーションの物理設定は図 1 に示されるものと同じです。

基本的な ADC グローバル設定およびハードウェアトリガを使用する場合のロジカルチャネル設定に記載される設定にしていることを確認してください。

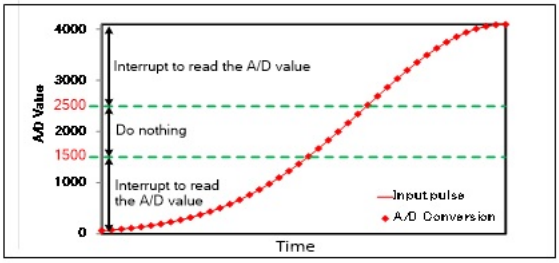


図 11 レンジ検出アプリケーション動作の例

このアプリケーションを実装するためには、以下のセクションの情報と例を使用してください。

6.1 レンジ検出の設定

図 12 に、レンジ検出の設定例を示します。

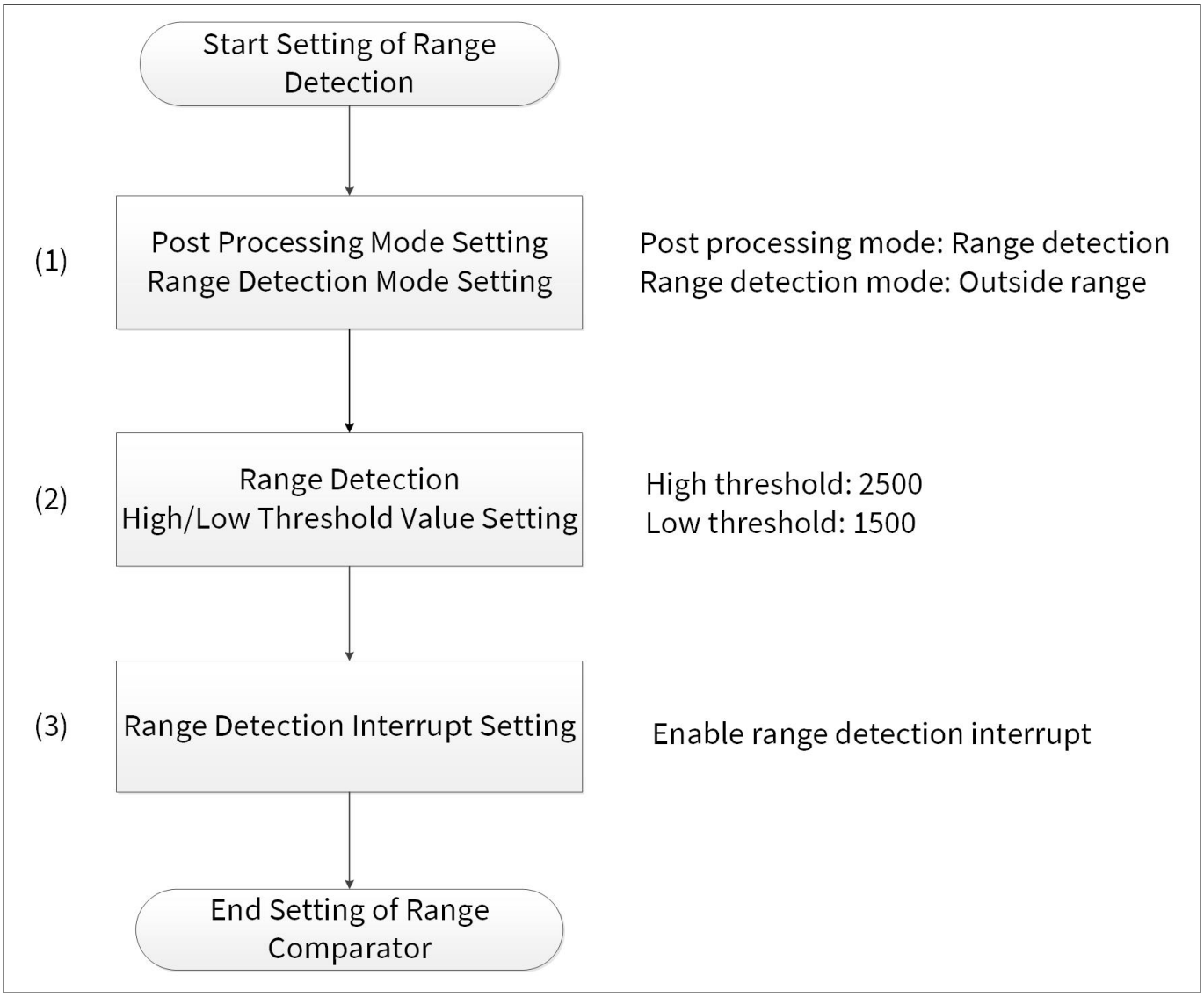


図 12　　レンジ検出の設定例

6.1.1　ユースケース

以下のユースケースは平均化の設定の例を示します。

- レンジ検出閾値: Low: 1500 / High: 2500
- その他のユースケースアイテムは、セクション 2.2.1 およびセクション 3.1.1 と同様です。

6.1.2　コンフィグレーション

レンジ検出を使用する場合の SDL の設定部のパラメータを表 17 に、関数を表 18 に示します。

表 17　　レンジ検出のパラメータ

パラメータ	説明	値
BB_POTI_ANALOG_MACRO	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログマクロ	CY_ADC_POT_MACRO: PASS0_SAR0

(続く)

表 17 (続き) レンジ検出のパラメータ

パラメータ	説明	値
BB_POTI_ANALOG_INPUT_NO	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログ入力数	CH0 (CH[ADC_LOGICAL_CHANNEL])
adcChannelConfig.triggerSelection	トリガ OFF	0ul (表 3 参照)
adcChannelConfig.channelPriority	チャネル優先度	0ul
adcChannelConfig.preenptionType	Pre-emption タイプ	3ul (表 3 参照)
adcChannelConfig.isGroupEnd	グループ終了？	true
adcChannelConfig.doneLevel	Done レベル	0ul (表 3 参照)
adcChannelConfig.pinAddress	ピンアドレス	BB_POTI_ANALOG_INPUT_NO
adcChannelConfig.portAddress	ポートアドレス	0ul (表 3 参照)
adcChannelConfig.extMuxSelect	外部 MUX 選択	0ul
adcChannelConfig.extMuxEnable	外部 MUX 有効	true
adcChannelConfig.preconditionMode	プリコンディションモード	0ul (表 3 参照)
adcChannelConfig.overlapDiagMode	オーバラップ診断モード	0ul (表 3 参照)
adcChannelConfig.sampleTime	サンプル時間	0ul
adcChannelConfig.calibrationValueSelect	キャリブレーション値選択	0ul (表 3 参照)
adcChannelConfig.postProcessingMode	ポストプロセッシングモード	3ul (表 3 参照)
adcChannelConfig.resultAlignment	結果の配置	0ul (表 3 参照)
adcChannelConfig.signExtention	符号拡張	0ul (表 3 参照)
adcChannelConfig.averageCount	平均カウント	0ul
adcChannelConfig.rightShift	右シフト	0ul
adcChannelConfig.rangeDetectionMode	レンジ検出モード	3ul (表 3 参照)
adcChannelConfig.rangeDetectionLoThreshold	レンジ検出閾値 (下限)	0x09C4ul
adcChannelConfig.rangeDetectionHiThreshold	レンジ検出閾値 (上限)	0x05DCul
adcChannelConfig.mask.grpDone	マスクグループ Done	false
adcChannelConfig.mask.grpCancelled	マスクグループキャンセル	false
adcChannelConfig.mask.grpOverflow	マスクグループオーバフロー	false
adcChannelConfig.mask.chRange	マスクチャネルレンジ	true
adcChannelConfig.mask.chPulse	マスクチャネルパルス	false
adcChannelConfig.mask.chOverflow	マスクチャネルオーバフロー	false

表 18 レンジ検出の関数

関数	説明	値
`Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)`	ADC チャネルの初期化	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig
`Cy_SysInt_InitIRQ(Config)`	割込みベクタを設定し参照されるシステム割込みを初期化	Config = irq_cfg
`Cy_Adc_Channel_Enable(PASS SARchannel)`	対応チャネル有効	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]
`Cy_Adc_Channel_SoftwareTrigger(PASS SARchannel)`	ソフトウェア開始トリガ発行	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

6.1.3 サンプルコード

レンジ検出設定の初期設定のサンプルコードについては、Code listing 25 を参照してください。

6 レンジ検出処理

Code listing 25 レンジ検出設定

```
 :
#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 */  /*
(1) Post Processing Mode Setting. */
    .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 */  /* (2) Range Detection Value Setting. */
    .rangeDetectionHiThreshold = 0x05DCul, /* 1500 */
    .mask.grpDone              = false,
    .mask.grpCancelled         = false,
    .mask.grpOverflow          = false,
    .mask.chRange              = true, /* Range detection interrupt */  /* (3) Range Detection
Interrupt Setting. */
    .mask.chPulse              = false,
    .mask.chOverflow           = false,
};
:
int main(void)
{
    __enable_irq(); /* Enable global interrupts. */

    SystemInit();
 :
    /* Initialize ADC */  /* Initialize ADC Channel. See Code Listing 5. */
    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  = (cy_en_intr_t)((uint32_t)CY_ADC_POT_IRQN + ADC_LOGICAL_CHANNEL),
            .intIdx     = CPUIntIdx3_IRQn,
            .isEnabled  = true,
    };
    Cy_SysInt_InitIRQ(&irq_cfg);  /* Initialize Interrupt Request. See Code Listing 7. */

 :

    /* Enable ADC ch. */  /* Enable ADC Channel. See Code Listing 8. */
    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]);  /*
Software Trigger Setting. See Code Listing 9. */

:

    for(;;);
}
```

6.2 レンジ検出を使用する場合の A/D 変換割込み処理

図 13 に、レンジ検出を使用する場合の A/D 変換割込み処理例を示します。CPU 割込み処理の詳細は、関連資料のアーキテクチャ TRM を参照してください。

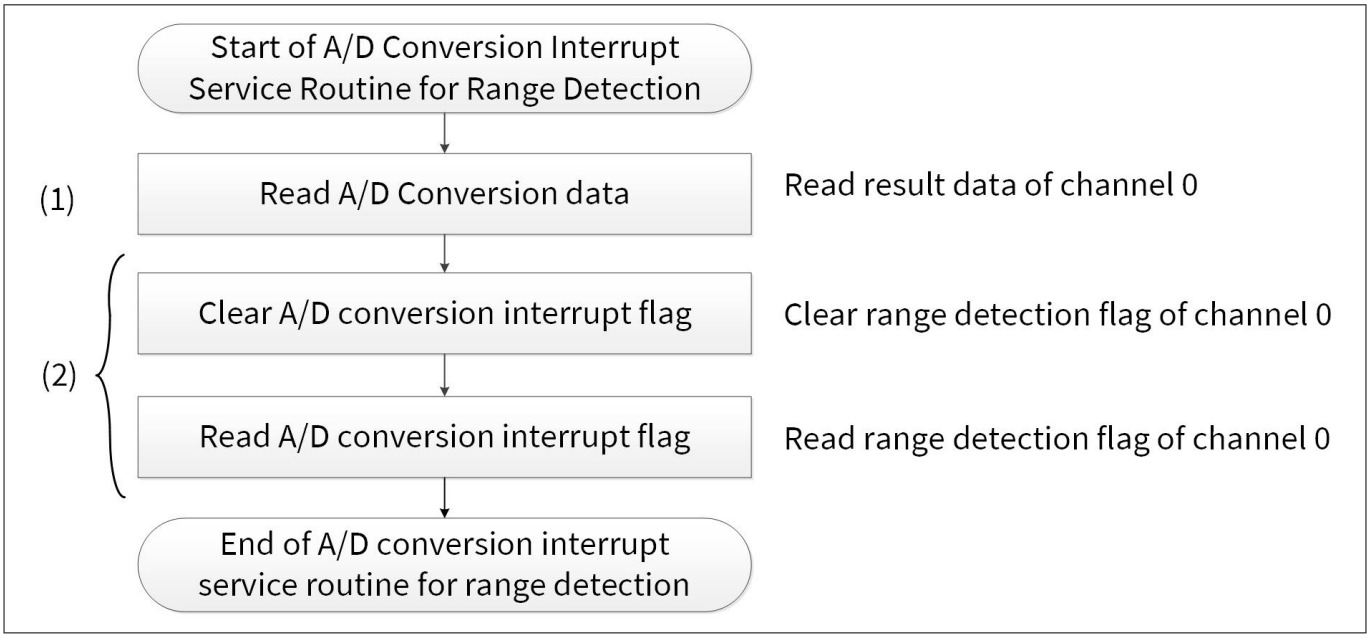


図 13 レンジ検出を使用する場合の A/D 変換割込み処理例

6.2.1 コンフィグレーション

レンジ検出を使用する場合の A/D 変換割込み処理の SDL の設定部のパラメータを表 19 に、関数を表 20 に示します。

表 19　　レンジ検出を使用する場合の A/D 変換割込み処理パラメータ

パラメータ	説明	値
BB_POTI_ANALOG_MACRO	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログマクロ	CY_ADC_POT_MACRO: PASS0_SAR0
intrSource	割込みソース	- (計算値)
resultIdx	インデックス結果	- (計算値)

表 20　　レンジ検出を使用する場合の A/D 変換割込み処理関数

関数	説明	値
Cy_Adc_Channel_GetInterruptMaskedStatus(PASS SARchannel,INTR Source)	割込みステータス取得	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] INTR Source = &intrSource
Cy_Adc_Channel_GetResult(SAR Channel,Result,Status)	変換結果とステータスの取得	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Result = resultBuff[resultIdx], Status = statusBuff[resultIdx]
Cy_Adc_Channel_ClearInterruptStatus(SAR Channel,Source)	対応するチャネル割込みステータスのクリア	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Source = intrSource

6.2.2　　サンプルコード

レンジ検出を使用する場合の ADC ISR の初期設定のサンプルコードについては、Code listing 26 を参照してください。

6 レンジ検出処理

Code listing 26 レンジ検出を使用する場合の ADC ISR

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



    /* Get interrupt source */  /* Get Interrupt Masked Status. See Code Listing 14. */
    Cy_Adc_Channel_GetInterruptMaskedStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&intrSource);

    if(intrSource.chRange)
    {
        /* Get the result(s) */  /* (1) Read A/D conversion data. See Code Listing 11. */
        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 */  /* (2) Clear and read A/D conversion flag. See Code
Listing 12. */
        Cy_Adc_Channel_ClearInterruptStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&intrSource);

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

7 パルス検出処理

パルス検出の比較結果はパルス検出機能によりフィルタをかけられます。

各ロジカルチャネルのパルス検出機能には、ポジティブとネガティブの各ダウンカウンタについて初期値を保持するための 1 組のリロードレジスタがあります。ポジティブとネガティブのカウンタは、レンジ検出による比較結果から得られたポジティブとネガティブイベントにより減少します。

この機能は電圧を監視する際、ノイズなどによる電圧異常の誤検知を避けるために使用されます。

レンジ検出の設定で説明した例の値では、A/D 変換結果が 2500 以上、または 1500 未満の場合がポジティブイベント、1500 以上かつ 2500 未満の場合がネガティブイベントです。

このアプリケーション例では、図 14 に示すように、ポジティブイベントが 5 回連続で発生した場合、割込みが発生します。

図 15 に示すように、ネガティブイベントが 2 回連続で発生すると、ポジティブイベントの連続カウントはキャンセルされます。すなわち、ネガティブイベントが 1 回だけ発生した場合でも、ポジティブイベントのカウントは続きます。

このアプリケーションの物理設定は図 1 に示されるものと同じです。

基本的な ADC グローバル設定およびハードウェアトリガを使用する場合のロジカルチャネル設定に記載される設定にしていることを確認してください。

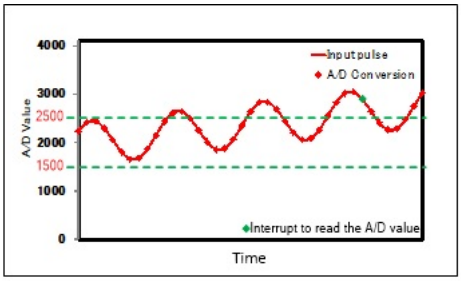


図 14 パルス検出アプリケーション 1 の動作例 パルス検出アプリケーション 2 の動作例

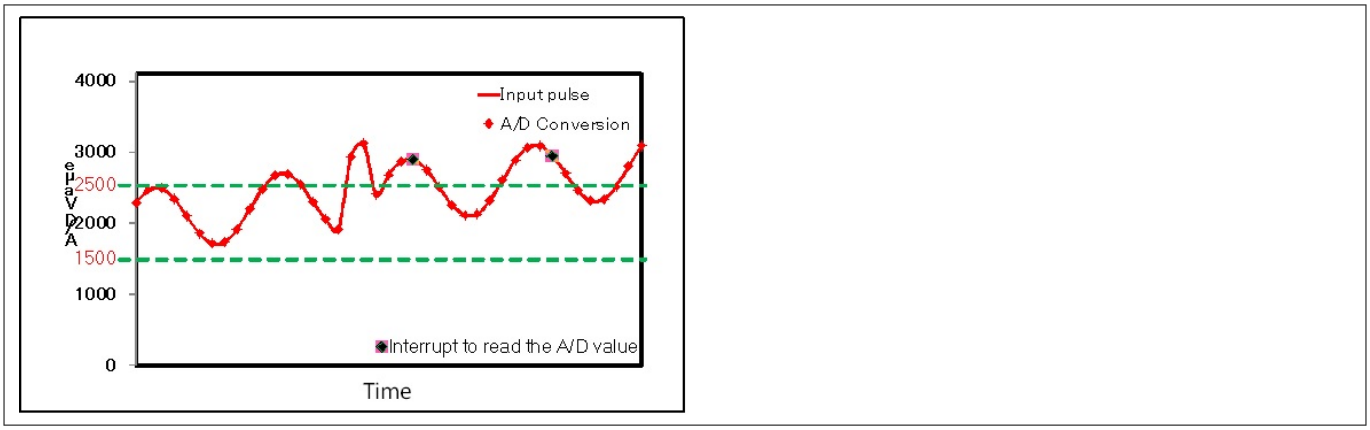


図 15 パルス検出アプリケーション動作の例 2

このアプリケーションを実装するためには、以下のセクションの情報と例を使用してください。

7.1 パルス検出の設定

図 16 に、パルス検出の設定例を示します。

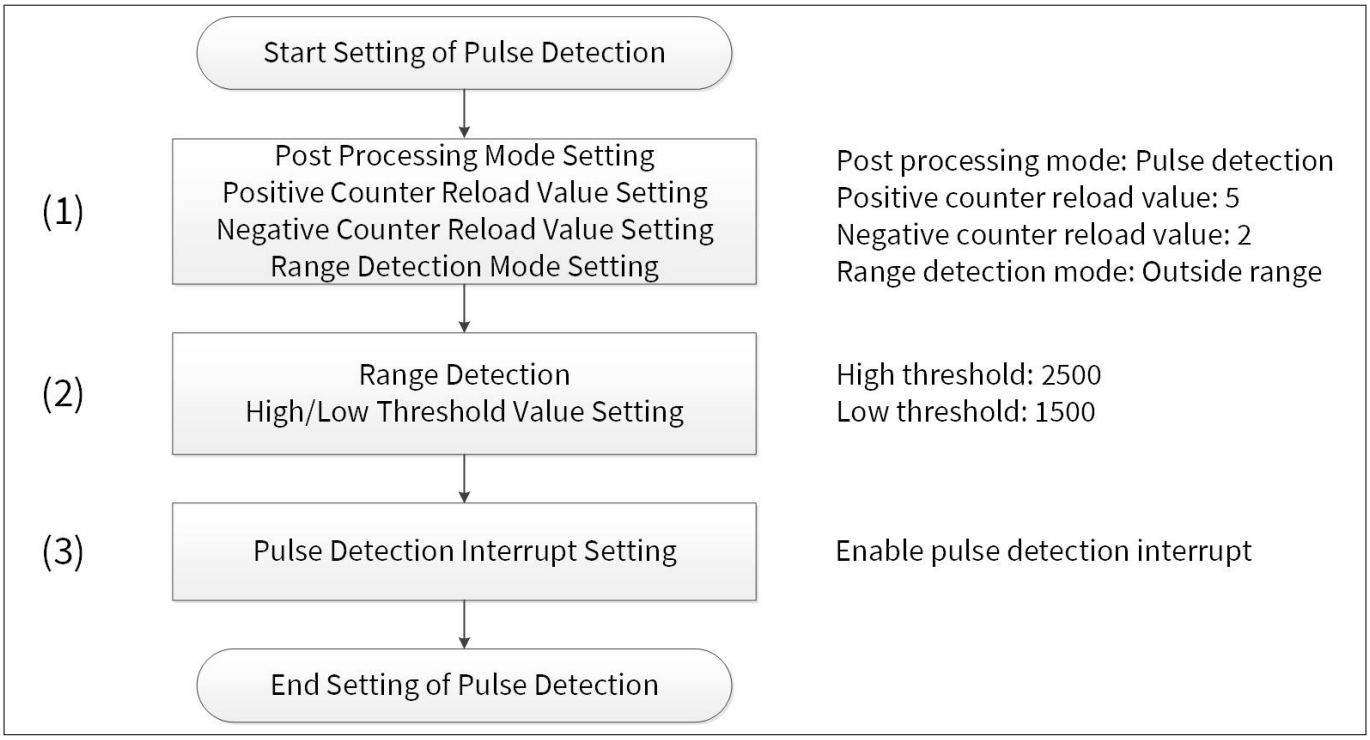


図 16　　パルス検出の設定例

7.1.1　　ユースケース

以下のユースケースはパルス検出の設定の例を示します。

- アナログ入力: ADC[0]_0
- レンジ検出閾値: Low: 1500 / High: 2500
- ADC トリガ: TCPWM
- 正のカウンタリロード値: 5
- 負のカウンタリロード値: 2

その他のユースケースアイテムはセクション 2.2.1 およびセクション 3.1.1 と同様です。

7.1.2　　コンフィグレーション

パルス検出設定を使用する場合の A/D 変換割込み処理の SDL の設定部のパラメータを表 21 に、関数を表 22 に示します。

表 21　　パルス検出設定パラメータ

パラメータ	説明	値
BB_POTI_ANALOG_MACRO	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログマクロ	CY_ADC_POT_MACRO: PASS0_SAR0
BB_POTI_ANALOG_INPUT_NO	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログ入力数	CH0 (CH[ADC_LOGICAL_CHANNEL])
adcChannelConfig.triggerSelection	トリガ TCPWM	1ul (表 3 参照)

(続く)

表 21 (続き) パルス検出設定パラメータ

パラメータ	説明	値
adcChannelConfig.channelPriority	チャネル優先度	0ul
adcChannelConfig.preenptionType	Pre-emption タイプ	3ul (表 3 参照)
adcChannelConfig.isGroupEnd	グループ終了？	true
adcChannelConfig.doneLevel	Done レベル	0ul (表 3 参照)
adcChannelConfig.pinAddress	ピンアドレス	BB_POTI_ANALOG_INPUT_NO
adcChannelConfig.portAddress	ポートアドレス	0ul (表 3 参照)
adcChannelConfig.extMuxSelect	外部 MUX 選択	0ul
adcChannelConfig.extMuxEnable	外部 MUX 有効	true
adcChannelConfig.preconditionMode	プリコンディションモード	0ul (表 3 参照)
adcChannelConfig.overlapDiagMode	オーバラップ診断モード	0ul (表 3 参照)
adcChannelConfig.sampleTime	サンプル時間	0ul
adcChannelConfig.calibrationValueSelect	キャリブレーション値選択	0ul (表 3 参照)
adcChannelConfig.postProcessingMode	ポストプロセッシングモード	4ul (表 3 参照)
adcChannelConfig.resultAlignment	結果の配置	0ul (表 3 参照)
adcChannelConfig.signExtention	符号拡張	0ul (表 3 参照)
adcChannelConfig.averageCount	平均カウントはパルス検出モードでの正のカウンタリロード値	5ul
adcChannelConfig.rightShift	右シフトは、パルス検出モードでの負のカウンタリロード値	2ul
adcChannelConfig.rangeDetectionMode	レンジ検出モード	3ul (表 3 参照)
adcChannelConfig.rangeDetectionLoThreshold	レンジ検出閾値 (下限)	0x05DCul
adcChannelConfig.rangeDetectionHiThreshold	レンジ検出閾値 (上限)	0x09C4ul
adcChannelConfig.mask.grpDone	マスクグループ Done	false
adcChannelConfig.mask.grpCancelled	マスクグループキャンセル	false
adcChannelConfig.mask.grpOverflow	マスクグループオーバフロー	false
adcChannelConfig.mask.chRange	マスクチャネルレンジ	false
adcChannelConfig.mask.chPulse	マスクチャネルパルス	true
adcChannelConfig.mask.chOverflow	マスクチャネルオーバフロー	false

表 22 パルス検出設定関数

関数	説明	値
`Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)`	ADC チャネルの初期化	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig
`Cy_SysInt_InitIRQ(Config)`	割込みベクタを設定し参照されるシステム割込みを初期化	Config = irq_cfg
`Cy_Adc_Channel_Enable(PASS SARchannel)`	対応チャネル有効	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]
`Cy_Tcpwm_Pwm_Enable(Group Counter)`	TCPWM ブロックを初期化解除	Group Counter = TCPWMx_GRPx_CNTx_PWM
`Cy_Tcpwm_TriggerStart(Group Counter)`	選択した TCPWM でソフトウェア開始をトリガ	Group Counter = TCPWMx_GRPx_CNTx_PWM

7.1.3 サンプルコード

Code listing 27 を参照してください。

7 パルス検出処理

Code listing 27 パルス検出設定

```
 :
#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
*/  /* (1) Post Processing 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 */  /* (2) Range Detection Value Setting.*/
    .rangeDetectionHiThreshold = 0x09C4ul, /* 2500 */
    .mask.grpDone              = false,
    .mask.grpCancelled         = false,
    .mask.grpOverflow          = false,
    .mask.chRange              = false,
    .mask.chPulse              = true, /* Enable pulse detection interrupt */  /* (3) Pulse
Detection Interrupt Setting. */
    .mask.chOverflow           = false,
};
:
int main(void)
{
    __enable_irq(); /* Enable global interrupts. */

    SystemInit();
:
    /* 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);  /*
ADC Channel Initialize. See Code Listing 5. */

    /* 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);  /* Initialize Interrupt Request. See Code Listing 7. */
:
    /* Clock Configuration for TCPWMs */
    uint32_t sourceFreq = 80000000ul;
    uint32_t targetFreq = 2000000ul;
    uint32_t divNum_PWM = (sourceFreq / targetFreq);
    CY_ASSERT((sourceFreq % targetFreq) == 0ul); // target frequency accuracy

:

    /* Enable ADC ch. and PWM */
    Cy_Adc_Channel_Enable(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);   /* Enable ADC
Channel. See Code Listing 8. */
    Cy_Tcpwm_Pwm_Enable(TCPWMx_GRPx_CNTx_PWM);  /* TCPWM PWM Enable. See Code Listing 18. */
    Cy_Tcpwm_TriggerStart(TCPWMx_GRPx_CNTx_PWM);  /* TCPWM Trigger Start Select. See Code
Listing 19. */

:

    for(;;);
}
```

7.2 パルス検出を使用する場合の A/D 変換割込み処理

図 17 に、パルス検出を使用する場合の A/D 変換割込み処理例を示します。CPU 割込み処理の詳細は、関連資料のアーキテクチャ TRM を参照してください。

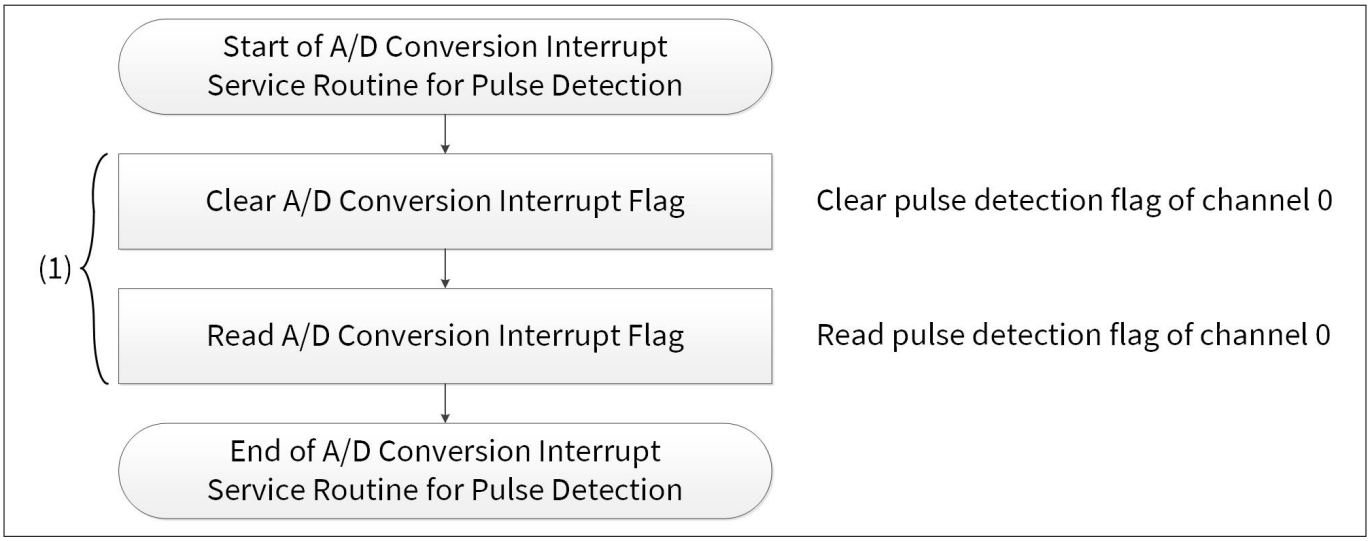


図 17　　パルス検出を使用する場合の A/D 変換割込み処理例

7.2.1　コンフィグレーション

A/D 変換グローバル設定の SDL の設定部のパラメータを表 23 に、関数を表 24 に示します。

表 23　　パルス検出設定を使用する場合の A/D 変換割込み処理パラメータ

パラメータ	説明	値
`BB_POTI_ANALOG_MACRO`	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログマクロ	CY_ADC_POT_MACRO: PASS0_SAR0
`intrSource`	割込みソース	- (計算値)
`resultIdx`	インデックス結果	- (計算値)

表 24　　パルス検出を使用する場合の A/D 変換割込み処理関数一覧

関数	説明	値
`Cy_Adc_Channel_GetInterruptMaskedStatus(PASS SARchannel,INTR Source)`	割込みステータス取得	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] INTR Source = &intrSource
`Cy_Adc_Channel_ClearInterruptStatus(SAR Channel,Source)`	対応するチャネル割込みステータスのクリア	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Source = intrSource

7.2.2　サンプルコード

パルス検出を使用する場合の ADC ISR の初期設定のサンプルコードについては、Code listing 28 を参照してください。

7 パルス検出処理

Code listing 28 パルス検出を使用する場合の ADC ISR

```
:
/* 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);  /* Get Interrupt Masked Status. See Code Listing 14. */

    if(intrSource.chPulse)
    {
:
        /* Clear interrupt source */
        Cy_Adc_Channel_ClearInterruptStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&intrSource);  */ (1    ) Clear and read A/D conversion flag. See Code Listing 12. */
    }
    else
    {

        CY_ASSERT(false);
    }
}
```

8 基準電圧を使用した診断

ここでは、基準電圧を使用して、A/D 値が固着しているかどうかを確認するフローチャートと例を示します。

図 18 に示すように、ADC から基準電圧 V_{REFH} と V_{REFL} へ入力できます。

V_{REFH} は上限基準電圧です。その A/D 値は 0xFFF (= 4095) です。

V_{REFL} は下限基準電圧です。その A/D 値は 0x000 (= 0) です。

それらは ADC 診断と ADC キャリブレーションの機能です。

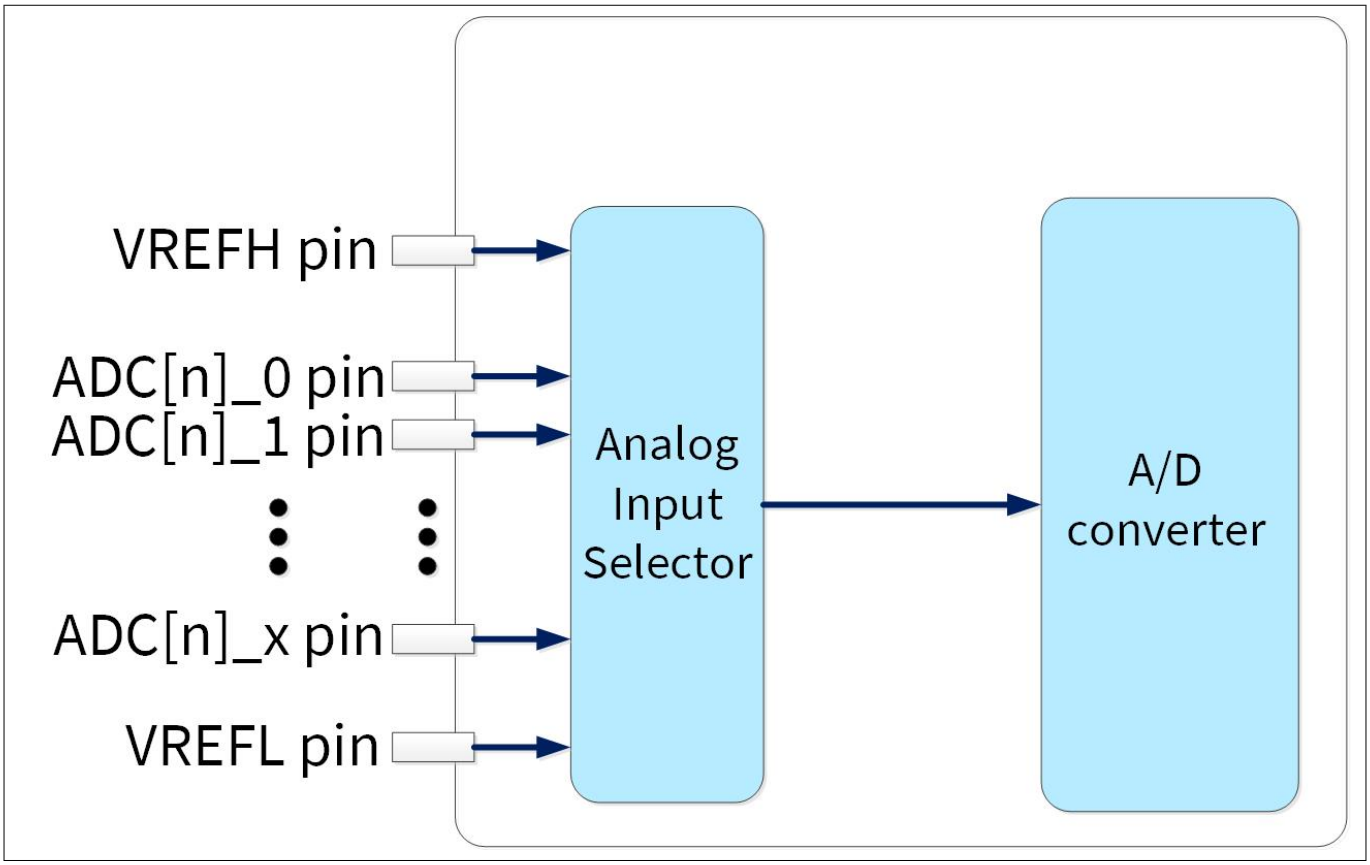


図 18 アナログ入力としての VREFH と VREFL

8.1 閾値の決定

A/D 値の固着診断で閾値を決定するために、この例では、関連資料のデータシートで、ゼロトランジション電圧 (VZT) の最小値とフルスケールトランジション電圧 (VFST) の最大値を参照しています。

データシートによると、VZT の最小値は、– 20 mV です。アナログ入力が 0 V の時、その A/D 値は 16.4 (20 / (5000 / 4096)) です。

すなわち、理想的な V_{REFL} が ADC に入力された場合、その値は 16 以下です。(Threshold L ≦ 16 = 0x010)

V_{FST} の最大値は +20 mV です。アナログ入力が 5 V の時、その A/D 値は、4079.6 ((5000 – 20) / (5000 / 4096)) です。

すなわち、理想的な VREFH が ADC に入力された場合、その値は 4080 以上です。(Threshold H ≧ 4080 = 0xFF0)

注: *これらの閾値は理想的な環境で決定されています。ただし、A/D 値は、電源電圧, 基準電圧, 入力アナログ電圧, 温度, ノイズなどの環境要因の影響を受けます。したがって、最終的な閾値を決定する際は、ユーザシステム環境を考慮し、マージンを付加することを推奨します。*

8.2 ADC 固着診断処理

図 19 に、ADC 固着診断のフローチャート例を示します。この例では、最小のサンプル時間が設定されます。温度センサチャネルが有効になっている場合は、リファレンスバッファモードも有効にする必要があります。システムに最適なサンプル時間を検討するためには、関連資料のデータシートを参照してください。

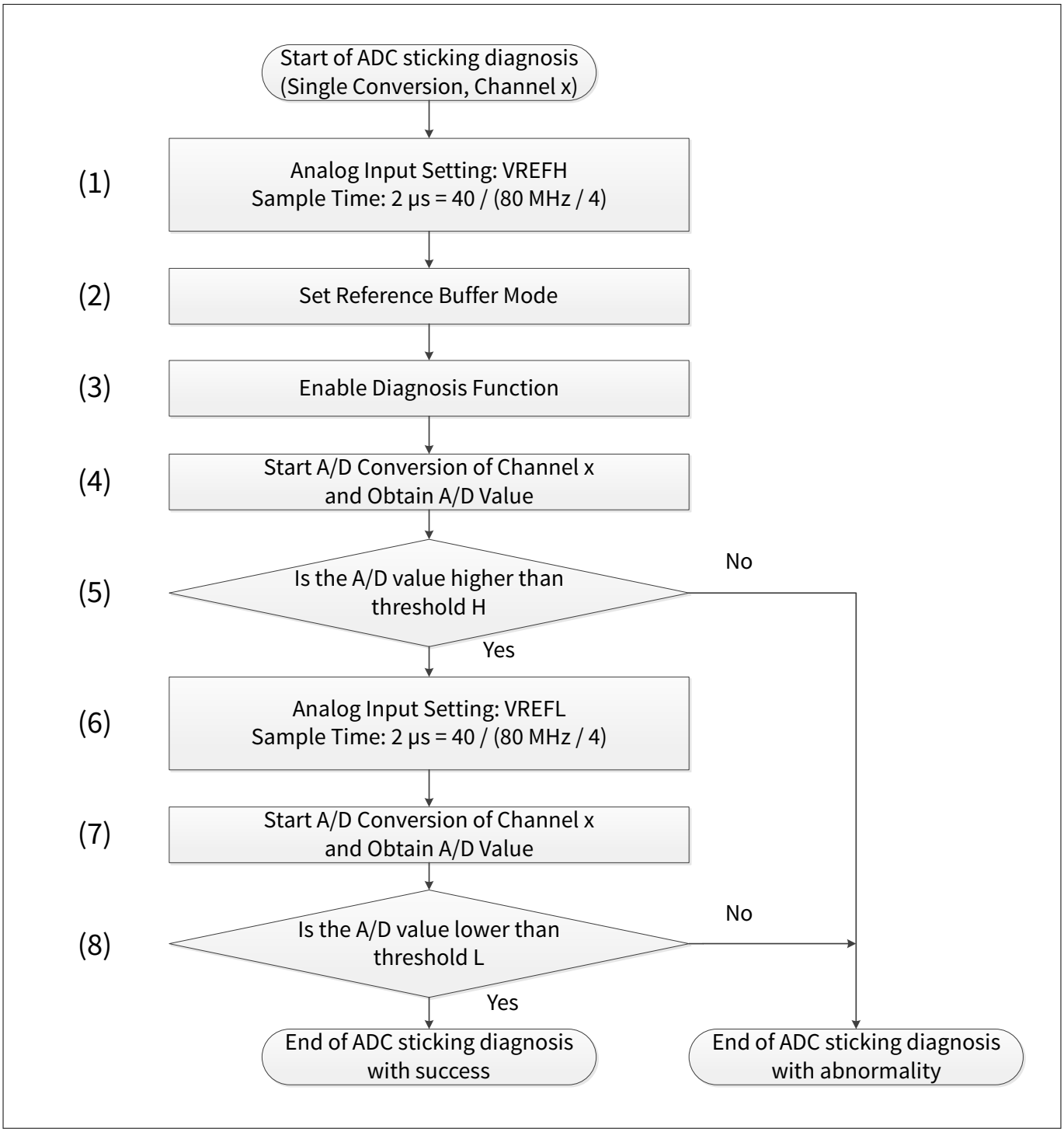


図 19 ADC 固着診断フローチャート例

8.2.1 ユースケース

以下のユースケースは ADC 固着診断設定の例を示します。

- アナログ入力設定
  - VREFH
  - VREFL
- サンプル時間: 2 µs

8.2.2 コンフィグレーション

ADC 固着診断設定の SDL の設定部のパラメータを表 25 に、関数を表 26 示します。

表 25 ADC 固着診断パラメータ

パラメータ	説明	値
adcChannelConfig.triggerSelection	トリガ OFF	0ul (表 3 参照)
adcChannelConfig.channelPriority	チャネル優先度	0ul
adcChannelConfig.preenptionType	Pre-emption タイプ	3ul (表 3 参照)
adcChannelConfig.isGroupEnd	グループ終了？	true
adcChannelConfig.doneLevel	Done レベル	0ul (表 3 参照)
adcChannelConfig.pinAddress	ピンアドレス	VREFH 設定時: CY_ADC_PIN_ADDRESS_VREF_H VREFL 設定時: CY_ADC_PIN_ADDRESS_VREF_L
adcChannelConfig.portAddress	ポートアドレス	0ul (表 3 参照)
adcChannelConfig.extMuxSelect	外部 MUX 選択	0ul
adcChannelConfig.extMuxEnable	外部 MUX 有効	false
adcChannelConfig.preconditionMode	プリコンディションモード	0ul (表 3 参照)
adcChannelConfig.overlapDiagMode	オーバラップ診断モード	0ul (表 3 参照)
adcChannelConfig.sampleTime	サンプル時間	0ul
adcChannelConfig.calibrationValueSelect	キャリブレーション値選択	0ul (表 3 参照)
adcChannelConfig.postProcessingMode	ポストプロセッシングモード	0ul (表 3 参照)
adcChannelConfig.resultAlignment	結果の配置	0ul (表 3 参照)
adcChannelConfig.signExtention	符号拡張	0ul (表 3 参照)
adcChannelConfig.averageCount	平均カウント	0ul
adcChannelConfig.rightShift	右シフト	0ul
adcChannelConfig.rangeDetectionMode	レンジ検出モード	1ul (表 3 参照)
adcChannelConfig.rangeDetectionLoThreshold	レンジ検出閾値 (下限)	0ul
adcChannelConfig.rangeDetectionHiThreshold	レンジ検出閾値 (上限)	0ul
adcChannelConfig.mask.grpDone	マスクグループ Done	false
adcChannelConfig.mask.grpCancelled	マスクグループキャンセル	false

(続く)

表 25 (続き) ADC 固着診断パラメータ

パラメータ	説明	値
adcChannelConfig.mask.grpOverflow	マスクグループオーバフロー	false
adcChannelConfig.mask.chRange	マスクチャネルレンジ	false
adcChannelConfig.mask.chPulse	マスクチャネルパルス	false
adcChannelConfig.mask.chOverflow	マスクチャネルオーバフロー	false
CY_ADC_REF_BUF_MODE_ON	リファレンスバッファモードの選択 0: リファレンスモード未選択 1: SRSS からバッファリングされた Vbg のリファレンス 3: SRSS からバッファリングされていない Vbg のリファレンス	1ul
BB_POTI_ANALOG_MACRO	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログマクロ	CY_ADC_POT_MACRO: PASS0_SAR0

表 26 ADC 固着診断設定関数

関数	説明	値
Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)	ADC チャネルの初期化	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig
Cy_Adc_SetReferenceBufferMode(PASS0_EPASS_MMIO,RefBufMode)	ePASS MMIO リファレンスバッファモード設定	RefBufMode = CY_ADC_REF_BUF_MODE_ON
Cy_Adc_Diag_Enable(AnalogMacro)	診断機能有効	AnalogMacro = BB_POTI_ANALOG_MACRO
Cy_Adc_Channel_Enable(PASS SARchannel)	対応チャネル有効	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]
Cy_Adc_Channel_SoftwareTrigger(PASS SARchannel)	ソフトウェア開始トリガ発行	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]
Cy_Adc_Channel_GetGroupStatus(PASS SARchannel, GroupStatus)	グループ変換ステータス取得	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

(続く)

表 26 (続き) ADC 固着診断設定関数

関数	説明	値
Cy_Adc_Channel_GetResult(SAR Channel,Result,Status)	変換結果とステータスの取得	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] , Result = resultBuff[resultIdx], Status = statusBuff[resultIdx]

8.2.3 サンプルコード

ADC 診断設定の初期構成のサンプル コードについては、Code listing 29～Code listing 32 を参照してください。

8 基準電圧を使用した診断

Code listing 29 ADC 診断設定

```
:
/* 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,
     /* (1) Analog Input Setting = VREFH. */
    .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,
    .mask.chRange              = false,
    .mask.chPulse              = false,
    .mask.chOverflow           = false,
};
:
/* Code Example for ADC Sticking Diagnosis  */

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

    SystemInit();
:
    /* Initialize ADC */
    /* 2us (2000ns) = 40/ *(80MHz/4) */
    uint32_t samplingCycle = (uint32_t)DIV_ROUND_UP((2000 * (uint64_t)actualAdcOperationFreq),
```

```
1000000000ull);
    adcChannelConfig.sampleTime = samplingCycle;

:

/* Initialize ADC Channel. See Code Listing 5 */
    Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &adcChannelConfig);



    /* Set ePASS MMIO reference buffer mode */
    /* (2) Set REF_BUF MODE. See Code Listing 30 */
    Cy_Adc_SetReferenceBufferMode(PASS0_EPASS_MMIO,CY_ADC_REF_BUF_MODE_ON);




    /* Enable diag function. */
    /* (3) Enable ADC Diagnosis. See Code Listing 31. */
    Cy_Adc_Diag_Enable(BB_POTI_ANALOG_MACRO);
    /* Enable ADC ch. */
    /* (4) Enable ADC Channel. See Code Listing 8 */
    Cy_Adc_Channel_Enable(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);
    /* Issue SW trigger */
    /* Software Trigger Setting. See Code Listing 9 */
    Cy_Adc_Channel_SoftwareTrigger(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);


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

}
cy_en_status_t DiagnosisProcedure(void)
{
    status = CY_BAD_PARAM;

    /* Wait Group Done */
    cy_stc_adc_group_status_t Groupstatus = { false };
    do
    {
    /* Returns group conversion status. See Code Listing 32 */
    Cy_Adc_Channel_GetGroupStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],&Groupstatus);
    }
    while(Groupstatus.grpComplete == false);

    /* Get the result(s) */
    /*Read A/D conversion data. See Code Listing 11 */
```

```
    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 threshold L */
        /* (8) Is the A/D value lower than threshold L */
        if(AD_Value <= 16) /* + 20 mV */
        {
            Flag_VREFL = 0;
       Test_Completed = 1;

            status = CY_SUCCESS;
            return(status);
        }
        else
        {
            /*AD Value Higher than threshold L */
            return(CY_BAD_PARAM);
        }
    }

    /* Is the A/D value Higher than threshold H */
    /* (5) Is the A/D value higher than threshold H */
    if(AD_Value >= 4080 && Flag_VREFL == 0) /*  5000 mV - 20 mV */
    {
        Flag_VREFL = 1;

        /* (6) Analog Input Setting = VREFL */
        adcChannelConfig.pinAddress = CY_ADC_PIN_ADDRESS_VREF_L;

/* ADC Channel Initialize. See Code Listing 5 */
        Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &adcChannelConfig);

        /* Enable ADC ch. */

/* (7) Enable ADC Channel. See Code Listing 8 */
        Cy_Adc_Channel_Enable(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);

        /* Trigger next conversion */

/* Software Trigger Setting. See Code Listing 9 */
        Cy_Adc_Channel_SoftwareTrigger(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);
    }
    else
    {
```

```
            /* A/D value lower than threshold H */
            return(CY_BAD_PARAM);
        }
    }
```

Code listing 30 Cy_Adc_SetReferenceBufferMode() 機能

```
void Cy_Adc_SetReferenceBufferMode(volatile stc_PASS_EPASS_MMIO_t * base,
cy_en_adc_ref_buf_mode_t mode)
{
    base->unPASS_CTL.stcField.u2REFBUF_MODE = mode;
}
```

Code listing 31 Cy_Adc_Diag_Enable() 機能

```
void Cy_Adc_Diag_Enable(volatile stc_PASS_SAR_t * base)
{
    base->unDIAG_CTL.stcField.u1DIAG_EN = 1ul;
}
```

Code listing 32 Cy_Adc_Channel_GetGroupStatus() 機能

```
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 キャリブレーション機能

SARn_ANA_CAL レジスタを使用して、ADC のオフセットとゲインを調整できます。
ここでは、レジスタの機能と ADC のキャリブレーションの方法を説明します。

9.1 オフセット調整の説明

ADC はオフセット誤差を補償するためのオフセット調整レジスタを搭載します。
SARn_ANA_CAL[7:0]がそのレジスタです。
SARn_ANA_CAL[7:0]には、+127 から-128 の値を設定できます。
オフセットは 4 分の 1 LSB 単位で調整できます。
出力デジタル値と SARn_ANA_CAL[7:0]値の関係式は以下です。(OFST = SARn_ANA_CAL[7:0]の値)

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

図 20 に、OFST とオフセット変化の関係を示します。

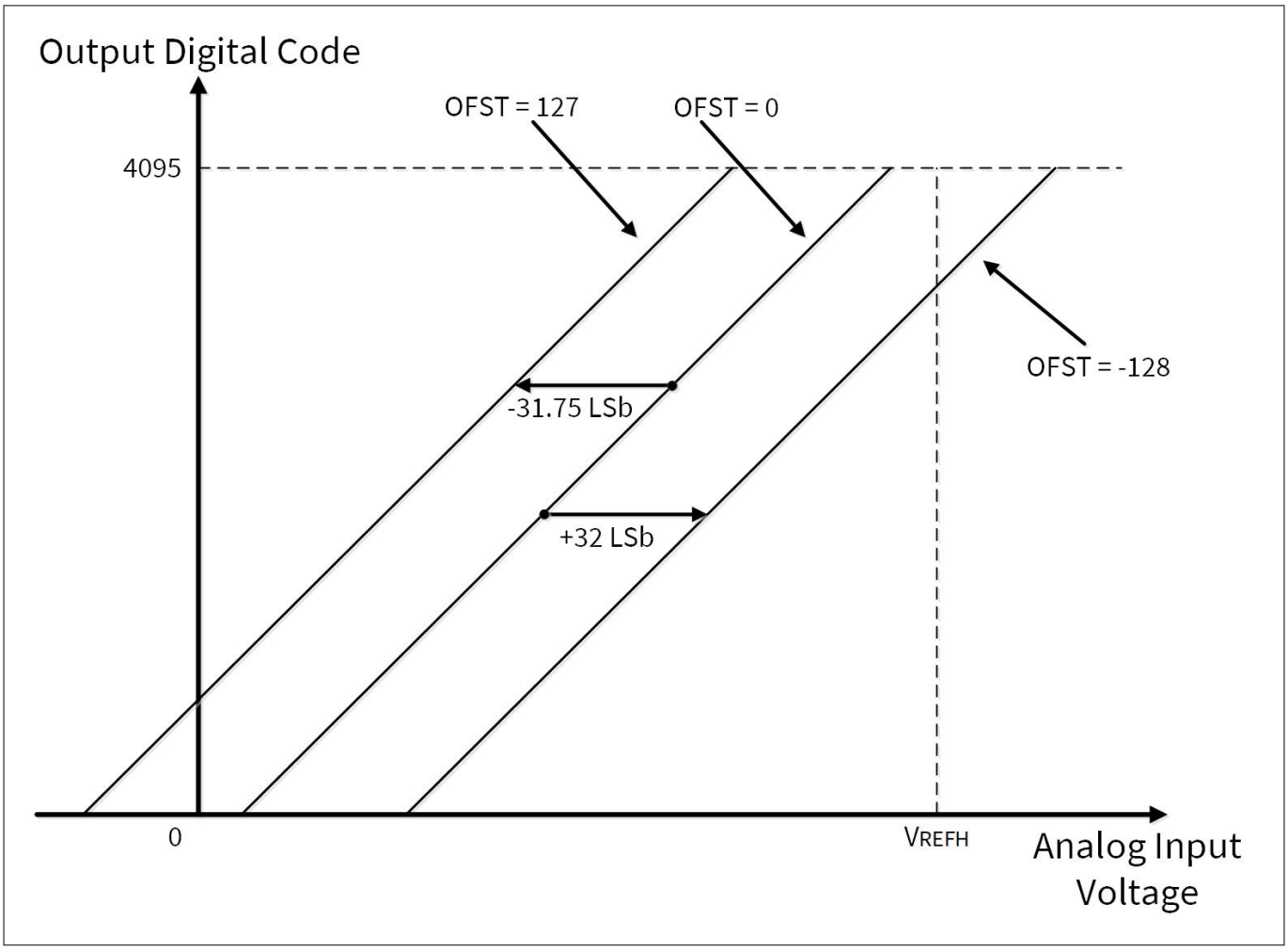


図 20 OFST とオフセット変化の関係

9.2 ゲイン調整の説明

ADC はゲイン誤差を補償するためのゲイン調整レジスタを搭載します。
SARn_ANA_CAL[20:16]がそのレジスタです。

9 キャリブレーション機能

SARn_ANA_CAL[20:16]レジスタには+15 から-15 までの値を設定できます。
ゲインは 2 分の 1LSB 単位で調整できます。
出力デジタル値と、SARn_ANA_CAL[20:16]値の関係式は以下です。(GAIN = SARn_ANA_CAL[20:16]の値)

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

図 21 に、GAIN とゲイン変化の関係を示します。

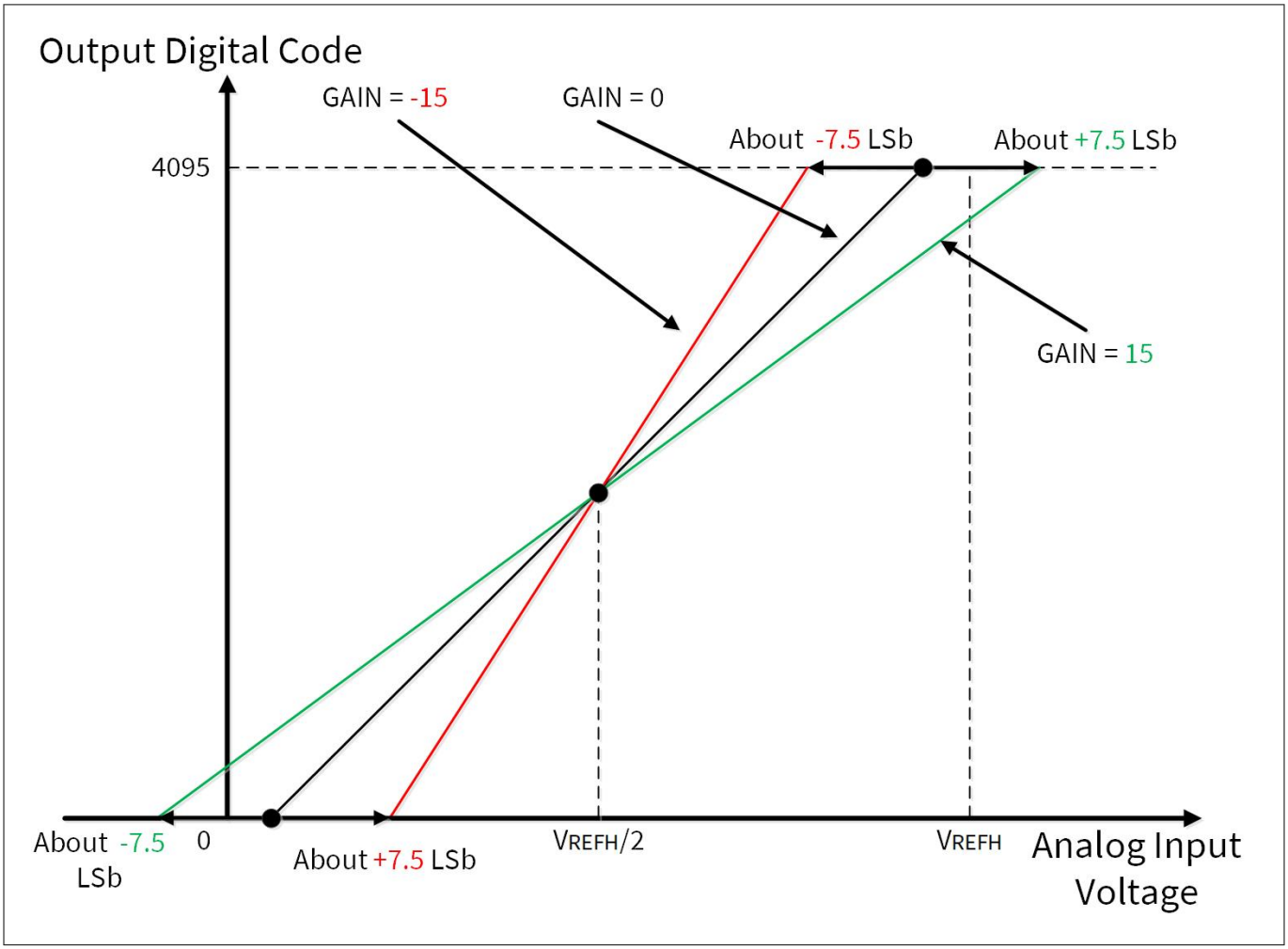


図 21　　GAIN とゲイン変化の関係

9.3　　キャリブレーション処理

図 22 に、ADC のキャリブレーションのフローチャート例を示します。最初にオフセット調整を行い、次にゲイン調整を行います。温度センサチャネルが有効の場合は、リファレンスバッファモードも有効にする必要があります。

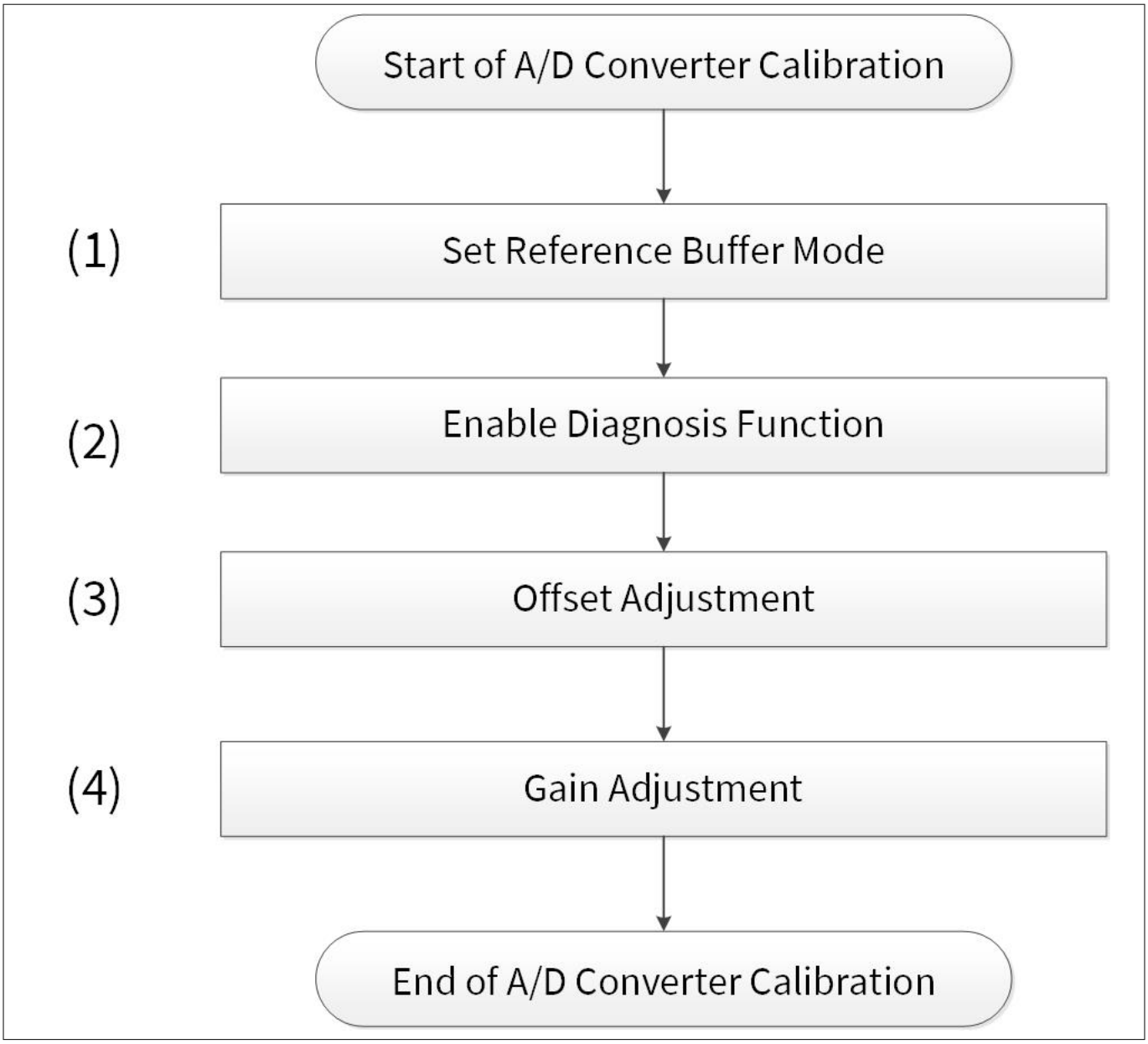


図 22　　ADC キャリブレーションのフローチャート例

9.3.1　　コンフィグレーション

ADC キャリブレーション手順設定の SDL の設定部のパラメータを表 27 に示します。

表 27　　ADC キャリブレーション手順設定パラメータ

パラメータ	説明	値
.offset	オフセットキャリブレーション設定	127ul
.gain	ゲインキャリブレーション設定	0ul
CY_ADC_REF_BUF_MODE_ON	リファレンスバッファモードの選択	1ul (表 25 参照)
BB_POTI_ANALOG_MACRO	TRAVEO™ T2G ベースボード上のポテンショメータ用アナログマクロ	CY_ADC_POT_MACRO: PASS0_SAR0

表 28 ADC キャリブレーション手順設定関数

関数	説明	値
`Cy_Adc_SetReferenceBufferMode(PASS0_EPASS_MMIO,RefBufMode)`	ePASS MMIO リファレンスバッファモード設定	RefBufMode = CY_ADC_REF_BUF_MODE_ON
`Cy_Adc_Diag_Enable(AnalogMacro)`	診断機能有効	AnalogMacro = BB_POTI_ANALOG_MACRO

9.3.2 サンプルコード

ADC キャリブレーション手順設定の初期設定のサンプルコードについては、Code listing 33 を参照してください。

Code listing 33 ADC キャリブレーション手順設定

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

    SystemInit();

:
    /* Set ePASS MMIO reference buffer mode */  /* (1) Set REF_BUF MODE. See Code Listing 30. */
    Cy_Adc_SetReferenceBufferMode(PASS0_EPASS_MMIO,CY_ADC_REF_BUF_MODE_ON);
    /* Enable diag function. */
    Cy_Adc_Diag_Enable(BB_POTI_ANALOG_MACRO);  /* (2) Enable ADC Diagnosis. See Code Listing
31. */
:
    if(Offset_Calibration() == true)  /* (3) Offset Adjustment. */
    {
        //Gain_Status = Gain_Calibration();
        if(Gain_Calibration() == true)  /* (4) Gain Adjustment. */
        {
            /* Test Completed */
            while(1);
        }
        else
        {
            /* Error */
            while(1);
        }
    }
    else
    {
        /* Error */
        while(1);
    }
}
```

9.4 オフセット調整手順

図 23 に、オフセット調整のフローチャート例を示します。

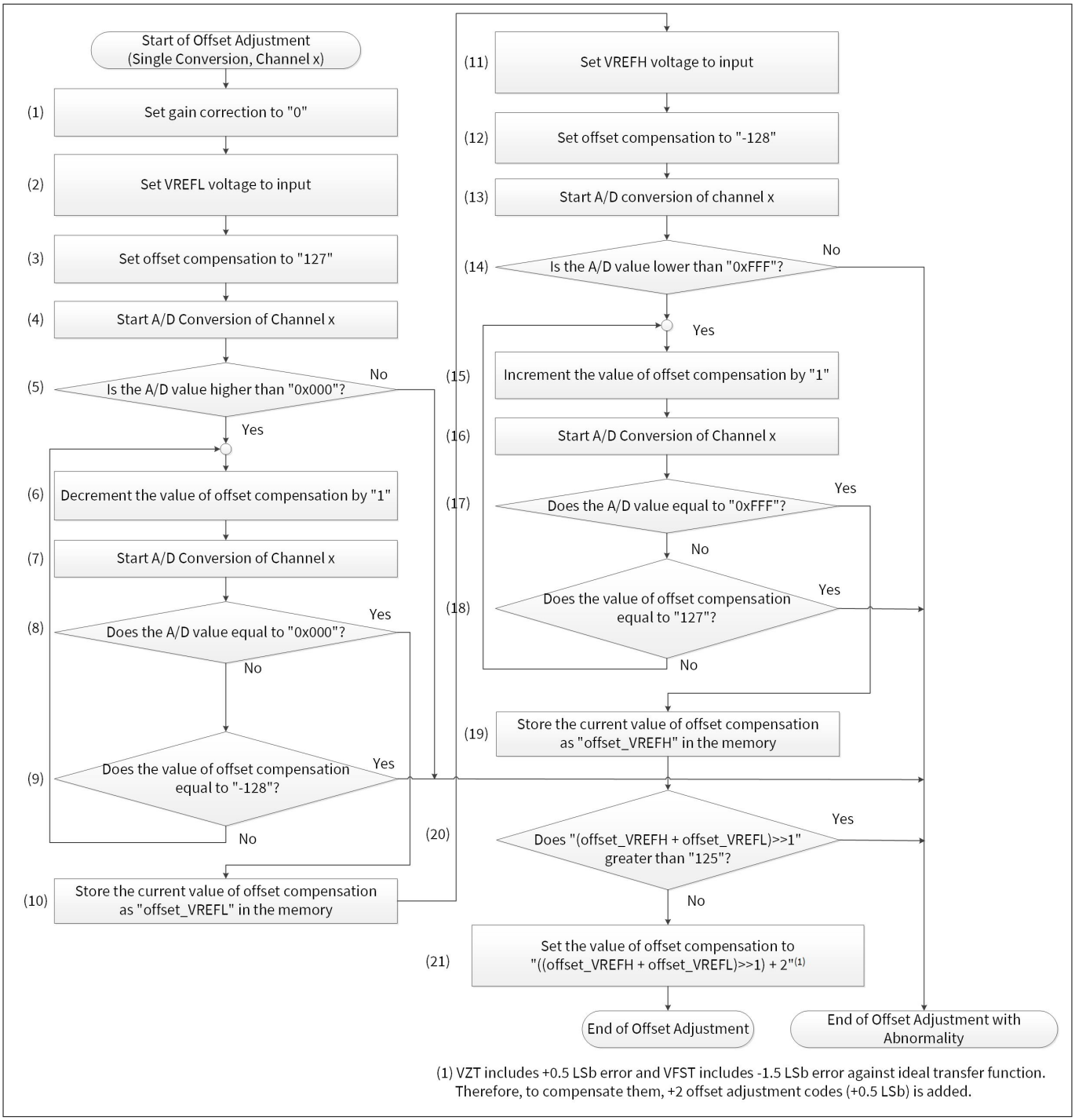


図 23　　オフセット調整のフローチャート例

9.4.1　ユースケース

以下のユースケースは、ADC ロジカルチャネル 0 のオフセット調整の例です。

9.4.2　コンフィグレーション

オフセット調整手順設定の SDL の設定部のパラメータを表 29 に、関数を表 30 示します。

表 29 オフセット調整手順設定パラメータ

パラメータ	説明	値
BB_POTI_ANALOG_MACRO	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログマクロ	CY_ADC_POT_MACRO: PASS0_SAR0
CY_ADC_REF_BUF_MODE_ON	リファレンスバッファモードの選択	1ul (表 25 参照)
calibrationConfig.offset	キャリブレーションオフセット値	-128 以上、127 以下
resultBuff	変換結果バッファ	- (計算値)

表 30 オフセット調整手順設定関数

関数	説明	値
Cy_Adc_Channel_SoftwareTrigger(PASS SARchannel)	ソフトウェア開始トリガ発行	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]
Cy_Adc_Channel_GetGroupStatus(PASS SARchannel, GroupStatus)	グループ変換ステータス取得	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]
Cy_Adc_Channel_GetResult(SAR Channel,Result,Status)	変換結果とステータスの取得	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Result = &resultBuff, Status = &statusBuff
Cy_Adc_SetAnalogCalibrationValue(PASS SARchannel,&calibrationConfig)	アナログキャリブレーション値の設定	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] Calibration Configure = calibrationConfig
Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)	ADC チャネルの初期化	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig
Cy_Adc_SetReferenceBufferMode(PASS0_EPASS_MMIO, RefBufMode)	ePASS MMIO リファレンスバッファモード設定	RefBufMode = CY_ADC_REF_BUF_MODE_ON
Cy_Adc_Diag_Enable(AnalogMacro)	診断機能有効	AnalogMacro = BB_POTI_ANALOG_MACRO
Cy_Adc_Channel_Enable(PASS SARchannel)	対応チャネル有効	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

9.4.3 サンプルコード

オフセット調整手順設定の初期設定のサンプルコードについては、Code listing 34 ～ Code listing 35 を参照してください。

9 キャリブレーション機能

Code listing 34 オフセット調整手順設定

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

    SystemInit();

:
    /* Set Gain Coreection to 0" */
    calibrationConfig.gain = 0ul;  /* (1) Set Gain Correction to “0”. */
    /* Set VREFL Voltage to Input */
    adcChannelConfig.pinAddress = CY_ADC_PIN_ADDRESS_VREF_L;  /* (2) Set VREFL Voltage to
Input. */
    /* Set Offset Compensation to "127" */
    calibrationConfig.offset = 127ul;  /* (3) Set offset compensation to “127”. */
:
     if(Offset_Calibration() == true)  /* (4) Start A/D Conversion of Channel x. */
    {
        //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]);  /*
Software Trigger Setting. See Code Listing 9. */h

    /* wait ADC completion */
    Cy_SysTick_DelayInUs(10000);
    cy_stc_adc_group_status_t Groupstatus = { false };

    do{
    Cy_Adc_Channel_GetGroupStatus(&BB_POTI_ANALOG_MACRO-
>CH[ADC_LOGICAL_CHANNEL],&Groupstatus);  /* Returns A/D conversion status. See Code Listing 32.
*/
}while(Groupstatus.grpComplete == false);


    /* Get the result(s) */
    Cy_Adc_Channel_GetResult(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &resultBuff,
&statusBuff);  /* Read A/D conversion data. See Code Listing 11. */
    return resultBuff;
}
/*****************************************************************************/
/* 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 ? */  /* (5) Is the A/D value higher than “0x000” ? */
    if(result > 0)
    {
        /* Decrment the Value of Offset Compensation by "1" */  /* (6) Decrement the value of
offset compensation by “1”. */
        for(offset_VREFL = 127; offset_VREFL >= -128; offset_VREFL -= 1)
        {
            /* Set "offset_VREFL" to offset regist value */
            calibrationConfig.offset = offset_VREFL;
            Cy_Adc_SetAnalogCalibrationValue(BB_POTI_ANALOG_MACRO, &calibrationConfig);  /*
Analog Calibration Value Setting. See Code Listing 35. */


            /* Get AD Value */
            result = GetAdcValue();  /* (7) Start A/D Conversion of Channel x. */


            /* Does the A/D value Equal to 0 ? */
            if(result == 0)  /* (8) Does the A/D value equal to “0x000” ? */
            {
                break;  /* (10) Store the current value of offset compensation as
```

```
“offset_VREFL” in the memory. */
            }


            if(offset_VREFL == -128)  /* (9) Does the value of offset compensation equal to
“-128”? */
            {
                /* Error */
                return false;
            }
}
}
    else
    {
        /* result <= 0x000 */
        /* End of Offset Adjustment with Abnormality */
        return false;
    }


    /*  Setting for VREFH */
    /* Set VREFH Voltage to Input */
    /* /* (11) Set VREFH voltage to input. */
    adcChannelConfig.pinAddress = CY_ADC_PIN_ADDRESS_VREF_H;
    /* Set Offset Compensation to "-128" */
    calibrationConfig.offset = -128;  /* (12) Set offset compensation to “-128”. */
    Cy_Adc_SetAnalogCalibrationValue(BB_POTI_ANALOG_MACRO, &calibrationConfig);  /* Analog
Calibration Value Setting. See Code Listing 35. */




    Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &adcChannelConfig);  /*
ADC Channel Initialize. See Code Listing 5. */

    /* Enable ADC ch. */

    Cy_Adc_Channel_Enable(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);  /* Enable ADC
Channel. See Code Listing 8. */

    /* Start A/D Conversion and get the A/D value */

    result = GetAdcValue();  /* (13) Start A/D Conversion of Channel x. */

    /* Is the AD value lower than 0xFFF */




    if(result < 0xFFF)  /* (14) Is the A/D value lower than “0xFFF”? */
    {
        /* Increment the Value of Offset Compensation by "1" */
        for(offset_VREFH = -128; offset_VREFH <= 127; offset_VREFH += 1)  /* (15) Increment the
```

```
value of offset compensation by “1” */
        {


            /* Set "offset_VREFH" to offset regist value */
            calibrationConfig.offset = offset_VREFH;
            Cy_Adc_SetAnalogCalibrationValue(BB_POTI_ANALOG_MACRO, &calibrationConfig);  /*
Analog Calibration Value Setting. See Code Listing 35. */


            /* Get AD Value */
            result = GetAdcValue();  /* (16) Start A/D Conversion of Channel x. */



            /* Does the A/D value Equal to 4095 ? */
            if(result == 0xFFF)  /* (17) Does the A/D value equal to “0xFFF” ? */
            {
                break;  /* (19) Store the current value of offset compensation as
“offset_VREFH” in the memory. */
            }



            /* Does the value of offset compensation equal to "127" ? */
     if(offset_VREFH == 127)  /* (18) Does the value of offset compensation equal to “127”? */
            {
                /* End of Offset Adjustment with Abnormality */
                return false;
            }
        }
    }
    else
    {
        /* End of Offset Adjustment with Abnormality */
        return false;
    }

    /* Does "(offset_VREFH+offset_VREFL)>>1" greater than "125" ?*/
    int32_t Value = (offset_VREFH+offset_VREFL)>>1;
    if(Value > 125)  /* (20) Does “(offset_VREFH + offset_VREFL)>>1” greater than “125”? */
    {


        /* End of Offset Adjustment with Abnormality */
        return false;
    }

    /* Set the value of offset compensation to ((offset_VREFH+offset_VREFL)>>1) + 2 */
    calibrationConfig.offset = ((offset_VREFH+offset_VREFL)>>1) + 2;  /* (21) Set the value of
offset compensation to “((offset_VREFH + offset_VREFL)>>1) + 2” */

    return true;
```

```
}
```

Code listing 35 Cy_Adc_SetAnalogCalibrationValue() 機能

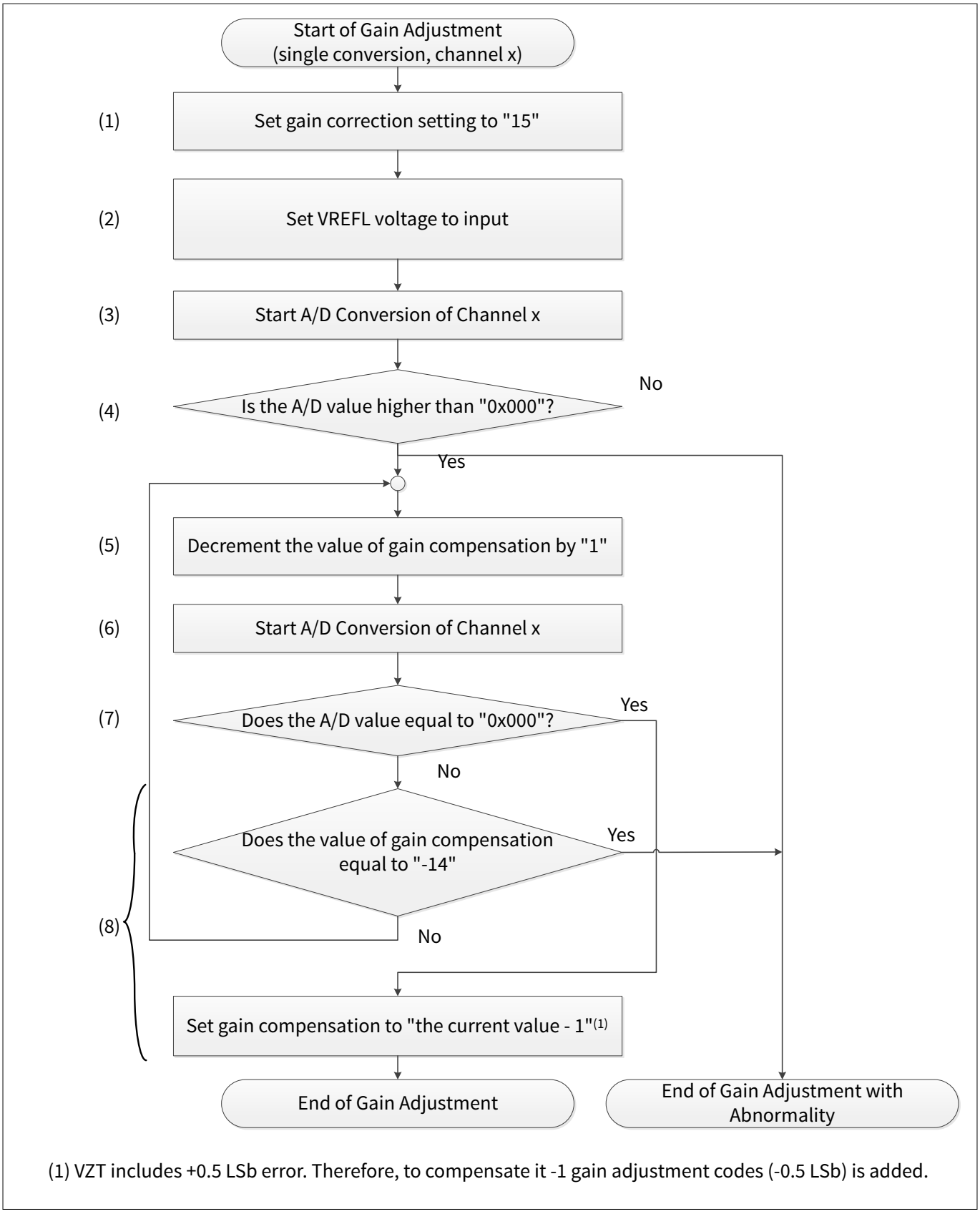
```
cy_en_adc_status_t Cy_Adc_SetAnalogCalibrationValue(volatile stc_PASS_SAR_t * base,
cy_stc_adc_analog_calibration_conifg_t * config)
{
    cy_en_adc_status_t    ret      = CY_ADC_SUCCESS;
    un_PASS_SAR_ANA_CAL_t unAnaCal = { 0ul };

    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 ゲイン調整手順

図 24 に、ゲイン調整のフローチャート例を示します。

9 キャリブレーション機能



(1) VZT includes +0.5 LSb error. Therefore, to compensate it -1 gain adjustment codes (-0.5 LSb) is added.

図 24　　ゲイン調整のフローチャート例

9.5.1 ユースケース

以下のユースケースは、ADC ロジカルチャネル 0 のゲイン調整の例です。

9.5.2 コンフィグレーション

A/D 変換グローバル設定の SDL の設定部のパラメータを表 31 に、関数を表 32 に示します。

表 31 ゲイン調整手順設定パラメータ

パラメータ	説明	値
BB_POTI_ANALOG_MACRO	TRAVEO™ T2G ベースボード上のポテンショメータ用のアナログマクロ	CY_ADC_POT_MACRO: PASS0_SAR0
calibrationConfig.gain	キャリブレーションゲイン値	-15 以上、15 以下
resultBuff	変換結果バッファ	- (計算値)

表 32 ゲイン調整手順設定関数

関数	説明	値
Cy_Adc_Channel_SoftwareTrigger(PASS SARchannel)	ソフトウェア開始トリガ発行	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]
Cy_Adc_Channel_GetGroupStatus(PASS SARchannel, GroupStatus)	グループ変換ステータス取得	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]
Cy_Adc_Channel_GetResult(SAR Channel,Result,Status)	変換結果とステータスの取得	SAR Channel = BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], Result = &resultBuff, Status = &statusBuff
Cy_Adc_SetAnalogCalibrationValue(PASS SARchannel,&calibrationConfig)	アナログキャリブレーション値の設定	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] Calibration Configure = calibrationConfig
Cy_Adc_Channel_Init(PASS SARchannel,ADCchannel Configure)	ADC チャネルの初期化	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL] ADCchannel Configure = adcChannelConfig
Cy_Adc_Channel_Enable(PASS SARchannel)	対応チャネル有効	PASS SARchannel = &BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]

9.5.3 サンプルコード

ゲイン調整手順設定の初期設定のサンプルコードについては、Code listing 36 を参照してください。

Code listing 36 ゲイン調整手順設定

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

    SystemInit();

:
    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]);  /*
Software Trigger Setting. See Code Listing 9. */


    /* wait ADC completion */
```

```
    Cy_SysTick_DelayInUs(10000);
    cy_stc_adc_group_status_t Groupstatus = { false };

    do{
    Cy_Adc_Channel_GetGroupStatus(&BB_POTI_ANALOG_MACRO-
>CH[ADC_LOGICAL_CHANNEL],&Groupstatus);  /* Returns A/D conversion status. See Code Listing 32.
*/
}while(Groupstatus.grpComplete == false);
    /* Get the result(s) */
    Cy_Adc_Channel_GetResult(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &resultBuff,
&statusBuff);  /* Read A/D conversion data. See Code Listing 11. */

    return resultBuff;
}

/*******************************************************************/
/* Function: Gain_Calibration
*/
/*******************************************************************/




bool Gain_Calibration(void)
{
    int16_t Gain;
    /* Set Gain Correction setting to "15" */  /* (1) Set Gain Correction to “15”. */
    calibrationConfig.gain = 15;
    /* Set VREFL Voltage to Input */
    adcChannelConfig.pinAddress = CY_ADC_PIN_ADDRESS_VREF_L;  /* (2) Set VREFL Voltage to
Input. */
    Cy_Adc_SetAnalogCalibrationValue(BB_POTI_ANALOG_MACRO, &calibrationConfig);  /* Analog
Calibration Value Setting. See Code Listing 35. */
    Cy_Adc_Channel_Init(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL], &adcChannelConfig);  /*
Initialize ADC Channel. See Code Listing 5. */



    /* Enable ADC ch. */
    Cy_Adc_Channel_Enable(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL]);  /* Enable ADC
Channel. See Code Listing 8 */
    /* Start A/D value Conversion */
    result = GetAdcValue();  /* (3) Start A/D Conversion of Channel x. */



    if(result > 0)  /* (4) Is the A/D value higher than “0x000”? */
    {
         for(Gain = 15; Gain >=-14; Gain -= 1)
         {
```

```
            /* Set gain to register */
            calibrationConfig.gain = Gain;  /* Analog Calibration Value Setting. See Code
Listing 35. */
            Cy_Adc_SetAnalogCalibrationValue(BB_POTI_ANALOG_MACRO, &calibrationConfig);



            result = GetAdcValue();  /* (5) Start A/D Conversion of Channel x. */


            /* Does the A/D value equal to "0" ?*/
            if(result == 0)  /* (6) Does the A/D value equal to “0x000”?  */
            {

                /* Set gain compension to "gain" -1 */
                calibrationConfig.gain = Gain - 1;  /* (7) Set gain compensation to “the
current value - 1". */
                Cy_Adc_SetAnalogCalibrationValue(BB_POTI_ANALOG_MACRO, &calibrationConfig);  /*
Analog Calibration Value Setting. See Code Listing 35. */

                break;
            }



            if( Gain == -14)  /* (8) Does the value of gain compensation equal to “-14” */
            {
            /* End of Offset Adjustment with Abnormality */
                return false;
            }
    }
    return true;
}
```

10 温度センサ

温度センサは ADC を使用してチップ温度を測定できます。動作時の温度を正しく測定するためには、製造時の参照測定を使用します。この参照データは、他のキャリブレーションデータと共に SFlash に保存されています。このデータを読み出すための詳細アドレスは、アーキテクチャ TRM の「SAR ADC」章を参照してください。

ここでは、チップ温度測定フロー例を示します。

基本的な ADC グローバル設定に記載されている設定がされていることを確認してください。

10.1 温度測定手順

図 25 に、チップ温度測定のフローチャート例を示します。

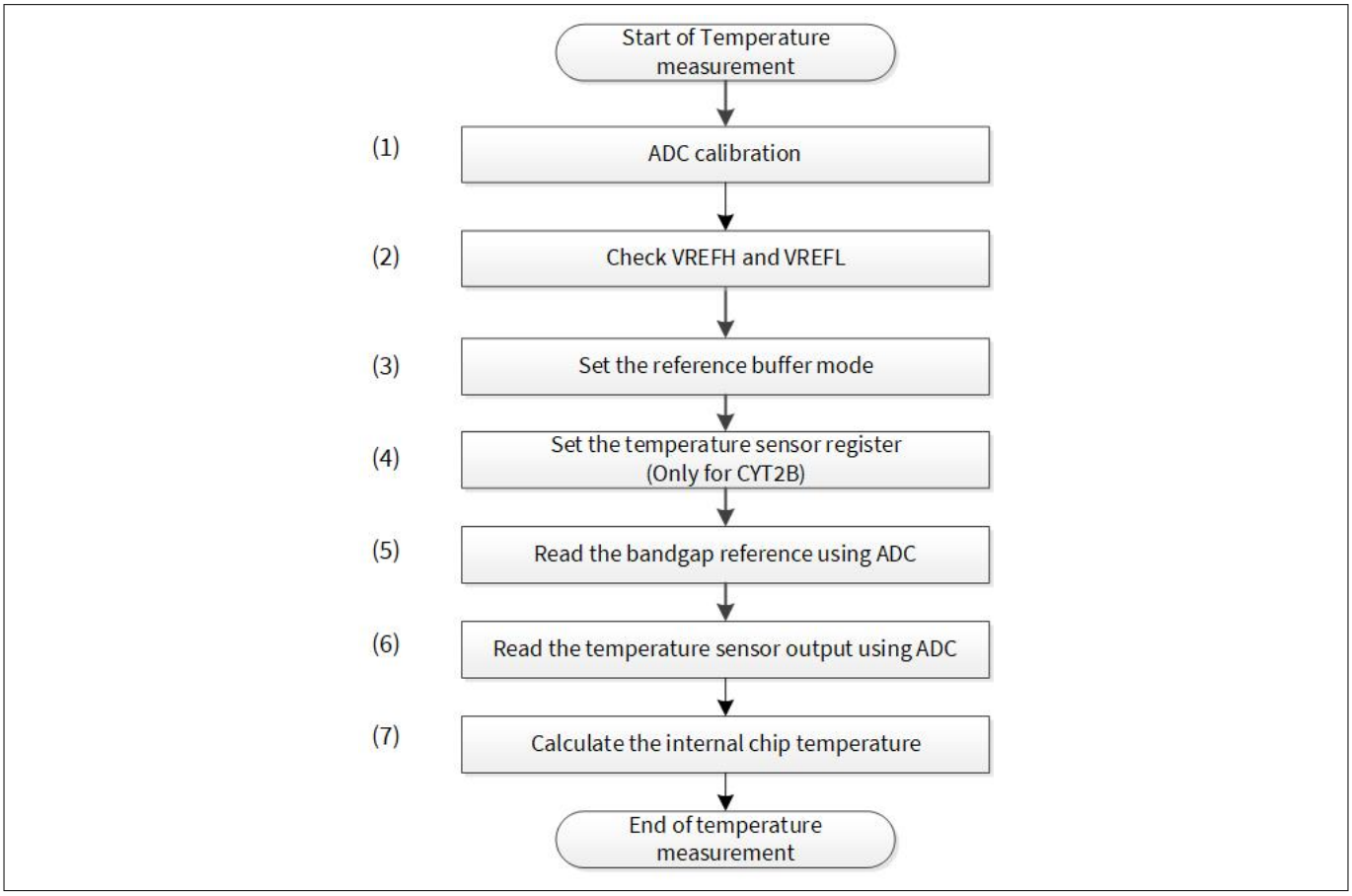


図 25 温度測定のフローチャート例

1. ADC キャリブレーション: オフセットとゲイン調整の例は 9.3 を参照してください。
2. VREFH と VREL の確認: VREFH と VREFL の診断例は 8.2 を参照してください。
3. リファレンスバッファモード設定: リファレンスバッファモードを有効するために PASS_CTL レジスタを設定します。
4. 温度センサレジスタの設定: CYT2B の場合のみ PASS_TEST_CTL レジスタのビット 9, 8, 6 に 1 を設定します。他のデバイスの場合は、デフォルト値を維持します。
5. ADC を使用したバンドギャップリファレンス (VBG) の読み出し: VBG の A/D 変換結果を保存します。
6. ADC を使用した温度センサ出力 (VBE) の読み出し: VBE の A/D 変換結果を保存します。温度センサに最適なサンプル時間を検討するためには、[1.]のデータシートを参照してください。
7. チップ内部温度の計算:
   a. VBE (温度センサ出力)は 、温度(T)の 2 次関数であり、以下の式で表せます。

$$V_{BE} = aT^2 + bT + c$$

VBE から温度を計算するために、上記の式の係数 (a, b, c) を求める必要があります。これらの係数は、SFlash に保存されているデータ (3 つの異なる温度で測定された VBE) を使用して計算できます。詳細は、アーキテクチャ TRM を参照してください。使用する (VBE, T) の 3 つの組合せは、電源電圧 (VDDA) によって異なります。例えば、多項式係数を計算するためには、

VDDA = 3.3 V の場合、アーキテクチャ TRM の SFlash データ set#0 と set#1 を使用してください。

VDDA = 5.0 V の場合、アーキテクチャ TRM の SFlash データ set#0 と set#2 を使用してください。

b. ADC 基準電圧はキャリブレーションから変更されている可能性があるため、温度計算に使用する前に、ADC (VBG_S3) と VBG を使用して、室温でのバンドギャップ電圧の比率を使用して VBE をスケーリングする必要があります。

c. 上記の多項式を使用して温度を計算してください。

$$V_{BE\_NEW} = aT^2 + bT + c$$

d. 温度を計算した後、ステップ.で最も近い温度からのバンドギャップリファレンスを使用することで精度を向上させることができます。基本的に、ステップ . で計算された温度が次の値に近い場合:

1. COLD (-40) の場合、ADC (VBG_S2) を使用して COLD 温度でのバンドギャップ電圧でステップ.を繰り返し、ステップ.(7.3) を使用して温度を再計算してください。
2. HOT (150) の場合、ADC (VBG_CHI) を使用して HOT 温度による高温でのバンドギャップ電圧でステップ.を繰り返し、ステップ.を使用して温度を再計算してください。
3. ROOM (27) の場合、ステップ.で計算された温度が最終温度です。

10.2 ユースケース

- VDDDA = 5.0 V
- ADC ロジカルチャネル: 0
- ADC 測定値を取得するための反復回数: 16
- サンプリング時間: 120 ADC クロックサイクル

10.3 コンフィグレーション

温度測定の SDL の設定部のパラメータを表 33 に、関数を表 34 に示します。

表 33 温度測定パラメータ

パラメータ	説明	値
USE_TEMPERATURE_TRIM_VALUES	アナログ電源の選択 3.3 V: USE_TRIM_FOR_3P3V_VDDA 5.0 V: USE_TRIM_FOR_5P0V_VDDA	USE_TRIM_FOR_5P0V_VDDA
ADC_LOGICAL_CHANNEL	使用する ADC ロジカルチャネル	0ul
ADC_CH_NUM_OF_ITERATION	ADC 測定値を取得するための反復回数	16ul

(続く)

表 33 (続き) 温度測定パラメータ

パラメータ	説明	値
ADC_CH_SAMPLE_TIME	ADC クロックサイクルのサンプル時間	120ul
adcChannelConfig.triggerSelection	トリガ OFF	0ul (表 3 参照)
adcChannelConfig.channelPriority	チャネル優先度	0ul
adcChannelConfig.preenptionType	Pre-emption タイプ	3ul (表 3 参照)
adcChannelConfig.isGroupEnd	グループ終了？	True
adcChannelConfig.doneLevel	Done レベル	0ul (表 3 参照)
adcChannelConfig.pinAddress	ピンアドレス	channelAddress
adcChannelConfig.portAddress	ポートアドレス	0ul (表 3 参照)
adcChannelConfig.extMuxSelect	外部 MUX 選択	0ul
adcChannelConfig.extMuxEnable	外部 MUX 有効	True
adcChannelConfig.preconditionMode	プリコンディションモード	0ul (表 3 参照)
adcChannelConfig.overlapDiagMode	オーバラップ診断モード	0ul (表 3 参照)
adcChannelConfig.sampleTime	サンプル時間	ADC_CH_SAMPLE_TIME
adcChannelConfig.calibrationValueSelect	キャリブレーション値選択	1ul (表 3 参照)
adcChannelConfig.postProcessingMode	ポストプロセッシングモード	1ul (表 3 参照)
adcChannelConfig.resultAlignment	結果の配置	0ul (表 3 参照)
adcChannelConfig.signExtention	符号拡張	0ul (表 3 参照)
adcChannelConfig.averageCount	平均カウント	(ADC_CH_NUM_OF_ITERATION-1)
adcChannelConfig.rightShift	右シフト	0ul
adcChannelConfig.rangeDetectionMode	レンジ検出モード	1ul (表 3 参照)
adcChannelConfig.rangeDetectionLoThreshold	レンジ検出閾値 (下限)	0x0000ul
adcChannelConfig.rangeDetectionHiThreshold	レンジ検出閾値 (上限)	0x0FFFul
adcChannelConfig.mask.grpDone	マスクグループ Done	True
adcChannelConfig.mask.grpCancelled	マスクグループキャンセル	False
adcChannelConfig.mask.grpOverflow	マスクグループオーバフロー	False
adcChannelConfig.mask.chRange	マスクチャネルレンジ	False
adcChannelConfig.mask.chPulse	マスクチャネルパルス	False
adcChannelConfig.mask.chOverflow	マスクチャネルオーバフロー	False

表 34 温度測定関数

関数	説明	値
`AdcReadChannelData(cy_en_adc_pin_address_t channelAddress)`	対象ピンアドレスの ADC 変換結果の読み出し	CY_ADC_IN_VBG (Y_ADC_PIN_ADDRESS_VBG = 38ul), CY_ADC_IN_TEMP (CY_ADC_PIN_ADDRESS_VTEMP = 39ul)
`AdcConfigureChannel(cy_en_adc_pin_address_t channelAddress)`	ADC チャネルの初期化と有効化	
`Cy_Adc_CalculateDieTemperature(cy_stc_adc_temp_ref_t *refValue, cy_stc_adc_temp_raw_t *rawValue)`	チップ温度の計算	&tempRefValue, &tempRawValue

10.4 サンプルコード

温度測定のサンプルコードについては、Code listing 37～Code listing 41 を参照してください。

10 温度センサ

Code listing 37 温度測定設定

```
/** Select the analaog power rails as per your hardware */
#define USE_TRIM_FOR_3P3V_VDDA                  0ul
#define USE_TRIM_FOR_5P0V_VDDA                  1ul
#define USE_TEMPERATURE_TRIM_VALUES             USE_TRIM_FOR_5P0V_VDDA

/** Read from the SFLASH table */
#if (USE_TEMPERATURE_TRIM_VALUES == USE_TRIM_FOR_3P3V_VDDA)
    /** SFLASH table for VDDA - 3.3V */
    #define EPASS_TEMP_TRIM_TEMP_COLDSORT       0x17000654ul
    #define EPASS_TEMP_TRIM_TEMP_ROOMSORT       0x1700064Eul
    #define EPASS_TEMP_TRIM_TEMP_HOTCLASS       0x1700065Aul
    #define EPASS_TEMP_TRIM_DIODE_COLDSORT      0x17000656ul
    #define EPASS_TEMP_TRIM_DIODE_ROOMSORT      0x17000650ul
    #define EPASS_TEMP_TRIM_DIODE_HOTCLASS      0x1700065Cul
    #define EPASS_TEMP_TRIM_VBG_COLDSORT        0x17000658ul
    #define EPASS_TEMP_TRIM_VBG_ROOMSORT        0x17000652ul
    #define EPASS_TEMP_TRIM_VBG_HOTCLASS        0x1700065Eul
#else
/** SFLASH table for VDDA - 5V */
    #define EPASS_TEMP_TRIM_TEMP_COLDSORT       0x17000654ul
    #define EPASS_TEMP_TRIM_TEMP_ROOMSORT       0x1700064Eul
    #define EPASS_TEMP_TRIM_TEMP_HOTCLASS       0x1700065Aul
    #define EPASS_TEMP_TRIM_DIODE_COLDSORT      0x1700066Eul
    #define EPASS_TEMP_TRIM_DIODE_ROOMSORT      0x1700066Aul
    #define EPASS_TEMP_TRIM_DIODE_HOTCLASS      0x17000672ul
    #define EPASS_TEMP_TRIM_VBG_COLDSORT        0x17000670ul
    #define EPASS_TEMP_TRIM_VBG_ROOMSORT        0x1700066Cul
    #define EPASS_TEMP_TRIM_VBG_HOTCLASS        0x17000674ul
#endif  /* (USE_TEMPERATURE_TRIM_VALUES == USE_TRIM_FOR_3P3V_VDDA) */

/** SFLASH table read macro */
#define GET_SFLASH_VALUE(x)                     CY_GET_REG16(x)
#define GET_TRIM_VALUE(x)                       (GET_SFLASH_VALUE(x) & 0xFFFFu)

/** Temperature trim values read from SFLASH */
#define TEMP_TRIM_TEMP_ROOMSORT                 GET_TRIM_VALUE(EPASS_TEMP_TRIM_TEMP_ROOMSORT)
#define TEMP_TRIM_TEMP_COLDSORT                 GET_TRIM_VALUE(EPASS_TEMP_TRIM_TEMP_COLDSORT)
#define TEMP_TRIM_TEMP_HOTCLASS                 GET_TRIM_VALUE(EPASS_TEMP_TRIM_TEMP_HOTCLASS)

#define TEMP_TRIM_DIODE_ROOMSORT                GET_TRIM_VALUE(EPASS_TEMP_TRIM_DIODE_ROOMSORT)
#define TEMP_TRIM_DIODE_COLDSORT                GET_TRIM_VALUE(EPASS_TEMP_TRIM_DIODE_COLDSORT)
#define TEMP_TRIM_DIODE_HOTCLASS                GET_TRIM_VALUE(EPASS_TEMP_TRIM_DIODE_HOTCLASS)

#define TEMP_TRIM_VBG_ROOMSORT                  GET_TRIM_VALUE(EPASS_TEMP_TRIM_VBG_ROOMSORT)
#define TEMP_TRIM_VBG_COLDSORT                  GET_TRIM_VALUE(EPASS_TEMP_TRIM_VBG_COLDSORT)
#define TEMP_TRIM_VBG_HOTCLASS                  GET_TRIM_VALUE(EPASS_TEMP_TRIM_VBG_HOTCLASS)

/**
 * Macro definition for ADC instance and Temperature measurement channels
 */
```

```
/** ADC instance, clock and irq configuration macro */
#define CY_ADC_MACRO                            PASS0_SAR0
#define CY_ADC_MMIO_MACRO                       PASS0_EPASS_MMIO
#define CY_ADC_PCLK                             PCLK_PASS0_CLOCK_SAR0
#define CY_ADC_IRQN                             pass_0_interrupts_sar_0_IRQn

/** ADC channels for diode and temperature measurements */
#define CY_ADC_IN_VBG                           CY_ADC_PIN_ADDRESS_VBG
#define CY_ADC_IN_TEMP                          CY_ADC_PIN_ADDRESS_VTEMP

/** ADC logical channel to be used */
#define ADC_LOGICAL_CHANNEL                     0ul

/** Number of iteration to get the ADC readings */
#define ADC_CH_NUM_OF_ITERATION                 16ul

/** Sample time for sampling ADC channel */
#define ADC_CH_SAMPLE_TIME                      120ul

/** Defines low and high range for range detection mode */
#define ADC_CH_RANGE_DETECT_LOW                 0x0000ul
#define ADC_CH_RANGE_DETECT_HIGH                0x0FFFul

/** Select the internal current requirement for the die-temperature sensor
    0->1uA, 2->2uA, 3->5uA, 3->10uA */
#define ADC_CAL_TS_CUR_SEL_VALUE                3ul
#define ADC_CAL_TS_CUR_SEL_OFFSET               8ul
#define ADC_CAL_TS_CUR_SEL_MASK             (ADC_CAL_TS_CUR_SEL_VALUE <<
ADC_CAL_TS_CUR_SEL_OFFSET)

/** Select current or voltage output from the die-temperature sensor
    0->current, 1->voltage */
#define ADC_CAL_TS_VI_SEL_VALUE                1ul
#define ADC_CAL_TS_VI_SEL_OFFSET               6ul
#define ADC_CAL_TS_VI_SEL_MASK                (ADC_CAL_TS_VI_SEL_VALUE <<
ADC_CAL_TS_VI_SEL_OFFSET)

/** Setup test calibration register to measure the die-temperature */
#define ADC_CAL_TEMP_TEST_CTL_ADDR_OFFSET       (0x80)
#define ADC_CAL_TEMP_TEST_CTL_ADDR  (uint32_t)CYREG_PASS0_PASS_CTL +
ADC_CAL_TEMP_TEST_CTL_ADDR_OFFSET)
#define ADC_CAL_TEMP_TEST_CTL_MASK              (ADC_CAL_TS_CUR_SEL_MASK |
ADC_CAL_TS_VI_SEL_MASK)


/*******************************************************************************/
/* Global Variables                                                            */
/*******************************************************************************/
:
/**
 * \var cy_stc_adc_channel_config_t adcChannelConfig
 * \brief ADC channel configuration structure
```

```
 */
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_L,
    .portAddress               = CY_ADC_PORT_ADDRESS_SARMUX0,
    .extMuxSelect              = 0ul,
    .extMuxEnable              = true,
    .preconditionMode          = CY_ADC_PRECONDITION_MODE_OFF,
    .overlapDiagMode           = CY_ADC_OVERLAP_DIAG_MODE_OFF,
    .sampleTime                = ADC_CH_SAMPLE_TIME,
    .calibrationValueSelect    = CY_ADC_CALIBRATION_VALUE_ALTERNATE,
    .postProcessingMode        = CY_ADC_POST_PROCESSING_MODE_AVG,
    .resultAlignment           = CY_ADC_RESULT_ALIGNMENT_RIGHT,
    .signExtention             = CY_ADC_SIGN_EXTENTION_UNSIGNED,
    .averageCount              = (ADC_CH_NUM_OF_ITERATION-1),
    .rightShift                = 0ul,
    .rangeDetectionMode        = CY_ADC_RANGE_DETECTION_MODE_INSIDE_RANGE,
    .rangeDetectionLoThreshold = ADC_CH_RANGE_DETECT_LOW,
    .rangeDetectionHiThreshold = ADC_CH_RANGE_DETECT_HIGH,
    .mask.grpDone              = true,
    .mask.grpCancelled         = false,
    .mask.grpOverflow          = false,
    .mask.chRange              = false,
    .mask.chPulse              = false,
    .mask.chOverflow           = false,
};

/**
 * \var static bool isConversionComplete
 * \brief ADC conversion complete flag */
static bool isConversionComplete = false;

/**
 * \var static uint16_t resultBuff
 * \brief ADC conversion result buffer place holder */
static uint16_t resultBuff[ADC_CH_NUM_OF_ITERATION];

/**
 * \var static cy_stc_adc_ch_status_t statusBuff
 * \brief ADC conversion status buffer place holder */
static cy_stc_adc_ch_status_t  statusBuff[ADC_CH_NUM_OF_ITERATION];

/**
 * \var static double temperatureData
 * \brief stores internal die-temperature value */
static double temperatureData;
```

10 温度センサ

Code listing 38 温度測定

```
int main(void)
{
    cy_stc_adc_temp_ref_t tempRefValue;
    cy_stc_adc_temp_raw_t tempRawValue;

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

    SystemInit();
:
    /* Update the sort temperature value matrix A */
    tempRefValue.adcTempValues.coldValue = -(TEMP_TRIM_TEMP_COLDSORT / 10.0);
    // Note: Temperature data read from sFLASH is multiple of 10.
    tempRefValue.adcTempValues.roomValue = (TEMP_TRIM_TEMP_ROOMSORT / 10.0);
    // Note: Temperature data read from sFLASH is multiple of 10.
    tempRefValue.adcTempValues.hotValue = (TEMP_TRIM_TEMP_HOTCLASS / 10.0);
    // Note: Temperature data read from sFLASH is multiple of 10.

    /* Update the sort temperature adc value matrix b */
    tempRefValue.adcDiodeValues.coldValue = (uint16_t)TEMP_TRIM_DIODE_COLDSORT;
    tempRefValue.adcDiodeValues.roomValue = (uint16_t)TEMP_TRIM_DIODE_ROOMSORT;
    tempRefValue.adcDiodeValues.hotValue = (uint16_t)TEMP_TRIM_DIODE_HOTCLASS;

    /* Update the reference Vbg values */
    tempRefValue.adcVbgValues.coldValue = (uint16_t)TEMP_TRIM_VBG_COLDSORT;
    tempRefValue.adcVbgValues.roomValue = (uint16_t)TEMP_TRIM_VBG_ROOMSORT;
    tempRefValue.adcVbgValues.hotValue = (uint16_t)TEMP_TRIM_VBG_HOTCLASS;

    /* This block needs to be called every time when reading the die temperature */
    {
        /* SoftTrim API */
        AdcCalculateSoftTrim(CY_ADC_MACRO);  /* (1) ADC calibration. See Code Listing 33. */

        /* Test with the updatded calibration values */
        AdcCheckSoftTrim();  /* (2) Check VREFH and VREFL. See Code Listing 29. */

        /* Set reference buffered mode on - to pump Vbg from SRSS */
        Cy_Adc_SetReferenceBufferMode(CY_ADC_MMIO_MACRO, CY_ADC_REF_BUF_MODE_ON);  /* (3) Set
the reference buffer mode. See Code Listing 30. */



        /* Set current configuration for temperature sensor to 10uA for better accuracy */
        CY_SET_REG32(ADC_CAL_TEMP_TEST_CTL_ADDR, ADC_CAL_TEMP_TEST_CTL_MASK);  /* (4) Set the
temperature sensor register only for CYT2B. */



        /* Read and update the raw values for VBG and Temp Sensor */
        tempRawValue.adcVbgRawValue = AdcReadChannelData(CY_ADC_IN_VBG);  /* (5) Read the
```

```
bandgap reference using ADC. See Code Listing 39. */



        tempRawValue.adcVtempRawValue = AdcReadChannelData(CY_ADC_IN_TEMP);  /* (6) Read the
temperature sensor output using ADC. See Code Listing 39 */



        /* Calculate the internal die temperature */
        temperatureData = Cy_Adc_CalculateDieTemperature(&tempRefValue, &tempRawValue);  /* (7)
Calculate the internal chip temperature. See Code Listing 41. */
    }

    for(;;)
    {
        if(temperatureData == 0.0)
        {
            /* Reaching here means SFLASH doesn't have valid values */
            CY_ASSERT(false);
        }
    }
}
```

Code listing 39 AdcReadChannelData() 機能

```
static uint16_t AdcReadChannelData(cy_en_adc_pin_address_t channelAddress)
{
    unsigned long avgAdcValue = 0ul;

    AdcConfigureChannel(channelAddress);  /* See Code Listing 40. */
    {
        Cy_Adc_Channel_SoftwareTrigger(&CY_ADC_MACRO->CH[ADC_LOGICAL_CHANNEL]);  /* See Code
Listing 9. */
        while(isConversionComplete != true);
        isConversionComplete = false;
        avgAdcValue += resultBuff[0];

    }
    return (avgAdcValue/ADC_CH_NUM_OF_ITERATION);
}
```

Code listing 40 AdcConfigureChannel() 機能

```
static void AdcConfigureChannel(cy_en_adc_pin_address_t channelAddress)
{
    /* Configure the ADC channel to requested address */
    adcChannelConfig.pinAddress = channelAddress;  /* ADC Channel Initialize. See Code Listing
5. */
    Cy_Adc_Channel_Init(&CY_ADC_MACRO->CH[ADC_LOGICAL_CHANNEL], &adcChannelConfig);



    /* Enable ADC ch. */  /* Enable ADC Channel. See Code Listing 8. */
    Cy_Adc_Channel_Enable(&CY_ADC_MACRO->CH[ADC_LOGICAL_CHANNEL]);
}
```

Code listing 41 Cy_Adc_CalculateDieTemperature() 機能

```
double Cy_Adc_CalculateDieTemperature(cy_stc_adc_temp_ref_t *refValue, cy_stc_adc_temp_raw_t
*rawValue)
{
    double determinant = 0.0;
    double coefficientMatrixForSort[3][3] = {{0.0},{0.0},{0.0}};

    double sortTempMatrixA[3][3];
    double sortTempValueMatrixB[3][1];
    double sortTempCoefficeintsX[3] = {0.0};

    double coeffA, coeffB, coeffC;
    double bSqrMin4ac = 0.0;
    double tempRefValue = 0.0;
    double tempScaleValue = 0.0;
    double correctionFactor = 0.0;
    double tempRootPos, tempRootNeg = 0.0;
    double tempRangeOffset = 10; /* relate to +/- 10 degC */
    double tempInDegreeC = 0.0;

    /*****************************************************************************
     * Update the class and sort matrix from the sFLASH values
     *****************************************************************************/

    /* Update the sort temperature value matrix A */
    sortTempMatrixA[0][0] = 1.0;
    sortTempMatrixA[0][1] = refValue->adcTempValues.coldValue;
    sortTempMatrixA[0][2] = pow(refValue->adcTempValues.coldValue, 2.0);
    sortTempMatrixA[1][0] = 1.0;
    sortTempMatrixA[1][1] = refValue->adcTempValues.roomValue;
    sortTempMatrixA[1][2] = pow(refValue->adcTempValues.roomValue, 2.0);
    sortTempMatrixA[2][0] = 1.0;
    sortTempMatrixA[2][1] = refValue->adcTempValues.hotValue;
    sortTempMatrixA[2][2] = pow(refValue->adcTempValues.hotValue, 2.0);

    /* Update the sort temperature adc value matrix B */
    sortTempValueMatrixB[0][0] = refValue->adcDiodeValues.coldValue;
    sortTempValueMatrixB[1][0] = refValue->adcDiodeValues.roomValue;
    sortTempValueMatrixB[2][0] = refValue->adcDiodeValues.hotValue;

    /*****************************************************************************
     * Get the 2nd order coefficient for sort value matrix
      /* (7.1) Get the second order coefficients (a, b, c) */
     *****************************************************************************/

    /* Get the determinant of sort temperature matrix A */
    for(uint8_t i = 0u; i < 3u; i++)
    {
        determinant = determinant + (sortTempMatrixA[0][i]*(sortTempMatrixA[1]
[(i+1)%3]*sortTempMatrixA[2][(i+2)%3] - sortTempMatrixA[1][(i+2)%3]*sortTempMatrixA[2]
[(i+1)%3]));
```

```
    }

    /* Get the inverse of sort temperature matrix A */
    for(uint8_t i = 0u; i < 3u; i++)
    {
        for(uint8_t j = 0u; j < 3u; j++)
        {
            coefficientMatrixForSort[i][j] = ((sortTempMatrixA[(i+1)%3][(j+1)%3] *
sortTempMatrixA[(i+2)%3][(j+2)%3]) - (sortTempMatrixA[(i+1)%3][(j+2)%3]*sortTempMatrixA[(i+2)%3]
[(j+1)%3]))/ determinant;
        }
    }
    for(uint8_t i = 0u; i < 3u; i++)
    {
        for(uint8_t j = 0u; j < 3u; j++)
        {
            sortTempMatrixA[i][j] = coefficientMatrixForSort[j][i];
        }
    }

    /* Calculate sort temperature coefficient matrix X = (invA)*B */
    for(uint8_t i = 0u; i < 3u; i++)
    {
        for(uint8_t j = 0u; j < 3u; j++)
        {
            sortTempCoefficeintsX[i] += (sortTempValueMatrixB[j][0]*sortTempMatrixA[i][j]);
        }
    }

    /*****************************************************************************
     * Get the temperature value from the 2nd order equation
     *****************************************************************************/

    /* Rearrange the coefficients for the predicted temperature formula */
    coeffA = sortTempCoefficeintsX[2];
    coeffB = sortTempCoefficeintsX[1];
    coeffC = sortTempCoefficeintsX[0];

    /* -40degC -- SORT2, 27degC -- SORT3, 130degC -- CHI and by default reference value is
SORT3 */
    tempRefValue = refValue->adcVbgValues.roomValue;

    /* Conditional label for recalculating roots */
    RECALCULATE:

        /* Calculate the correction factor (k) to remove dependency of the ADC on the reference
voltage */
        correctionFactor = (tempRefValue) / (rawValue->adcVbgRawValue);

        /* Scale the data in raw with k */  /* (7.2) Get the scaled V_BE_NEW */
        tempScaleValue = (correctionFactor) * (rawValue->adcVtempRawValue);
```

```
        /* Calculate the predicted temperature */  /* (7.3) Calculate the temperature */
        bSqrMin4ac = ( pow(coeffB, 2.0)) - (((4.0)*(coeffA))*(coeffC - tempScaleValue));
        tempRootNeg = ((-coeffB)-(sqrtf(bSqrMin4ac))) / ((2.0)*(coeffA));
        tempRootPos = ((-coeffB)+(sqrtf(bSqrMin4ac))) / ((2.0)*(coeffA));     // this can be
minimize, kept for comparition


        /* Select the root that lies between the Hot and Cold temperature sort values [-40
degC, 150 degC] */  /* (7.4) Accuracy improvement by using V_BG from the nearest temperature */
        if((tempRootPos < (refValue->adcTempValues.hotValue)) && (tempRootPos > (refValue-
>adcTempValues.coldValue)))
        {
            /* Temperature value is positive root of the curve */
            tempInDegreeC = tempRootPos;
        }
        else if((tempRootNeg < (refValue->adcTempValues.hotValue)) && (tempRootNeg > (refValue-
>adcTempValues.coldValue)))
        {
            /* Temperature value is negetive root of the curve */
            tempInDegreeC = tempRootNeg;
        }
        else
        {
            /* Apt value is not found */
            tempInDegreeC = 0.0;
        }

        /* Check for the close proximity of calculated temperature with the reference
temeprature values */
        if (tempInDegreeC <= ((refValue->adcTempValues.coldValue) + tempRangeOffset))
        {
            /* Use SORT2 value to scale the measured temperature */
            tempRefValue = refValue->adcVbgValues.coldValue;
            goto RECALCULATE;
        }
        else if (tempInDegreeC >= ((refValue->adcTempValues.hotValue) - tempRangeOffset))
        {
            /* Use CHI value to scale the measured temperature */
            tempRefValue = refValue->adcVbgValues.hotValue;
            goto RECALCULATE;
        }
        else if ((tempInDegreeC <= ((refValue->adcTempValues.roomValue) + tempRangeOffset))
            && (tempInDegreeC >= ((refValue->adcTempValues.roomValue) - tempRangeOffset)))
        {
            /* Use SORT3 value to scale the measured temperature */
            tempRefValue = refValue->adcVbgValues.roomValue;
        }
        else
        {
            /* Fail safe case */
        }
```

```
    return tempInDegreeC;
}
```

用語集

用語	説明
SAR ADC	Successive Approximation Register Analog-to-Digital Converter (逐次比較型アナログ デジタル コンバータ)
SARMUX	Analog input multiplexer (アナログ インプット マルチプレクサ)
TCPWM	Timer, Counter, and Pulse Width Modulator (タイマ, カウンタ, およびパルス幅変調器)
VREFH	High reference voltage (上限基準電圧)
VREFL	Low reference voltage (下限基準電圧)
VBG	バンドギャップリファレンス電圧
VBE	温度センサ出力

関連資料

以下は、TRAVEO™ T2G ファミリシリーズのデータシートとテクニカルリファレンスマニュアルです。これらドキュメントの入手についてはインフィニオン サポートに連絡してください。

[1] デバイスデータシート

- CYT2B6 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- CYT2B7 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- CYT2B9 datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- CYT2BL datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family
- CYT3BB/4BB datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
- CYT4BF datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
- CYT6BJ datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-33466)
- CYT3DL datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
- CYT4DN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family
- CYT4EN datasheet 32-bit Arm® Cortex®-M7 microcontroller TRAVEO™ T2G family (Doc No. 002-30842)
- CYT2CL datasheet 32-bit Arm® Cortex®-M4F microcontroller TRAVEO™ T2G family

[2] Body Controller Entry ファミリ

- TRAVEO™ T2G automotive body controller entry family architecture technical reference manual (TRM)
- TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B7
- TRAVEO™ T2G automotive body controller entry registers technical reference manual (TRM) for CYT2B9
- TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT2BL (Doc No. 002-29852)

[3] Body Controller High ファミリ

- TRAVEO™ T2G automotive body controller high family architecture technical reference manual (TRM)
- TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT3BB/4BB
- TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT4BF
- TRAVEO™ T2G automotive body controller high registers technical reference manual (TRM) for CYT6BJ (Doc No. 002-36068)

[4] Cluster 2D ファミリ

- TRAVEO™ T2G automotive cluster 2D architecture technical reference manual (TRM)
- TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT3DL
- TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT4DN
- TRAVEO™ T2G automotive cluster 2D registers technical reference manual (TRM) for CYT4EN (Doc No. 002-35181)

[5] Cluster entry family

- TRAVEO™ T2G automotive cluster entry family architecture technical reference manual (TRM)
- TRAVEO™ T2G automotive cluster entry registers technical reference manual (TRM) for CYT2CL

その他の参考資料

さまざまな周辺機器にアクセスするためのサンプルソフトウェアとしてのスタートアップを含むサンプルドライバライブラリ (SDL) が提供されます。SDL は、公式の AUTOSAR 製品でカバーされないドライバの顧客へのリファレンスとしても機能します。SDL は自動車規格に適合していないため、製造目的で使用できません。このアプリケーションノートのプログラムコードは SDL の一部です。SDL の入手については、テクニカルサポートに連絡してください。

改訂履歴

版数	発行日	変更内容
**	2018-05-16	これは英語版 002-19755 Rev. **を翻訳した日本語版です。
*A	2018-12-17	これは英語版 002-19755 Rev. *A を翻訳した日本語版 002-23557 Rev. *A です。英語版の改訂内容: Changed target part number (CYT2B series). Added Glossary section.
*B	2019-05-23	これは英語版 002-19755 Rev. *B を翻訳した日本語版 002-23557 Rev. *B です。英語版の改訂内容: Added target part number (CYT4B series).
*C	2019-12-17	これは英語版 002-19755 Rev. *C を翻訳した日本語版 002-23557 Rev. *C です。英語版の改訂内容: Added target part number (CYT4D series).
*D	2020-07-17	これは英語版 002-19842 Rev. *D を翻訳した日本語版 002-23842 Rev. *D です。主な変更点は以降のとおりです。対象の製品番号を変更しました。(CYT2/ CYT4 シリーズ) Added target parts number (CYT3 series).
英語版*E	-	本版は英語版のみの発行です。英語版の改訂内容: Added flowchart and example codes. Moved to Infineon template.
*E	2021-05-25	これは英語版 002-19755 Rev. *F を翻訳した日本語版 002-23557 Rev. *E です。 英語版の改訂内容: Updated “8.2 Diagnosis Function”. Updated “9.3 Calibration Function”.
*F	2022-11-01	これは英語版 002-19755 Rev. *G を翻訳した日本語版 002-23557 Rev. *F です。英語版の改訂内容: Updated “5.1 Averaging settings”. Updated “9.2 Gain adjustment direction”. Added “10 Temperature Sensor”. Updated “12 Related Documents”.
英語版*H	-	本版は英語版のみの発行です。英語版の改訂内容: Updated MSB stretch mode from 0 to 1. Updated “9.4 Offset adjustment procedure”. Updated “9.5 Gain adjustment procedure”. Updated “10.1 Temperature measurement procedure”.
*G	2022-08-23	これは英語版 002-19755 Rev. *I を翻訳した日本語版 002-23557 Rev. *G です。英語版の改訂内容: Updated “9.4.3 Sample code”.
*H	2022-11-01	これは英語版 002-19755 Rev. *J を翻訳した日本語版 002-23557 Rev. *H です。英語版の改訂内容: Updated “1 Introduction”. Updated “9.4 Offset adjustment procedure”. Updated “9.4.3 Sample code”. Updated “9.4.5 Sample code”. Updated “12 Related documents”.
*I	2023-07-25	これは英語版 002-19755 Rev. *K を翻訳した日本語版 002-23557 Rev. *I です。英語版の改訂内容: Updated section 8.
*J	2024-03-27	これは英語版 002-19755 Rev. *L を翻訳した日本語版 002-23557 Rev. *J です。英語版の改訂内容: Template update; no content update.
*K	2025-04-18	これは英語版 002-19755 Rev. *M を翻訳した日本語版 002-23557 Rev. *K です。英語版の改訂内容: Updated 関連資料 and added CYT6 series

Trademarks

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

Edition 2025-04-18

Published by

Infineon Technologies AG

81726 Munich, Germany

Do you have a question about any aspect of this document?

Email: erratum@infineon.com

Document reference
IFX-eue1681235520749

重要事項

本手引書に記載された情報は、本製品の使用に関する手引きとして提供されるものであり、いかなる場合も、本製品における特定の機能性能や品質について保証するものではありません。本製品の使用の前に、当該手引書の受領者は実際の使用環境の下であらゆる本製品の機能及びその他本手引書に記された一切の技術的情報について確認する義務が有ります。インフィニオンテクノロジーズはここに当該手引書内で記される情報につき、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

本文書に含まれるデータは、技術的訓練を受けた従業員のみを対象としています。本製品の対象用途への適合性、およびこれら用途に関連して本文書に記載された製品情報の完全性についての評価は、お客様の技術部門の責任にて実施してください。

警告事項

技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。

インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。