TRAVEO™ T2G の割込みの使用方法

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では、TRAVEO™ T2G ファミリ MCU の割込みの設定および使用方法について説明します。このドキュメントは、割込みベースのプロジェクトを開発するためのガイドとして役立ちます。また、割込み構造, ベクタ選択, およびハンドラコード記述に関する注意事項についても説明します。さらに、障害情報を収集する Fault report structure を設定および使用する方法についても説明します。

対象者

このドキュメントは、TRAVEO™ T2G ファミリの割込み機能を使用するすべての人を対象とします。

目次

目次

1 はじめに

このアプリケーションノートでは、TRAVEO™ T2G ファミリ MCU の割込みについて説明します。本シリーズは、Arm® Cortex®-M CPU と暗号化ハードウェア (eSHE), CAN FD, メモリ, アナログおよびデジタルの周辺機能を 1 つのチップに搭載します。

CYT2 シリーズは、1 つの Arm® Cortex®-M4F ベースの CPU (CM4) と Cortex®-M0+ベースの CPU (CM0+) があります。CYT3 シリーズは、1 つの Arm® Cortex®-M7 ベースの CPU (CM7) と CM0+ があります、CYT4 シリーズは 2 つの Arm® Cortex®-M7 ベースの CPU (CM7) と CM0+ があり、CYT6 シリーズは 4 つの Arm® Cortex®-M7 ベースの CPU (CM7) と CM0+ があります。

割込みは、組込みアプリケーションの重要な部分です。特定のイベントが発生した時のみ、CPU に通知することによって、CPU はそのイベントの発生を連続的にポーリングする必要がなくなります。割込みが発生すると、CPU は、現在のプログラムフローを停止し割込みサービスルーチン (ISR) を実行します。

組込みアプリケーションでは、割込みは周辺機能からの状態通知に頻繁に使用されます。また、周辺機能からのエラー検出時の通知にも使用されます。

このドキュメントでは、周辺機能からの割込みをシステム割込みと呼びます。本シリーズは、最大 1023 のシステム割込みをサポートします。デバイスの種類によってサポートされるシステム割込みは、デバイスデータシート [1]を参照してください。

Fault report structure は、ソフトウェア実行中に発生する故障を集約します。TRAVEO™ T2G プラットフォームは、集中型の Fault report structure を採用しているため、システム全体にわたって一貫したフォールト処理を行え、ソフトウェア開発を簡素化できます。

Fault report structure は、発生した故障をシステムに通知するための信号インタフェースを持ちます。

このアプリケーションノートで使用される機能と用語については、アーキテクチャ テクニカル リファレンス マニュアル (Architecture TRM) [2] の”Interrupts”と”Fault Subsystem”章を参照してください。

2 割込みの概要

2.1 割込み構造

図 1 に、CYT2B シリーズの割込みアーキテクチャを示します。

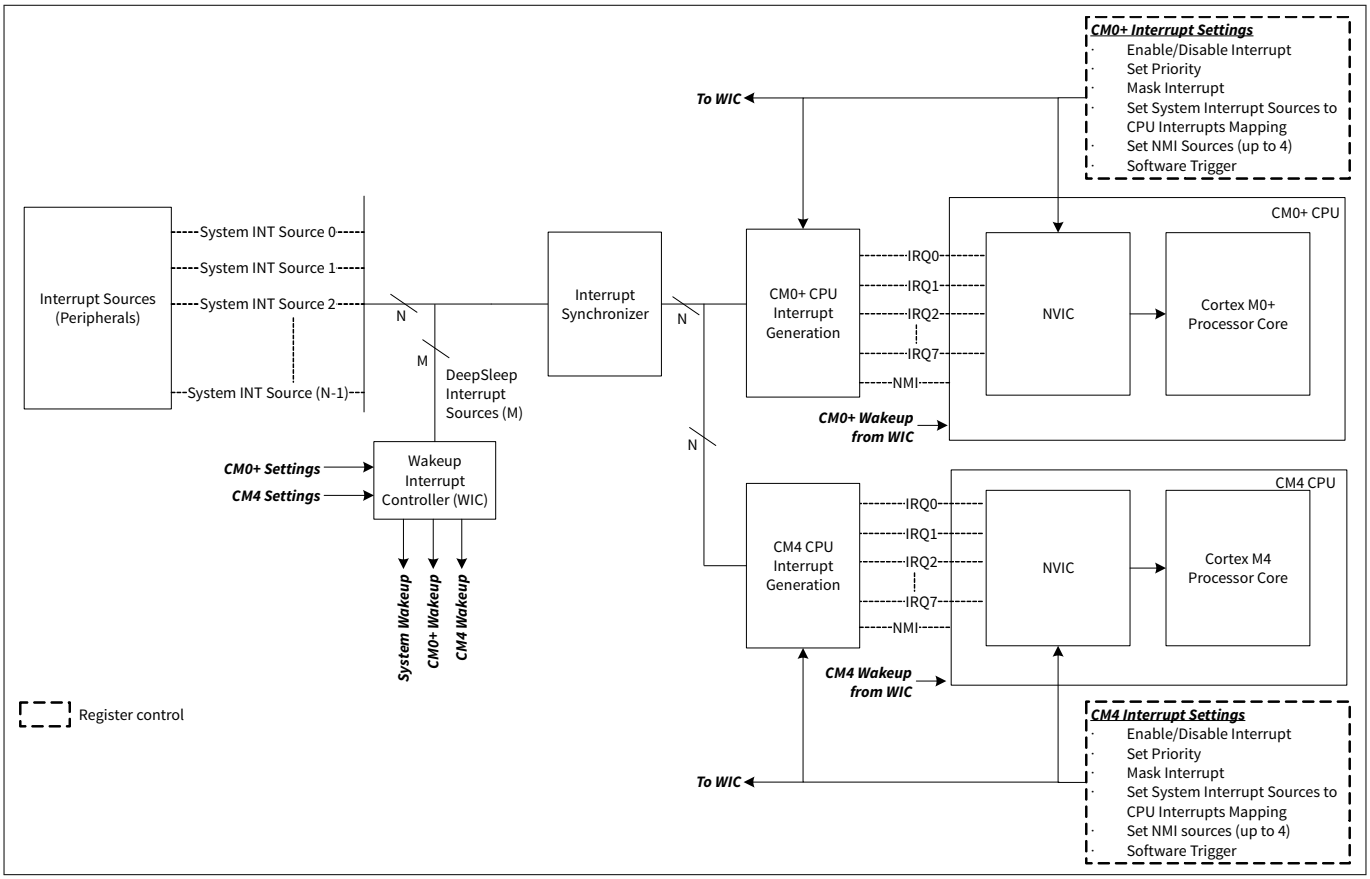

図 1 CYT2B シリーズ割込みアーキテクチャ

その他のシリーズの割込みアーキテクチャについては、アーキテクチャ TRM [2]を参照してください。

本シリーズのシステム割込みは各コアの NVIC によって処理されます。

TRAVEO™ T2G の割込みアーキテクチャでは、各 CPU は 8 つの CPU 割込み IRQ[7:0]を使用でき、N 個のシステム割込みは、各 CPU の CPU 割込み IRQ[7:0]のいずれかにマッピングできます。すべてのシステム割込みは、任意の CPU 割込みに同時にマッピングでき、両 CPU に対する独立したマッピングもできます。

複数のシステム割込みを同じ CPU にマッピングできます。したがって、アクティブな CPU 割込みは、1 つ以上のアクティブなシステム割込みを示す場合があります。

各割込みに対して、CM4, CM7 は 8 レベル、CM0+には 4 レベルの設定可能な優先レベルがあります。

NVIC に加え、TRAVEO™ T2G はウェイクアップ割込みコントローラ (WIC) と割込みシンクロナイザーブロックをサポートします。

WIC は、DeepSleep パワーモードで DeepSleep 割込みを検出します。1 つ以上のウェイクアップ要因割込みを検出すると、WIC はウェイクアップ信号を生成し、CPU をアクティブモードに遷移させます。DeepSleep ウェイクアップに使用可能な割込みソースについてはデバイスデータシート [1]を参照してください。

Synchronizer ブロックは、CPU クロックに割込みを同期させます。

最大 4 つのシステム割込みを、各 CPU の NMI にマッピングできます。

2.2 動作概要

図 2 に、割込みシーケンスでの CPU 動作を示します。

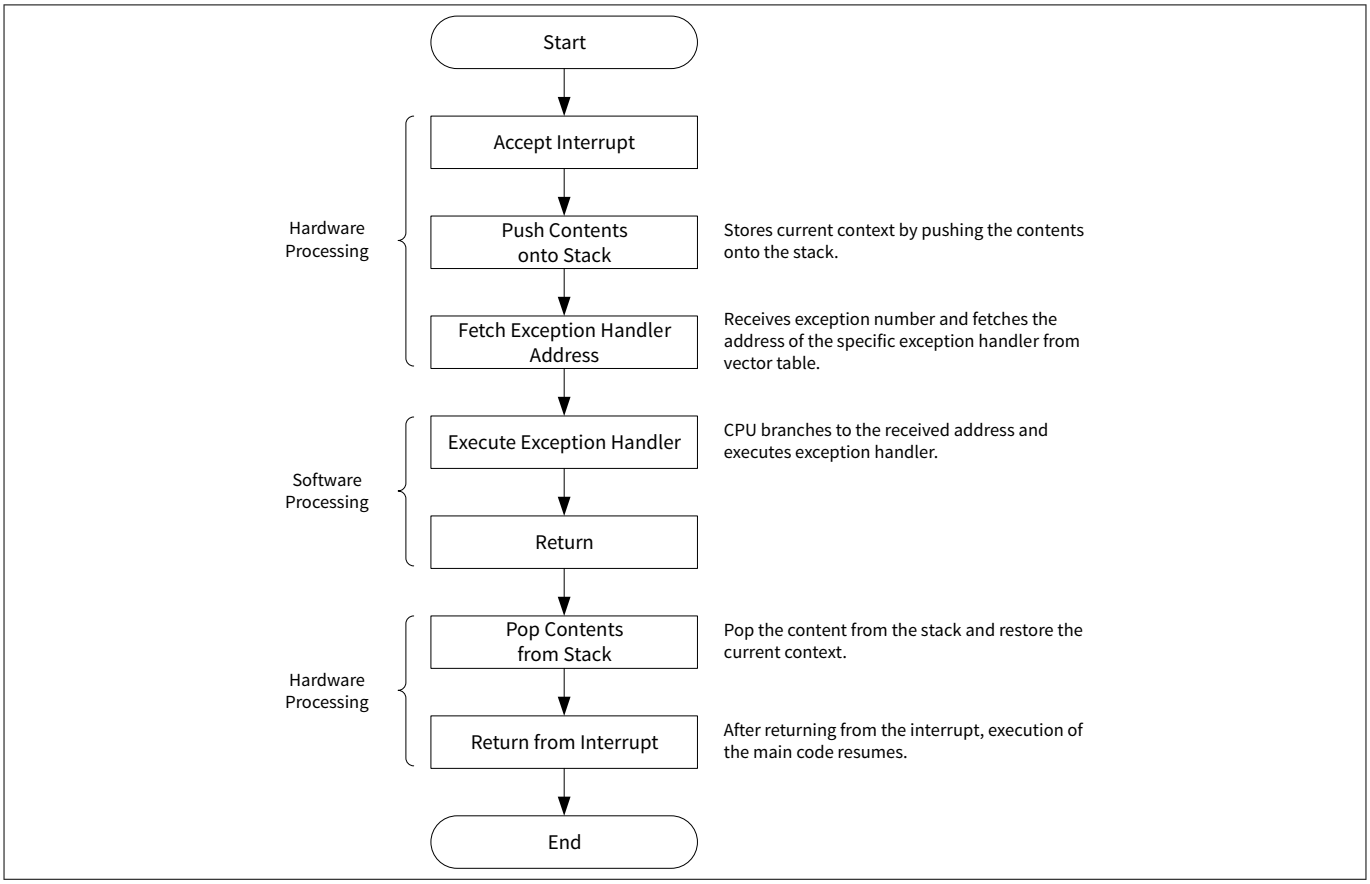

図 2 割込みシーケンス

NVIC は、他の割込み処理を実行中に割込み要求を受信するか、複数の割込みを同時に受信した場合、これらすべての割込みの優先度を評価し、最も高い優先度を持つ割込みの例外番号を CPU に通知します。したがって、より高い優先度の割込みは、より低い優先度の割込み処理の実行をブロックする場合があります。

図 3 に、割込み優先度による割込み処理を示します。

2 割込みの概要

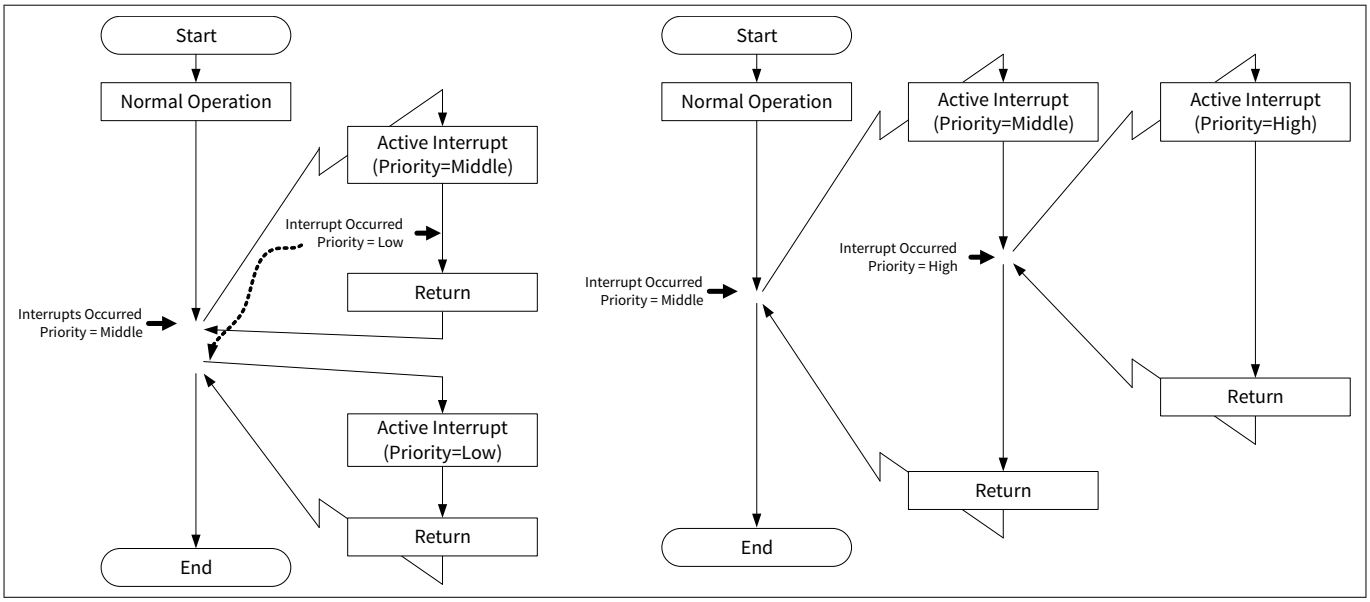

図 3　　割込み処理中に割込みが発生した場合の動作

詳細は、Arm®ドキュメンテーションセット (CM4, CM7, および CM0+) を参照してください。

2.3　　初期設定

ここではサンプルドライバライブラリ(SDL)を使用して、割込みの初期設定を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。SDL 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。図 4 に、割込みの初期設定フローを示します。

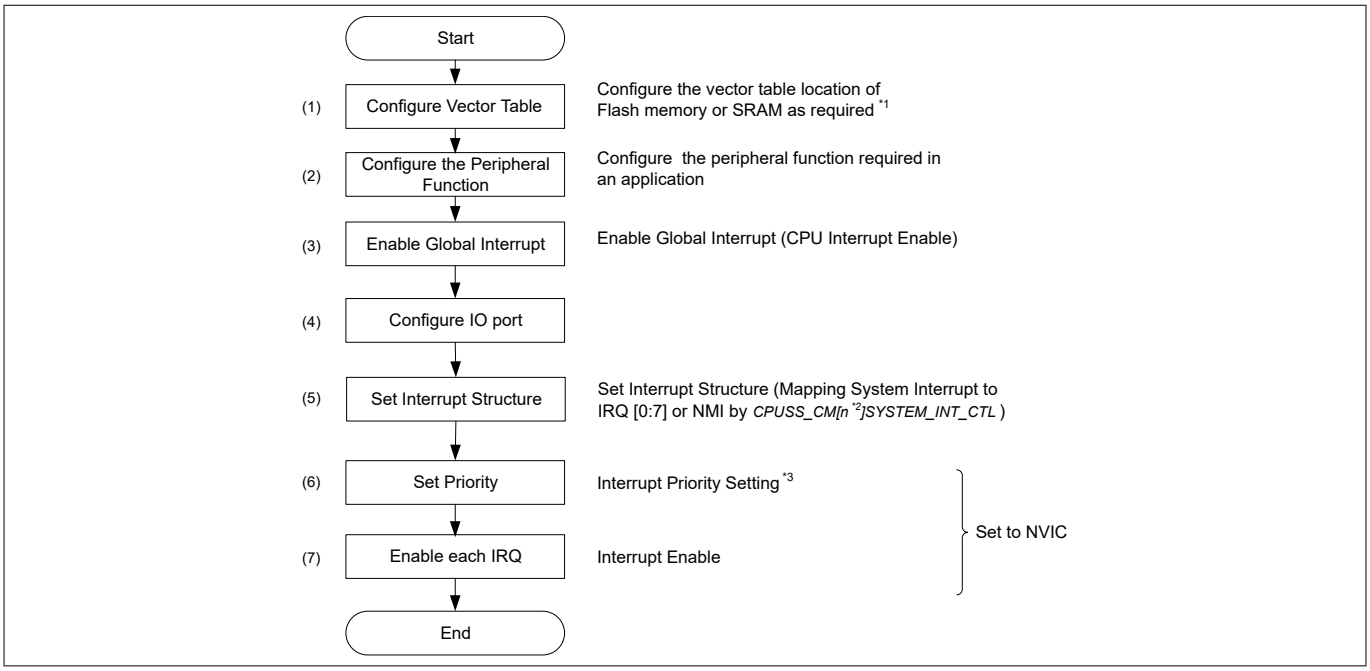

図 4　　割込み初期設定フロー

注: *(*1) ベクタテーブルオフセットレジスタ(VTOR) はベクタテーブルの設定に使用します。VTOR の下位 7 ビットは予約されています。したがって、ベクタテーブルは 128 バイトの境界に整列する必要があります。*

注: *(*2) 図 4 の n は 0, 4 を示します。例えば、CPUSS_CM4_SYSTEM_INT_CTL は CM4 のシステム割込み制御レジスタを示します。CYT2 シリーズには、CPUSS_CM4_SYSTEM_INT_CTL と CPUSS_CM0_SYSTEM_INT_CTL レジスタがあります。CYT3 および CYT4 シリーズには、CPUSS_CM7_0_SYSTEM_INT_CTL, CPUSS_CM7_1_SYSTEM_INT_CTL (CYT4/CYT6 シリーズのみ), CPUSS_CM7_2_SYSTEM_INT_CTL (CYT6 シリーズのみ), CPUSS_CM7_3_SYSTEM_INT_CTL (CYT6 シリーズのみ), および CPUSS_CM0_SYSTEM_INT_CTL レジスタがあります。詳細はレジスタ TRM [2] を参照してください。*

注: *(*3) 割込みの優先順位は、NVIC_IPR レジスタの 8 ビットの PRI_N フィールドで設定できます。割込み番号とレジスタのビットフィールド間の対応は、以下の式で表すことができます: IRQ 番号 = NVIC_IPR レジスタ番号 *4 + PRI_N ビット フィールド番号(例えば、IRQ5 = NVIC_IPR1.PRI_N1)*

各 CPU には VTOR があります。これは、フラッシュから SRAM へのベクタテーブルのリロケーションに使用でき、割込みハンドラの動的な変更が可能です。VTOR を SRAM 領域に設定する場合、ベクタテーブルは SRAM に配置してください。

注: *ソフトウェアは、デフォルトのハードフォールトベクターエントリが特定のユーザーハンドラに置き換えられていることを確認する必要があります。デフォルトの SROM ハンドラは、起動時にのみ使用されるように設計されています。*

2.3.1 ユースケース

ここでは、以下のユースケースを使用した割込み設定例について説明します。

この設定では、GPIO Poer 8_0 ピンがボタンスイッチに接続されます。スイッチを押すことにより割込みが発生します。ここで使用される IDX 番号は、システム割込みのインデックス番号を示します。ここで使用する IDX 番号は CYT2B シリーズの番号です。実際に使用するインデックス番号については、デバイスデータシート [1]を参照してください。GPIO 設定については、アーキテクチャ TRM [2]の“I/O System”章およびアプリケーションノート [3]を参照してください。

- システム割込みソース: GPIO ポート割込み#8 (IDX: 29)
- CPU 割込みに割当て: IRQ3
- CPU 割込み優先順位: 3

2.3.2 コンフィグレーション

表 1 および表 2 に、割込み初期化のための SDL 設定部のパラメータと関数を示します。

表 1 割込み初期化パラメータ

パラメータ	説明	値
irq_cfg.sysIntSrc	システム割込みインデックス番号 [0: 1022]	CY_BUTTON3_IRQN Port 8 に割当て (インデックス番号 = 29)

(続く)

2 割込みの概要

表 1 (続き) 割込み初期化パラメータ

パラメータ	説明	値
irq_cfg.intIdx	CPU 割込み設定 [0: 7] CPUIntIdx0_IRQn は IRQ 0 に割当て CPUIntIdx1_IRQn は IRQ 1 に割当て CPUIntIdx2_IRQn は IRQ 2 に割当て CPUIntIdx3_IRQn は IRQ 3 に割当て CPUIntIdx4_IRQn は IRQ 4 に割当て CPUIntIdx5_IRQn は IRQ 5 に割当て CPUIntIdx6_IRQn は IRQ 6 に割当て CPUIntIdx7_IRQn は IRQ 7 に割当て	CPUIntIdx3_IRQn
irq_cfg.isEnabled	割込みイネーブル設定 False: ディセーブル True: イネーブル	True

表 2 割込み初期化関数

関数	説明	値
Cy_SysInt_InitIRQ(irq_cfg)	割込みを設定 Irq_cfg: 割込み設定パラメータアドレス	&irq_cfg
Cy_SysInt_SetSystemIrqVector (sysintSrc, Handler)	システム割込みハンドラをベクタテーブルに設定 sysintSrc: システム割込みインデックス番号 Handler: 割込みハンドラアドレス	sysintSrc: CY_BUTTON3_IRQN Handler: ButtonIntHandler, Code Listing 6 を参照してください。
NVIC_SetPriority(intSrc, intPriority)	割込み優先順位設定: intSrc: CPU 割込み番号 intPriority: 割込み優先順位レベル	intSrc: CPUIntIdx3_IRQn intPriority: 3ul
NVIC_ClearPendingIRQ(intSrc)	ペンディングフラグクリア: intSrc: CPU 割込み番号	CPUIntIdx3_IRQn
NVIC_EnableIRQ(intSrc)	割込みイネーブル設定 intSrc: CPU 割込み番号	CPUIntIdx3_IRQn

Code Listing 1 と Code Listing 2 に割込み設定部分のプログラム例を示します。

以下の説明は、SDL ドライバ部の表記例を示します。

- **CPUSS**->un**unCM4_SYSTEM_INT_CTLx**.u32Register はレジスタ TRM [2]に記載される CPUSS_CM4_SYSTEM_INT_CTLx レジスタです。他のレジスタも同様に記述されます。“**x**”は、システム割込みインデックス番号を示します。
- パフォーマンスの改善策

2 割込みの概要

レジスタ設定のパフォーマンス向上のため、SDL は完全な 32 ビットデータをレジスタに書込みます。各ビットフィールドはビット書込み可能なバッファに事前に生成され、最終的に 32 ビットデータとしてレジスタに書き込まれます。

```
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;
```

レジスタの共用体および構造体の詳細については *hdr/rev_x/ipn の cyip_cpuss_v2.h* を参照してください。

Code Listing 1 ベクタテーブルの設定例

```
/* (1) Configure Vector Table */
static void CopyVectorTable(void)
{
    const uint8_t    u8NrOfVectors = (uint8_t) ((uint32_t) &__Vectors_Size / 4);
    /* SRAM vector table address */
    uint32_t * const pu32RamTable = (uint32_t *) __ramVectors;
    /* Flash memory vector table address */
    uint32_t * const pu32RomTable = (uint32_t *) (&__vector_table);


    /* Copy vector table from Flash memory to SRAM */
    for(uint8_t u8Index = 0; u8Index < u8NrOfVectors; u8Index++)
    {
        pu32RamTable[u8Index] = pu32RomTable[u8Index];
    }

    SCB->VTOR = (uint32_t) pu32RamTable;    /* Set VTOR to SRAM */
}
```

このコードはスタートアップ中に呼び出されます。

Code Listing 2 割込みの設定例

```
#define CY_BUTTON3_PORT                 GPIO_PRT8    /* IO port definition */
#define CY_BUTTON3_PIN                  0       /* IO port configuration */

cy_stc_gpio_pin_config_t user_button3_port_pin_cfg =
{
    .outVal    = 0ul,
    .driveMode = CY_GPIO_DM_HIGHZ,
    .hsiom     = CY_BUTTON3_PIN_MUX,
    .intEdge   = CY_GPIO_INTR_FALLING,
    .intMask   = 1ul,
    .vtrip     = 0ul,
    .slewRate  = 0ul,
    .driveSel  = 0ul,
    .vregEn    = 0ul,
    .ibufMode  = 0ul,
    .vtripSel  = 0ul,
    .vrefSel   = 0ul,
    .vohSel    = 0ul,
};

void ButtonIntHandler(void)         /* System interrupt handler */
{
:
}

int main(void)
{
    SystemInit();         /* (2) Peripheral function setting */

    __enable_irq(); /* (3) Enable global interrupt (*1). */


    /* (4) Configure IO Port. */
    Cy_GPIO_Pin_Init(CY_BUTTON3_PORT, CY_BUTTON3_PIN, &user_button3_port_pin_cfg);

    /* Setup GPIO for BUTTON3 interrupt */
    /* (5) Set interrupt structure. See Code Listing 3 and Code Listing 4 for details of each
function. */
    cy_stc_sysint_irq_t irq_cfg =
    {
        .sysIntSrc  = CY_BUTTON3_IRQN,
        .intIdx     = CPUIntIdx3_IRQn,
        .isEnabled  = true,
    };

    /* (5) Set interrupt structure. See Code Listing 3 and Code Listing 4 for details of each
function. */
    Cy_SysInt_InitIRQ(&irq_cfg);
    /*  System interrupt handler */
    /* (5) Set interrupt structure. See Code Listing 3 and Code Listing 4 for details of each
function. */
```

```
    Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, ButtonIntHandler);

    /*  (6) Set priority (*1) */
    /* (5) Set interrupt structure. See Code Listing 3 and Code Listing 4 for details of each
function. */
    NVIC_SetPriority(CPUIntIdx3_IRQn, 3ul);
    /*  (7) Interrupt Enable (*1) */
    /* (5) Set interrupt structure. See Code Listing 3 and Code Listing 4 for details of each
function. */
    NVIC_EnableIRQ(CPUIntIdx3_IRQn);

    for(;;)
    {
    }
}
```

(*1) プログラミングヒント: CM4/CM0+割込み許可命令および NVIC 命令は、Cortex®マイクロコントローラのソフトウェアインタフェーススタンダード (CMSIS) で提供されます。

Code Listing 3 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;

   un_CPUSS_CM4_SYSTEM_INT_CTL_t unIntCtl = { 0ul };

    /*  Check if configuration parameter values are valid  */
    if(NULL != config)
    {
      unIntCtl.stcField.u3CPU_INT_IDX = (uint8_t)config->intIdx;
      unIntCtl.stcField.u1CPU_INT_VALID = config->isEnabled ? 1ul : 0ul;
      /*  Set the parameters to interrupt structure  */
      CPUSS->unCM4_SYSTEM_INT_CTL[config->sysIntSrc].u32Register = unIntCtl.u32Register;
    }
    else
    {
        status = CY_SYSINT_BAD_PARAM;
    }
    return(status);
}
```

Code Listing 4 Cy_SysInt_SetSystemIrqVector() 機能

```
void  Cy_SysInt_SetSystemIrqVector(cy_en_intr_t sysIntSrc, cy_systemIntr_Handler userIsr)
{
    /*  Check the vector location  */
    if (Cy_SysInt_SystemIrqUserTableRamPointer != NULL)
    {
        /*  Set the system interrupt handler  */
        Cy_SysInt_SystemIrqUserTableRamPointer[sysIntSrc] = userIsr;
    }
}
```

2.4 割込み処理

本シリーズ MCU では、複数のシステム割込みは、同じ CPU 割込みにマッピングできます。したがって、これらのシステム割込みは CPU 割込みハンドラを共有します。

CPU 割込み発生時にどのシステム割込みが発生したかを識別するため、各 CPU 割込みは CMn_INT_STATUS レジスタがあります。レジスタには、以下のフィールドがあります。

- CMn_INTx_STATUS.SYSTEM_INT_VALID: CPU 割込みに対し、有効なシステム割込みの有無を示します
- CMn_INTx_STATUS.SYSTEM_INT_IDX [9:0]: CPU 割込みを発生させた有効なシステム割込みの中で最も低いインデックス ([0, 1022]の範囲の数値) を持つ割込みを示します。

CPU 割込みハンドラでは SYSTEM_INT_IDX を使用して、割込みルックアップテーブルを検索し、システム割込みハンドラにジャンプします。

本シリーズ MCU は、これらの割込みはレベル割込みであるため、復帰処理内で周辺割込みをクリアする必要があります。また、レジスタアクセス完了を確実にするためにレジスタをリードバックし、Pending レジスタをクリアした後、割込みから復帰する必要があります。図 5 に、CYT2B シリーズの割込み処理例を示します。

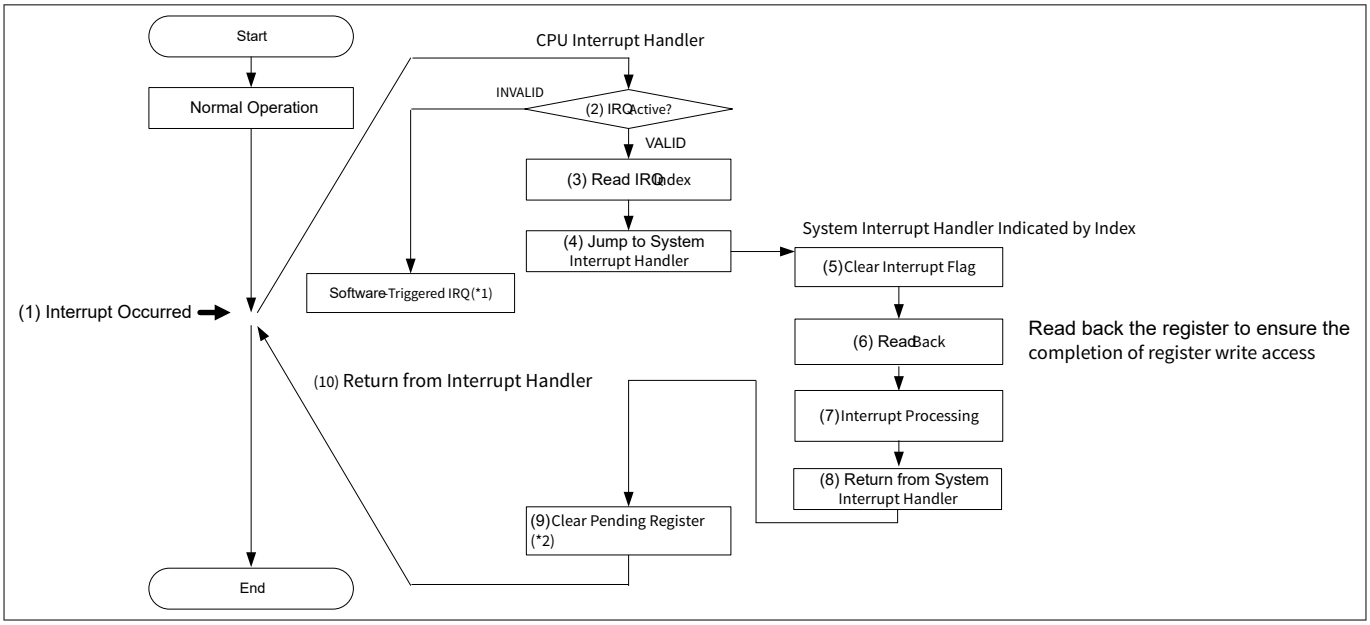

図 5 割込み処理

Code Listing 5 と Code Listing 6 に割込みハンドラのプログラム例を示します。

2 割込みの概要

Code Listing 5 CPU 割込みハンドラ

```
void CpuUserInt3_Handler(void)
{
    un_CPUSS_CM0_INT3_STATUS_t system_int_status = {0};

    system_int_status.u32Register = CPUSS->unCM4_INT3_STATUS.u32Register;

    if(system_int_status.stcField.u1SYSTEM_INT_VALID)   /*(2) Check if IQR is Active */
    {
// (3) Read interrupt index
// (4)jump to system interrupt handler
       Cy_SystemIrqUserTable[system_int_status.stcField.u10SYSTEM_INT_IDX]();
    }
    else
    {
     // Triggered by SW or due to SW clear error (SW cleared a peripheral
     // interrupt flag but didn't clear the Pending flag at NVIC)
     // Case of Software Trigger (*1)
    }

    //(9) Clear the IRQ pending flag executed by the Pending register in NVIC (*2)
    NVIC_ClearPendingIRQ(CPUIntIdx3_IRQn);
}
```

Code Listing 6 システム割込みハンドラ

```
// Interrupt handler registered in the configuration part
void ButtonIntHandler(void)
{
    uint32_t intStatus;

    /* If button3 falling edge detected */
                                                // Port number     Pin number in Port
    intStatus = Cy_GPIO_GetInterruptStatusMasked(CY_BUTTON3_PORT, CY_BUTTON3_PIN);
    if (intStatus != 0ul)
    {
        // Clear the peripheral interrupt flag that became the interrupt source. See Code
Listing 7.
        Cy_GPIO_ClearInterrupt(CY_BUTTON3_PORT, CY_BUTTON3_PIN);

        :    //(7) Interrupt processing
    }
}
```

2 割込みの概要

Code Listing 7 Cy_GPIO_ClearInterrupt() 機能

```
__STATIC_INLINE void Cy_GPIO_ClearInterrupt(volatile stc_GPIO_PRT_t* base, uint32_t pinNum)
{
    /* Any INTR MMIO registers AHB clearing must be preceded with an AHB read access */
    (void)base->unINTR.u32Register;

    base->unINTR.u32Register = CY_GPIO_INTR_STATUS_MASK << pinNum;  //(5) Clear interrupt flag

    /* This read ensures that the initial write has been flushed out to the hardware */
    (void)base->unINTR.u32Register;   //(6) Read back
}
```

***注**:* *(*1) ソフトウェアトリガ割込みレジスタ(STIR) を使用してソフトウェア割込みが発生した場合、SYSTEM_INT_VALID ビットはセットされません。ソフトウェア割込みについてはソフトウェア割込み (CPU レジスタを使用)を参照してください。*

***注**:* *(*2) ペンディングレジスタのフラグ(保留中のフラグ) は、CPU 割込みが受け付けられるとクリアされます。したがって、通常は CPU 割込みハンドラ内でクリアする必要はありません。ただし、1 つの割込みハンドラで複数の割込みを処理する場合は、注意が必要です。次の場合、割込みからの復帰時にペンディングフラグをクリアすることで無効な割込みを防ぐことができます。*

***注**:* *図 6 に 2 つの割込み要因(割込み 1 と割込み 2) が 1 つのシステム割込みハンドラで処理する場合を示します。*

2 割込みの概要

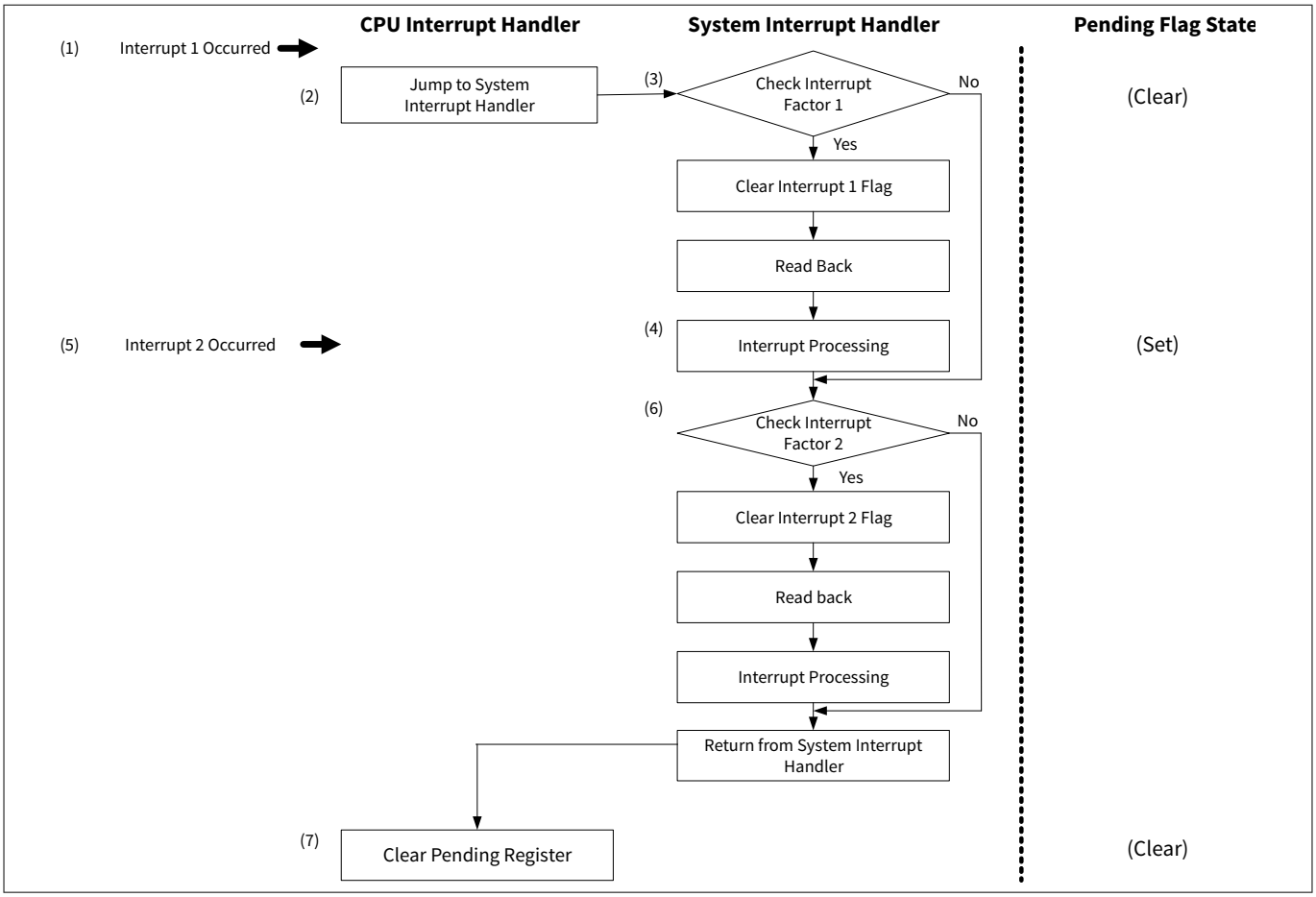

図 6 ペンディングフラグクリア例

1. 割込み 1 発生
2. CPU 割込みハンドラは、割込み要因を確認し、対応するシステム割込みハンドラにジャンプします。ここで、割込みが受け付けられたため、ペンディングフラグがクリアされます。
3. 割込み要因を確認します。割込み 1 の場合、割込み 1 フラグがクリアされます。
4. 割込み 1 を処理します。
5. 割込み 2 は割込み 1 の処理中に発生します。そのため、保留フラグが再びセットされます。
6. 割込み 1 の処理が完了後、割込み 2 を確認します。この例では割込み 2 の割込みフラグクリアおよび割込み処理を実行します。
7. ペンディングフラグをクリアし、割込みハンドラから復帰します。ペンディングフラグがクリアされていない場合、CPU 割込みハンドラから復帰後、再度割込みが生成されます。

2.5 周辺機能の割込み構成

ほとんどの周辺機能の割込みは、INTR, INTR_SET, INTR_MASK, および INTR_MASKED レジスタで構成されます。

- INTR: 割込み要因が発生したことを示します。ソフトウェアは”1”書込みによりクリアできます。
- INTR_SET: 割込みをセットします。ソフトウェアはビットへの”1”書込みにより対応する INTR ビットをセットできます。
- INTR_MASK: 割込みをマスクします。ソフトウェアは対応するビットへの”1”書込みにより割込みを選択的に有効にできます。
- INTR_MASKED: マスクされた割込み要因が発生したことを示します。これは割込み要求レジスタ (INTR) とマスクレジスタ (INTR_MASK) 間のビット単位の AND を反映します。

INTR_SET レジスタを使用して、ソフトウェア割込みを生成できます。詳細はソフトウェア割込み (周辺機能を使用)を参照してください。

2.6 割込み構成の使用例

割込み構成の使用例を使用想定にしたがって説明します。

ここで使用される IDX 番号は、システム割込みインデックス番号を示します。IDX 番号は、CYT2B シリーズのものです。実際に使用するインデックス番号についてはデバイスデータシート [1]を参照してください。

2.6.1 周辺機能割込み処理

ここでは、外部端子 (GPIO ポート#0,1,2) からの一般的な割込みについて説明し、その動作, 初期設定, および割込み処理を示します。各割込みは CM4 によって処理されます。以下にユースケースを示します。

2.6.1.1 ユースケース

- 割込み設定 1
  - システム割込みソース: GPIO ポート割込み#0 (IDX: 21) 立ち上りエッジ検出
  - CPU 割込みへ割当て: IRQ4
  - CPU 割込み優先順位: 2
- 割込み設定 2
  - システム割込みソース:
    - GPIO ポート割込み#1 (IDX: 22) 立ち上りエッジ検出
    - GPIO ポート割込み#2 (IDX: 23) 立ち上りエッジ検出
  - CPU 割込みへ割当て: IRQ5
  - CPU 割込み優先順位: 3

図 7 に、割込み動作を示します。

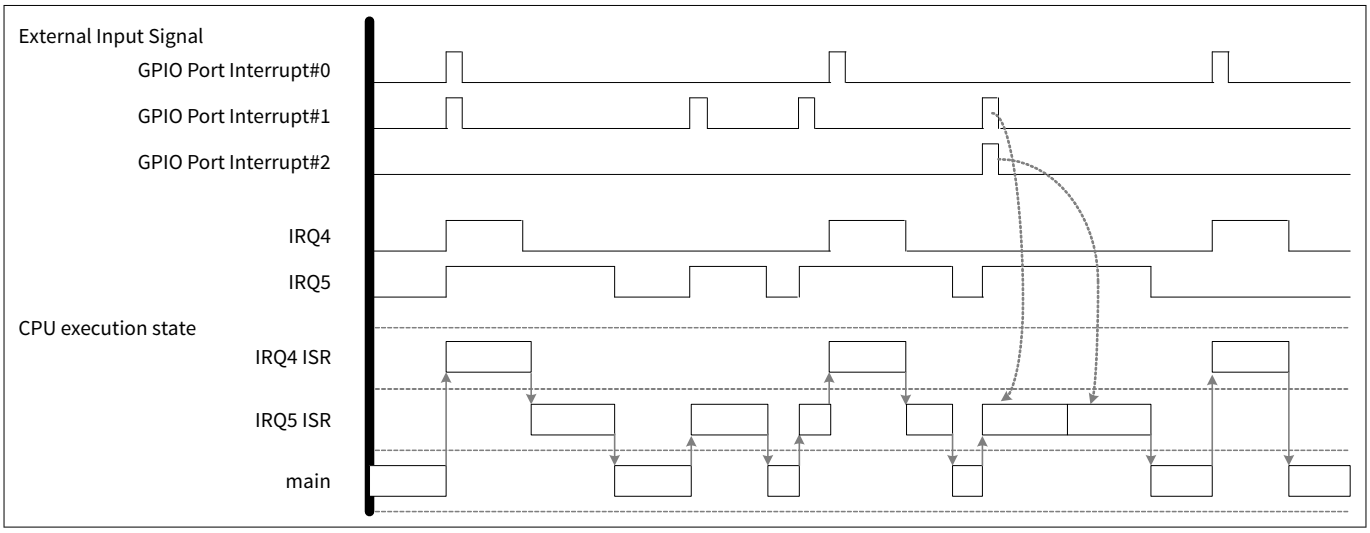

図 7 割込み動作例

GPIO Port Interrupt#0 と GPIO Port Interrupt#1 が同時に発生した場合、GPIO Port Interrupt#1 が割当てられた IRQ5 より GPIO Port Interrupt#0 が割当てられた IRQ4 が高い優先順位を持つため、GPIO Port Interrupt#0 が優先されます。

GPIO Port Interrupt#1 処理中に GPIO Port Interrupt#0 が発生した場合、GPIO Port Interrupt#1 が割当てられた IRQ5 より GPIO Port Interrupt#0 が割当てられた IRQ4 が高い優先順位を持つため、GPIO Port Interrupt#0 が優先されます。

GPIO Port Interrupt#1 と GPIO Port Interrupt#2 が同時に発生した場合、両割込みとも IRQ5 に割当てられていますが、GPIO Port Interrupt#2 のインデックス (23) よりも GPIO Port Interrupt#1 のインデックス (22) が低いため、GPIO Port Interrupt#1 が優先 (CM0/4_INT5_STATUS レジスタ) されます。

2.6.1.2 コンフィグレーション

- **初期設定手順**
  初期設定手順については、初期設定を参照してください。各周辺リソース (GPIO Port Interrupt#0,1,2) を設定後、各周辺リソースの割込みは CPU 割込みに接続されます。次に、各 CPU 割込みのレベルと許可を NVIC に設定します。割込み接続設定後、各リソースを起動します。GPIO の設定については、アーキテクチャ TRM [2]の“I/O System”章を参照してください。
- **割込みハンドラ動作**
  割込みハンドラの設定については、割込み処理を参照してください。割込みが発生すると、CPU はベクタテーブルで指定された割込みハンドラにジャンプします。CPU 割込みハンドラ内で CPU 割込みをトリガしたシステム割込みを識別し、対応する ISR にジャンプします。CPU 割込みハンドラのシステム割込みフラグをクリアし、ISR を実行後、関連する CPU ペンディングレジスタビットをクリアしてメインルーチンに復帰します。

図 8 と図 9 に、単一および複数の割込みが発生した場合のソフトウェアフローを示します。

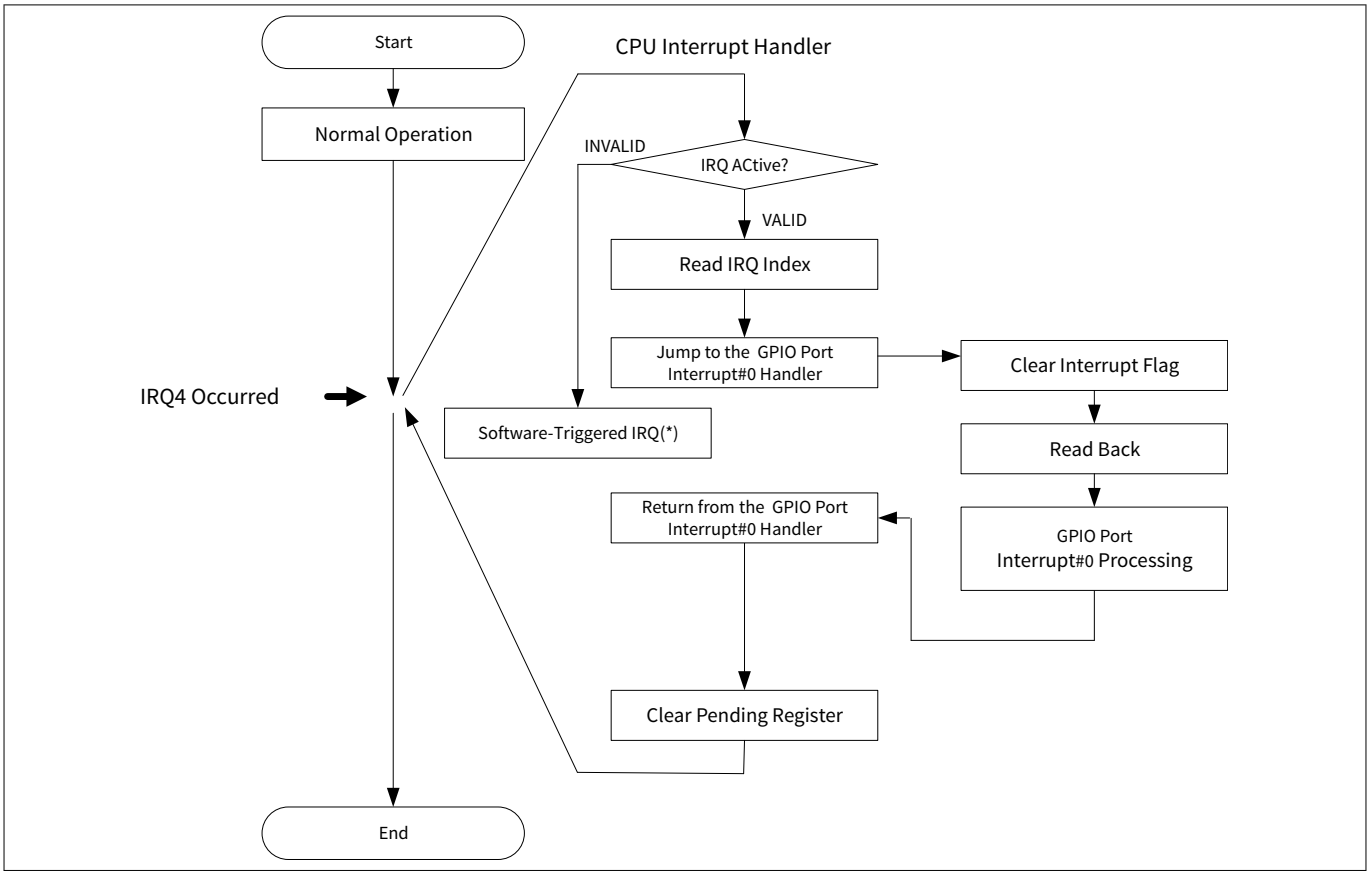

図 8 割込み動作フロー

注: (*) ソフトウェアトリガ割込みレジスタ(STIR) を使用してソフトウェア割込みが発生した場合、SYSTEM_INT_VALID ビットはセットされません。ソフトウェア割込みについてはソフトウェア割込み (CPU レジスタを使用)を参照してください。

2 割込みの概要

ISR を実行中に、より高い優先順位の CPU 割込みが発生した場合、現在の ISR を中断し、より高い優先順位の ISR を実行します。

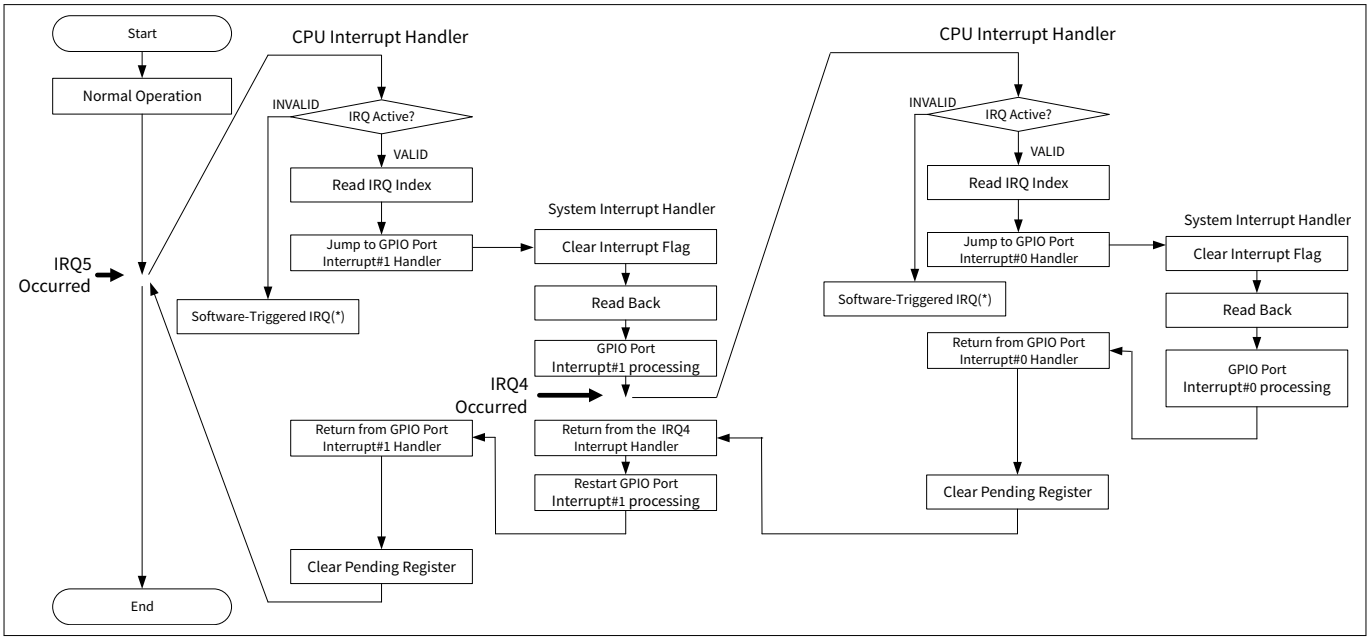

図 9　　割込みネスト時の動作フロー

注:　(*) ソフトウェアトリガ割込みレジスタ(STIR) を使用してソフトウェア割込みが発生した場合、SYSTEM_INT_VALID ビットはセットされません。ソフトウェア割込みについてはソフトウェア割込み (CPU レジスタを使用)を参照してください。

2.6.2　システム割込みによる NMI の生成

本シリーズでは、使用可能なすべてのシステム割込みのうち最大 4 つを各 CPU の NMI に割当てできます。ここでは、NMI を使用したタイミング保護の設定とその動作, 初期設定および割込み処理について説明します。各割込みは CM4 によって処理されます。

2.6.2.1　ユースケース

ここでは、TCPWM Group#0 Counter#1 によって呼び出される定期割込みルーチン (500 カウント) の処理時間を監視します。

TCPWM Group#0 Counter#0 は、割込みルーチンの処理時間を監視するために使用します。TCPWM Group#0 Counter#0 は、カウントアップするカウンタであり、カウント値が比較値 (100 カウント) を超えると割込みを生成します。定期割込みが始まるとカウントを開始し、割込み処理が完了すると停止します。割込み処理が特定の時間内に完了しない場合、つまりカウントが停止されない場合、TCPWM Group#0 Counter#0 アンダーフロー割込みによって NMI が CPU に通知されます。

- TCPWM Group#0 Counter#0
  - システム割込みソース: TCPWM Group#0 Counter#0(IDX: 274) TC 割込み
  - CPU NMI に割当て
  - IRQ5 ISR 実行時間を監視
  - コンペア値: 100 (監視期間)
- TCPWM Group#0 Counter#1
  - システム割込みソース: TCPWM Group#0 Counter#1(IDX: 275) TC 割込み

- CPU 割込みに割当て: IRQ5
- CPU 割込み優先順位: 3
- コンペア値: 500 (システム動作における割込み周期)

図 10 に、割込み動作を示します。

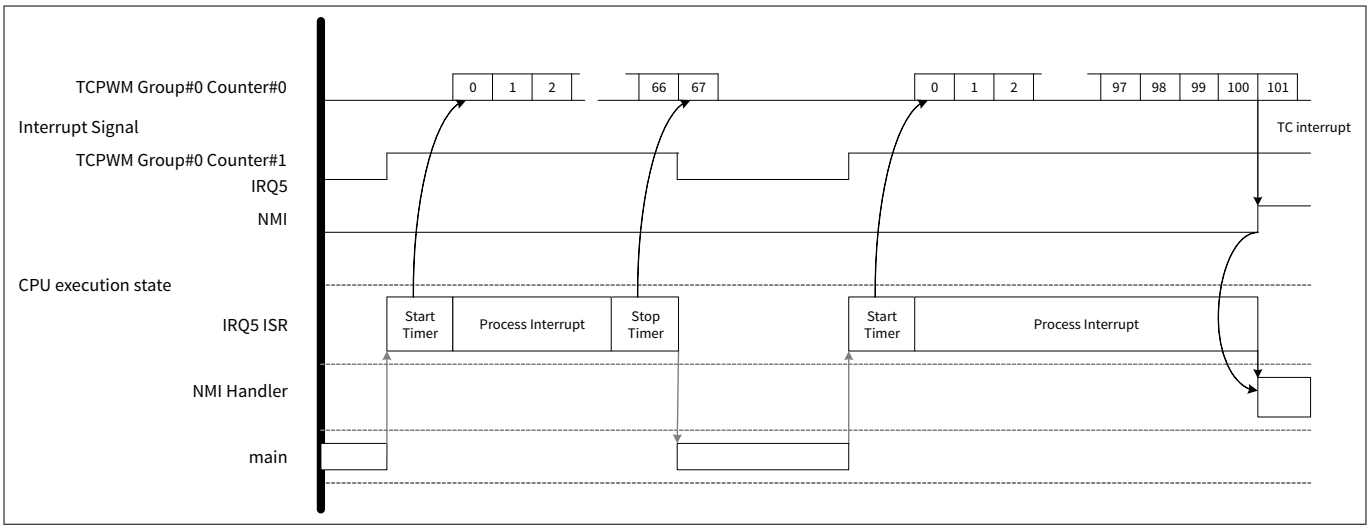

図 10　　タイミング保護の動作例

2.6.2.2　コンフィグレーション

- 初期設定手順

  初期設定手順については、初期設定を参照してください。各周辺機能 (TCPWM Group#0 Counter#0 および #1) を設定後、各周辺機能割込みは CPU 割込みに接続されます。次に、各 CPU 割込みのレベルと許可を NVIC に設定します。割込み接続設定後、各周辺機能を起動します。

図 11 に NMI 生成の割込み初期化について示します。

2 割込みの概要

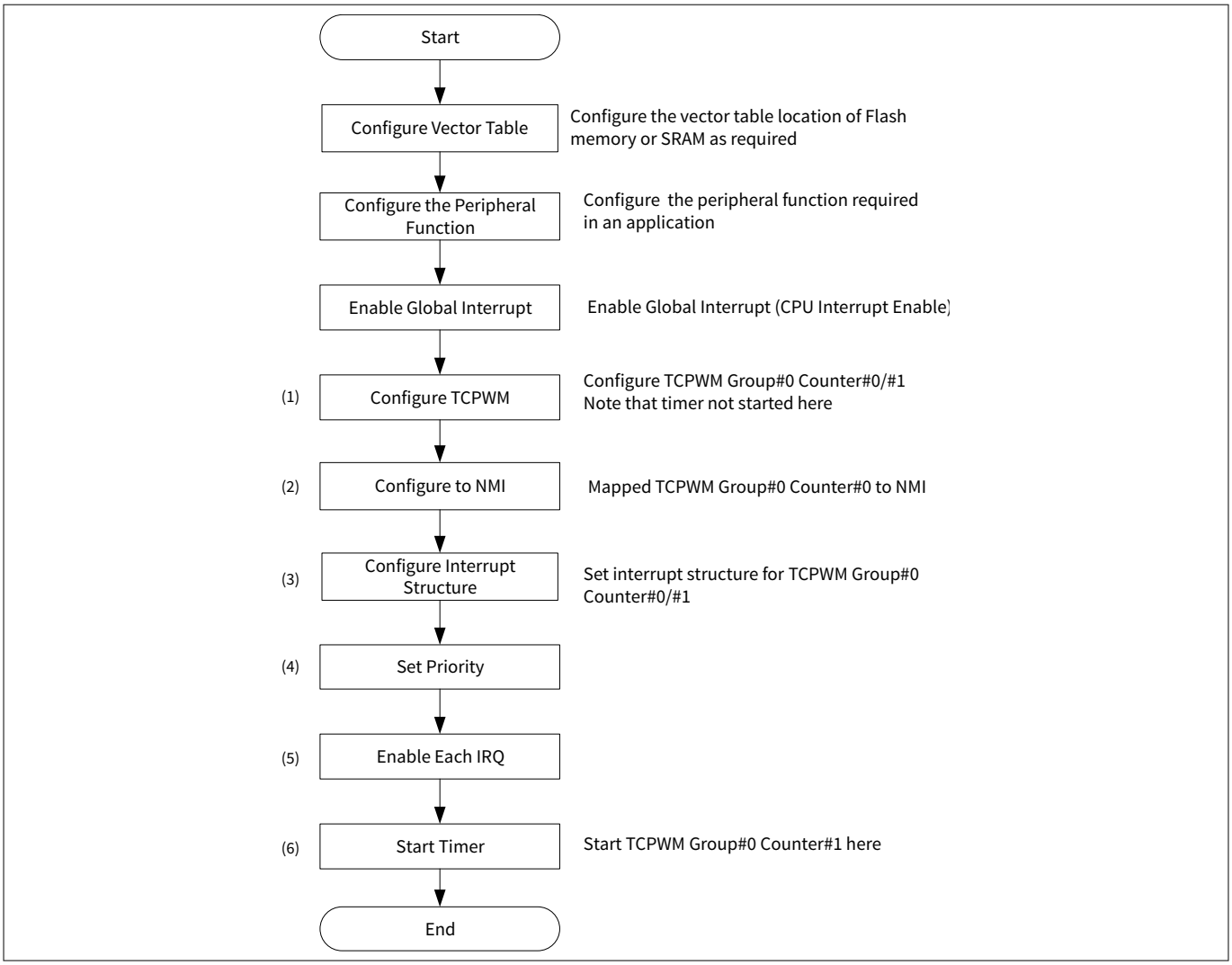

図 11　　NMI 割込み構造初期化設定手順

表 3 および表 4 に、割込み初期化のための SDL 設定部のパラメータと関数を示します。

表 3　　割込み初期化パラメータ

パラメータ	説明	値
irq_cfg_0.sysIntSrc	システム割込みインデックス番号設定 [0: 1022]	tcpwm_0_interrupts_0_IRQn TCPWM Group#0 Counter#0 に割当て (Index number = 274)

(続く)

表 3 (続き) 割込み初期化パラメータ

パラメータ	説明	値
irq_cfg_0.intIdx	CPU 割込み番号設定 [0: 7] CPUIntIdx0_IRQn は IRQ 0 に割当て CPUIntIdx1_IRQn は IRQ 1 に割当て CPUIntIdx2_IRQn は IRQ 2 に割当て CPUIntIdx3_IRQn は IRQ 3 に割当て CPUIntIdx4_IRQn は IRQ 4 に割当て CPUIntIdx5_IRQn は IRQ 5 に割当て CPUIntIdx6_IRQn は IRQ 6 に割当て CPUIntIdx7_IRQn は IRQ 7 に割当て	CPUIntIdx4_IRQn
irq_cfg_0.intIdx.isEnabled	割込みイネーブル設定 False: ディセーブル True: イネーブル	True
irq_cfg_1.sysIntSrc	システム割込みインデックス番号設定 [0: 1022]	tcpwm_0_interrupts_1_IRQn TCPWM Group#0 Counter#1 に割当て (Index number = 275)
irq_cfg_1.intIdx	CPU 割込み番号設定 [0: 7] CPUIntIdx0_IRQn は IRQ 0 に割当て CPUIntIdx1_IRQn は IRQ 1 に割当て CPUIntIdx2_IRQn は IRQ 2 に割当て CPUIntIdx3_IRQn は IRQ 3 に割当て CPUIntIdx4_IRQn は IRQ 4 に割当て CPUIntIdx5_IRQn は IRQ 5 に割当て CPUIntIdx6_IRQn は IRQ 6 に割当て CPUIntIdx7_IRQn は IRQ 7 に割当て	CPUIntIdx5_IRQn
irq_cfg_1.intIdx.isEnabled	割込みイネーブル設定 False: ディセーブル True: イネーブル	True

表 4 割込み初期化関数

関数	説明	値
Cy_SysInt_SetIntSourceNMI (Reg_num, sysintSrc)	NMI レジスタ設定 Reg_num: CM4 NMI 制御レジスタ番号 CPUSS_CM4_NMI_CTLx レジスタ (x = 0 から 3) sysintSrc: システム割込みインデックス番号	Reg_nim: 0 CPUSS_CM4_NMI_CTL0 レジスタ使用 sysIntSrc: tcpwm_0_interrupts_0_IRQn
Cy_SysInt_InitIRQ(irq_cfg)	割込み構造を設定 Irq_cfg: 割込み構造パラメータアドレス	&irq_cfg0/1

(続く)

表 4 (続き) 割込み初期化関数

関数	説明	値
Cy_SysInt_SetSystemIrqVector (sysintSrc, Handler)	システム割込みハンドラをベクタテーブルに設定 sysintSrc: システム割込みインデックス番号 Handler: 割込みハンドラアドレス	sysintSrc: irq_cfg_0/1.sysIntSrc Handler: NMI_Handler / Counter_Handler Code Listing 11 と Code Listing 13 を参照してください。
NVIC_SetPriority(intSrc, intPriority)	割込み優先順位設定: intSrc: CPU 割込み番号 intPriority: 割込み優先順位レベル	intSrc: irq_cfg_1.intIdx intPriority: 3ul
NVIC_EnableIRQ(intSrc)	割込みイネーブル設定 intSrc: CPU 割込み番号	irq_cfg_0/1.intIdx

Code Listing 8 と Code Listing 9 に、割込み設定部分のプログラム例を示します。TCPWM 設定の詳細は、アーキテクチャ TRM [2]の “Timer, Counter, and PWM”章およびアプリケーションノート [3]を参照してください。

2 割込みの概要

Code Listing 8 割込みの設定例 1

```
#define COMPARE0_NUM (100ul)   /* Counter period parameter definition */
#define COMPARE1_NUM (500ul)

cy_stc_tcpwm_counter_config_t MyCounter_config =   /* TCPWM configuration  */
{
    .period             = 0xFFFFul,
    .clockPrescaler     = CY_TCPWM_COUNTER_PRESCALER_DIVBY_4,
    .runMode            = CY_TCPWM_COUNTER_CONTINUOUS,
    .countDirection     = CY_TCPWM_COUNTER_COUNT_UP,
    .debug_pause        = false,
    .CompareOrCapture   = CY_TCPWM_COUNTER_MODE_COMPARE,      /* Compare value  */
    .compare0           = COMPARE0_NUM,                       // Set the comp value
    .compare0_buff      = 0ul,                                // do not care
    .compare1           = 0ul,                                // do not care
    .compare1_buff      = 0ul,                                // do not care
    .enableCompare0Swap = false,                              // do not care
    .enableCompare1Swap = false,                              // do not care
    .interruptSources   = 0ul,                                // do not care
    .capture0InputMode  = 3ul,                                // do not care
    .capture0Input      = 0ul,                                // do not care
    .reloadInputMode    = 3ul,                                // do not care
    .reloadInput        = 0ul,                                // do not care
    .startInputMode     = 3ul,                                // do not care
    .startInput         = 0ul,                                // do not care
    .stopInputMode      = 3ul,                                // do not care
    .stopInput          = 0ul,                                // do not care
    .capture1InputMode  = 3ul,                                // do not care
    .capture1Input      = 0ul,                                // do not care
    .countInputMode     = 3ul,                                // do not care
    .countInput         = 1ul,                                // do not care
    .trigger1           = CY_TCPWM_COUNTER_OVERFLOW,          // do not care
};

      /* Configure interrupt structure parameters for TCPWM Group#0 counter#0  */
cy_stc_sysint_irq_t irq_cfg_0 =
{
    .sysIntSrc  = tcpwm_0_interrupts_0_IRQn,
    .intIdx     = CPUIntIdx4_IRQn,
    .isEnabled  = true,
};
      /* Configure interrupt structure parameters for TCPWM Group#0 counter#1  */
cy_stc_sysint_irq_t irq_cfg_1 =
{
    .sysIntSrc  = tcpwm_0_interrupts_1_IRQn,
    .intIdx     = CPUIntIdx5_IRQn,
    .isEnabled  = true,
};
```

Code Listing 9 割込みの設定例 2

```
/* System interrupt handler */
void Counter_Handler(void)
{
:
}

void NMI_Handler(void)
{
:
}

int main(void)
{
    :
    /* Initialize TCPWM0_GRPx_CNTx_COUNTER as Counter & Enable */
    /*  (1) Configure TCPWM  */
    Cy_Tcpwm_Counter_Init(TCPWMx_GRPx_CNTx_COUNTER_0, &MyCounter_config);  /*(1) Configure
TCPWM*/
    MyCounter_config.compare0 = COMPARE1_NUM;    /*  (1) Configure TCPWM  */
    Cy_Tcpwm_Counter_Init(TCPWMx_GRPx_CNTx_COUNTER_1, &MyCounter_config);  /*(1)Configure
TCPWM*/
    Cy_Tcpwm_Counter_Enable(TCPWMx_GRPx_CNTx_COUNTER_0);    /*  (1) Configure TCPWM  */
    Cy_Tcpwm_Counter_Enable(TCPWMx_GRPx_CNTx_COUNTER_1);    /*  (1) Configure TCPWM  */

    /* Enable Interrupt */
    Cy_Tcpwm_Counter_SetCC0_IntrMask(TCPWMx_GRPx_CNTx_COUNTER_0);    /*  (1) Configure TCPWM  */
    Cy_Tcpwm_Counter_SetCC0_IntrMask(TCPWMx_GRPx_CNTx_COUNTER_1);    /*  (1) Configure TCPWM  */

    /* Set NMI mapping of CM4 */
    /* (2) Configure NMI.  See Code Listing 10 */
    Cy_SysInt_SetIntSourceNMI(0ul, tcpwm_0_interrupts_0_IRQn);

    /*  Interrupt setting for TCPWMs  */
    /*  (3) Set interrupt structure. See Code Listing 3 and Code Listing 4 for details of each
function. */
    Cy_SysInt_InitIRQ(&irq_cfg_0);
    /*  (3) Set interrupt structure. See Code Listing 3 and Code Listing 4 for details of each
function. */
    Cy_SysInt_InitIRQ(&irq_cfg_1);
    /*  (3) Set interrupt structure. See Code Listing 3 and Code Listing 4 for details of each
function. */
    Cy_SysInt_SetSystemIrqVector(irq_cfg_0.sysIntSrc, NMI_Handler);    /*  NMI handler */
    /*  (3) Set interrupt structure. See Code Listing 3 and Code Listing 4 for details of each
function. */
    Cy_SysInt_SetSystemIrqVector(irq_cfg_1.sysIntSrc, Counter_Handler);    /*  System interrupt
handler */

    /* Set the Interrupt Priority & Enable the Interrupt */
    NVIC_SetPriority(irq_cfg_1.intIdx, 3ul);    /*  (4) Set priority  */
    NVIC_EnableIRQ(irq_cfg_1.intIdx);    /*  (5) Interrupt Enable  */
```

```
    /* Start TCPWM Group#0 Counter#1 */
    /*  (6) Start TCPWM Group#0 counter#1 timer  */
    Cy_Tcpwm_TriggerStart(TCPWMx_GRPx_CNTx_COUNTER_1);

    for(;;)
    {
    }
}
```

Code Listing 10 Cy_SysInt_SetIntSourceNMI () 機能

```
void Cy_SysInt_SetIntSourceNMI(uint8_t nmiNum, cy_en_intr_t sysIntSrc)
{
     /*  Set the fault structure interrupt to NMI  */
    CPUSS->unCM4_NMI_CTL[nmiNum].stcField.u10SYSTEM_INT_IDX = sysIntSrc;
}
```

- 割込みハンドラ動作

  割込みハンドラの設定については、割込み処理を参照してください。割込み処理が決められた時間内に完了した場合、NMI は発生しません。ただし、割込み処理が決められた時間以内に完了しない場合、NMI が通知され NMI ハンドラが実行されます。

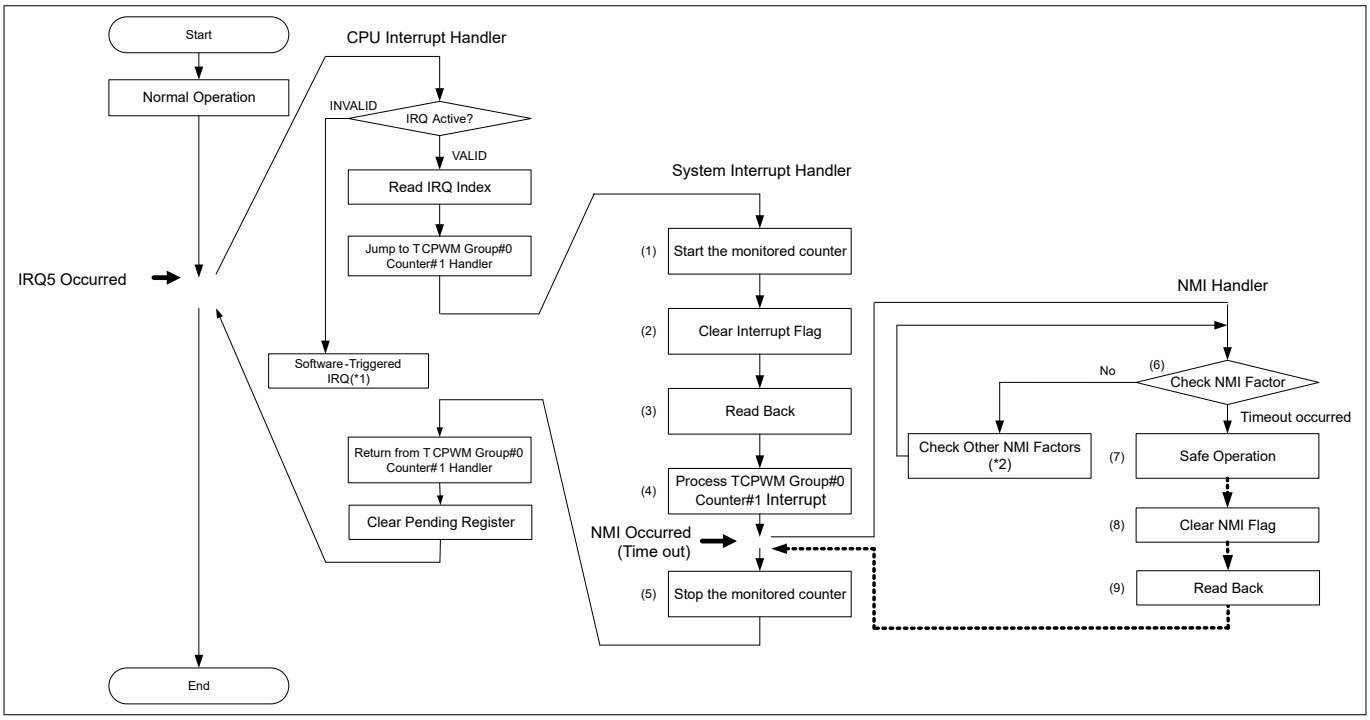

図 12　　タイミング保護違反時の動作例

注:　*(*1) ソフトウェアトリガ割込みレジスタ(STIR) を使用してソフトウェア割込みが発生した場合、SYSTEM_INT_VALID ビットはセットされません。ソフトウェア割込みについてはソフトウェア割込み (CPU レジスタを使用)を参照してください。*

2 割込みの概要

注: *(*2)システムによって複数(4) の NMI 要因がある場合、NMI ハンドラは CPU NMI のソースを特定するために、選択されたすべてのシステム割込みソースの確認が必要な場合があります。*

プログラミングヒント: NMI が発生した場合、リセット発行など適切な安全動作を実行してください。安全動作後、通常動作に復帰できる場合は、割込みから復帰してください。

Code Listing 11 TCPWM group#0 counter#1 のシステム割込みハンドラ

```
void Counter_Handler(void)   /* Interrupt handler registered in the configuration part */
{
     /* Check the source that generated the interrupt  */
    if(Cy_Tcpwm_Counter_GetCC0_IntrMasked(TCPWMx_GRPx_CNTx_COUNTER_1))
    {
        /* Start TCPWM Group#0 Counter#0 */
     /* (1) Start the monitored counter (TCPWM Group#0 Counter#0) */
        Cy_Tcpwm_TriggerStart(TCPWMx_GRPx_CNTx_COUNTER_0);

        /* Clear TCPWM CC0 interrupt flag Group#0 Counter#1 */
     /* (2) Clear the peripheral interrupt flag. See Code Listing 12.  */
        Cy_Tcpwm_Counter_ClearCC0_Intr(TCPWMx_GRPx_CNTx_COUNTER_1);

     /* (4) Interrupt processing */
        /* user program here.. */

        /* Stop TCPWM Group#0 Counter#0 */
     /* (5) Stop the monitored counter (TCPWM Group#0 Counter#0) */
        Cy_Tcpwm_TriggerStopOrKill(TCPWMx_GRPx_CNTx_COUNTER_0);
    }
}
```

Code Listing 12 Cy_Tcpwm_Counter_ClearCC0_Intr() 機能

```
void Cy_Tcpwm_Counter_ClearCC0_Intr(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unINTR.stcField.u1CC0_MATCH = 1ul;      /* (2) Clear interrupt flag */
    ptscTCPWM->unINTR.u32Register;         /* (3) Read back */
}
```

Code Listing 13 TCPWM group#0 counter#0 の NMI ハンドラ

```
void NMI_Handler(void)          /* NMI handler registered in the configuration part */
{
    if(Cy_Tcpwm_Counter_GetCC0_IntrMasked(TCPWMx_GRPx_CNTx_COUNTER_0))  /*(6) Check NMI factor*/
    {
        /* Safe Operation (Counter#0 Stop)*/
         /* (7) Safe Operation (Stop the monitored counter) */
        {
            /* Stop TCPWM Group#0 Counter#0 */
            Cy_Tcpwm_TriggerStopOrKill(TCPWMx_GRPx_CNTx_COUNTER_0);
        }

        /* Clear TCPWM CC0 interrupt flag Group#0 Counter#0 */
         /* (8) Clear the NMI flag, and (9) Read back. See Code Listing 12. */
        Cy_Tcpwm_Counter_ClearCC0_Intr(TCPWMx_GRPx_CNTx_COUNTER_0);
    }
}
```

2.6.3 ウェイクアップ割込み

使用可能なすべての割込みは Avtive モード、Sleep モードからの復帰に使用できます。

システム割込みのサブセットは、DeepSleep モードからの復帰に使用できます。このサブセットは、Active モードそして Sleep または DeepSleep モードからの復帰に使用できます。使用可能な割込みの詳細はデバイスデータシート [1]を参照してください。

ここでは、real-time-clock (RTC) からの割込みであるバックアップドメインからの割込みについて説明し、その設定方法と割込みによって DeepSleep から復帰する動作について説明します。

2.6.3.1 ユースケース

- 割込み設定
  - システム割込みソース: バックアップドメイン (IDX: 12) RTC の ALARM1 割込み
  - CPU 割込みに割当て: IRQ7
  - CPU 割込み優先順位: 6

図 13 に、割込み動作を示します。

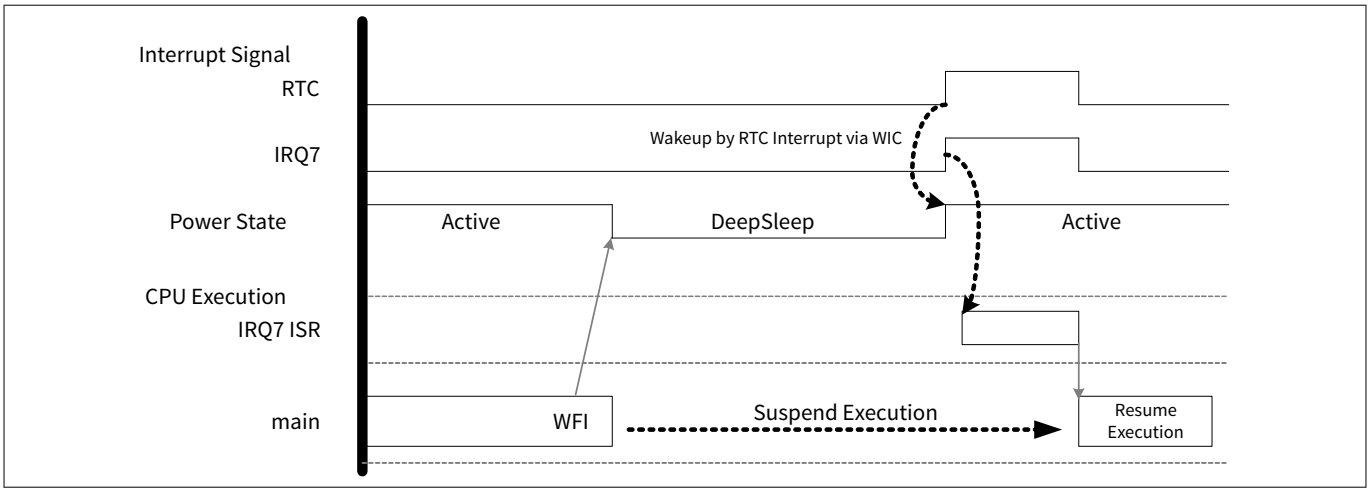

図 13　　ウェイクアップ割込み動作例

CPU が実行する割込み待ち (WFI) は、Sleep または DeepSleep への遷移をトリガします。WFI は、割込みイベントが発生するまでプロセッサの実行を一時中断します。1 つ以上のウェイクアップ割込みが発生すると、WIC はウェイクアップ信号を生成し、CPU を Active モードに遷移させます。

CPU は、Active モードになると ISR を実行し、ISR の復帰後、中断したプログラムを再開します。

2.6.3.2　　コンフィグレーション

- 初期設定手順

  WIC を使用してウェイクアップ割込みを生成する初期設定手順は通常の割込み設定と同じです。初期設定手順については初期設定を、RTC 設定についてはアーキテクチャ TRM [2]の“Real-Time Clock”章を参照してください。
- 割込みハンドラ動作

  割込みハンドラ設定については、割込み処理を参照してください。ウェイクアップ割込みから復帰後、CPU はプログラムの停止位置から再開します。

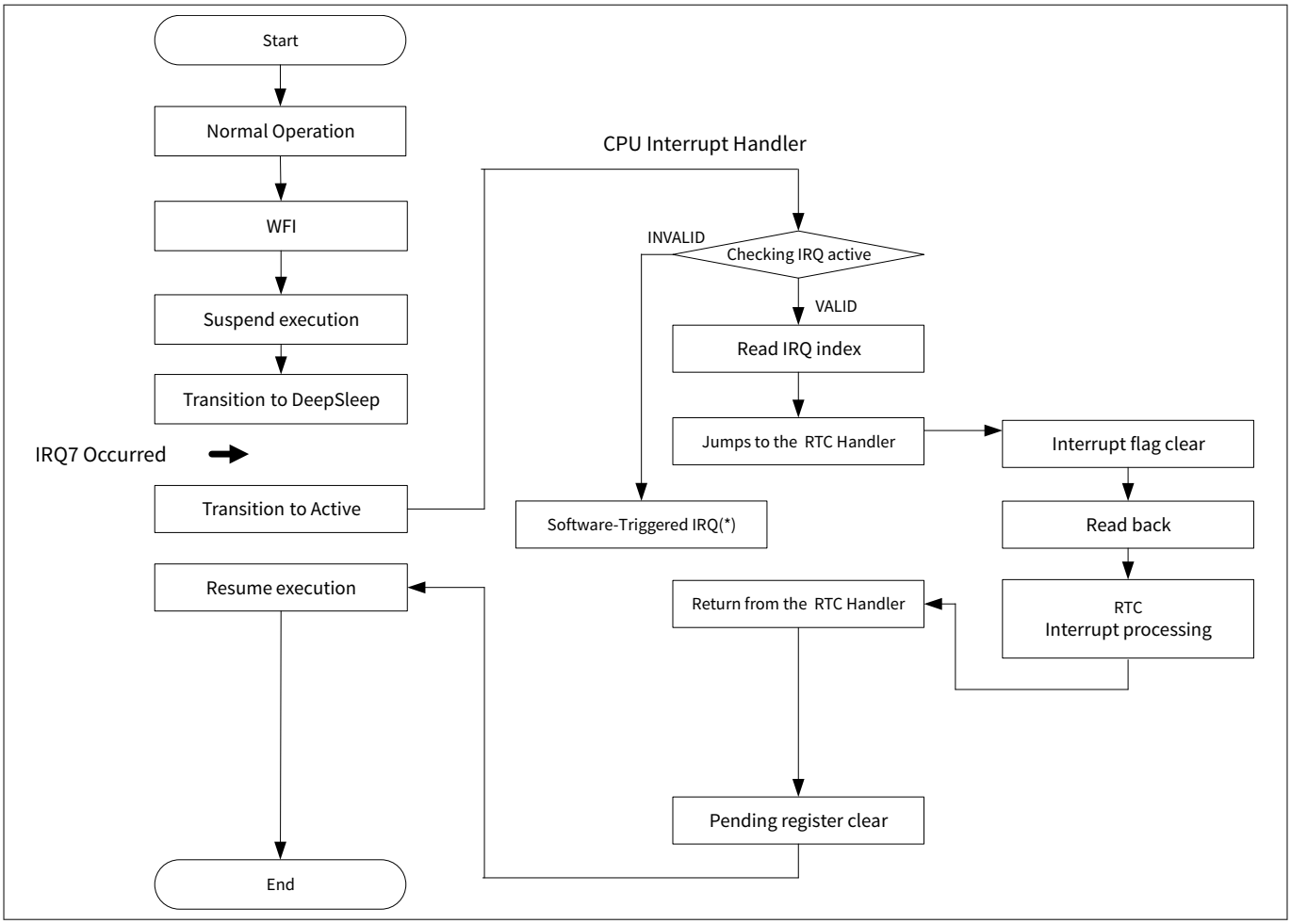

図 14 ウェイクアップ割込み処理例

注: (*) ソフトウェアトリガ割込みレジスタ(STIR) を使用してソフトウェア割込みが発生した場合、SYSTEM_INT_VALID ビットはセットされません。ソフトウェア割込みについてはソフトウェア割込み (CPU レジスタを使用)を参照してください。

2.6.4 ソフトウェア割込み (CPU レジスタを使用)

本シリーズは、IRQ8 から IRQ15 をソフトウェア割込みとして使用できます。ソフトウェア割込みは、対応する CPU 割込み IRQ8 から IRQ15 の ソフトウェアトリガ割込みレジスタ (STIR) に'1'を書込むことにより生成できます。IRQ0 から IRQ7 も STIR でソフトウェア割込みをトリガできます。

ここでは、STIR レジスタを使用したソフトウェア割込みの生成について説明し、設定手順と復帰処理について説明します。

2.6.4.1 ユースケース

- 割込み設定
  - CPU 割込みに割当て: IRQ8
  - CPU 割込み優先順位: 3

図 15 に、この設定の割込み動作を示します。

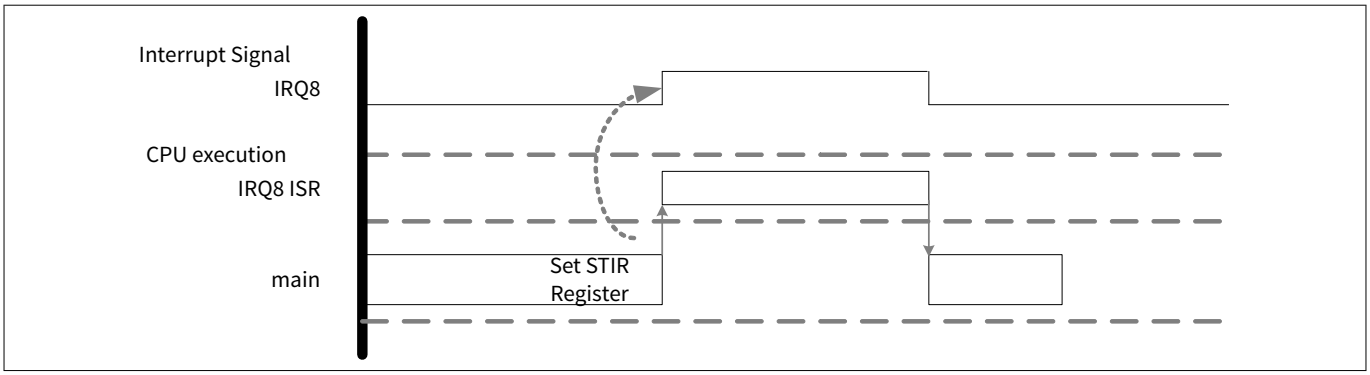

図 15　　ソフトウェア割込み動作

2.6.4.2　　コンフィグレーション

- 初期設定
  ソフトウェア割込みは、割込み構造を使用せず、NVIC のみ設定します。

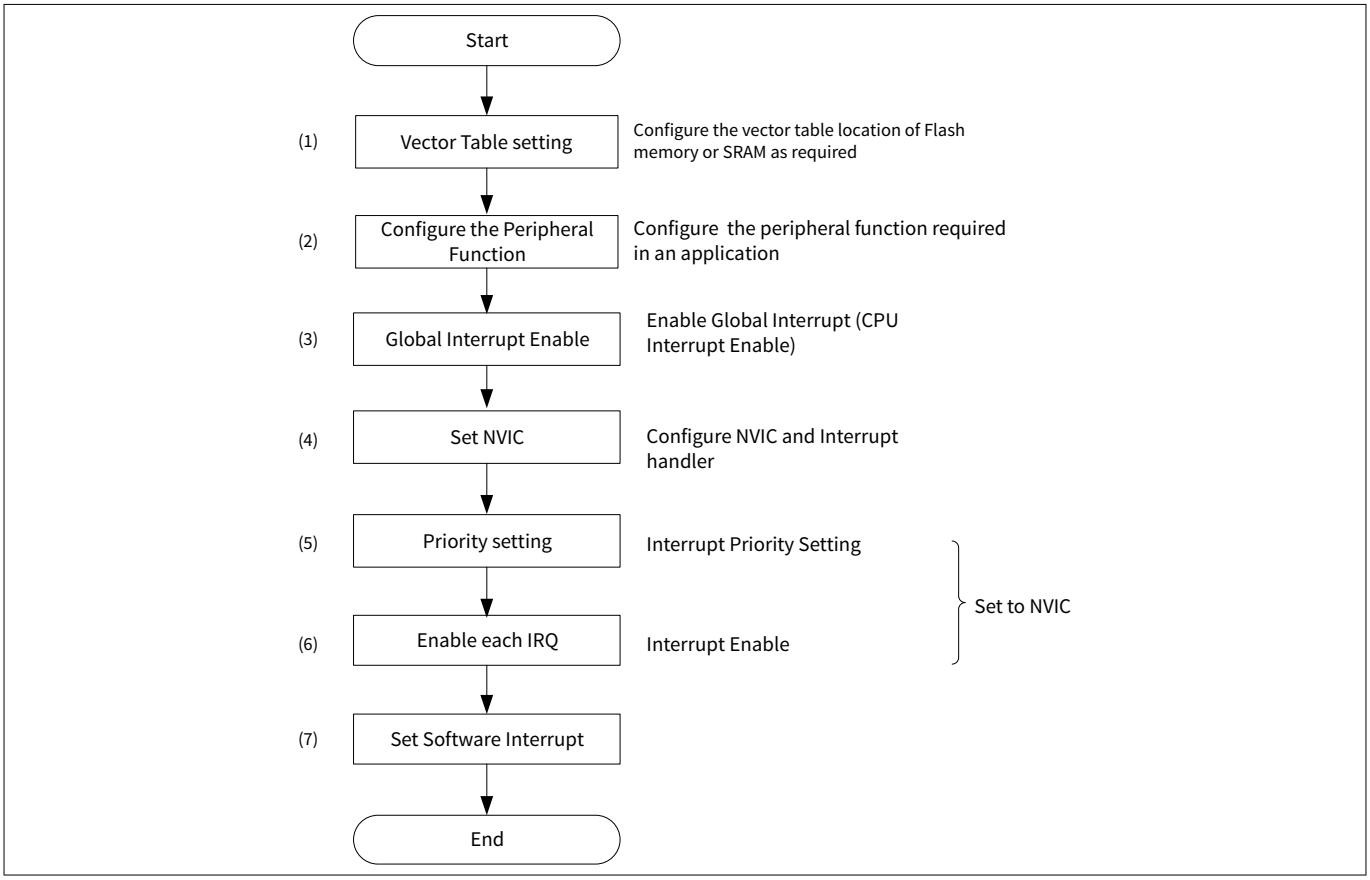

図 16　　ソフトウェア割込みの初期設定手順 (CPU レジスタを使用)

表 5 および表 6 に、割込み初期化のための SDL 設定部のパラメータと関数を示します。

2 割込みの概要

表 5　　割込み初期化パラメータ

パラメータ	説明	値
cfg.intrSrc	CPU 割込み設定 [8: 15] Internal0_IRQn は IRQ 8 に割当て Internal1_IRQn は IRQ 9 に割当て Internal2_IRQn は IRQ 10 に割当て Internal3_IRQn は IRQ 11 に割当て Internal4_IRQn は IRQ 12 に割当て Internal5_IRQn は IRQ 13 に割当て Internal6_IRQn は IRQ 14 に割当て Internal7_IRQn は IRQ 15 に割当て	Internal0_IRQn
cfg.intrPriority	割込み優先順位設定	3ul

表 6　　割込み初期化関数

関数	説明	値
Cy_SysInt_Init (cfg, Handler)	割込み構造を設定 cfg: ソフトウェア割込みパラメータアドレス Handler: 割込みハンドラアドレス	cfg: cfg address Handler: SoftwareInterupt0Handler, Code Listing 18 を参照してください。
NVIC_SetPriority(intrSrc, intPriority)	割込み優先順位設定: intrSrc: CPU 割込み番号 intPriority: 割込み優先順位レベル	intrSrc: Internal0_IRQn intPriority: 3ul
NVIC_EnableIRQ(intSrc)	割込みイネーブル設定 intSrc: CPU 割込み番号	Internal0_IRQn
Cy_SysInt_SoftwareTrig(intSrc)	ソフトウェア割込み設定 intSrc: CPU 割込み番号	Internal0_IRQn

Code Listing 14 にソフトウェア割込みの割込み設定部のプログラム例を示します。(1)のコンフィグレーションについては初期設定を参照してください。

2 割込みの概要

Code Listing 14 ソフトウェア割込みの設定例

```
void SoftwareInterupt0Handler(void)   /*  System interrupt handler */
{
    __NOP();
}

int main(void)
{
    SystemInit();   /*  (2) Peripheral function setting */

    __enable_irq(); /* Enable global interrupts. */    /*  (3) Enable global interrupt */

    /* Software Interrupt 0 */
    {
        /* At first change software interrupt 0 handler */
        /* The default handler is defined at startup_tviibe1m_cm4 */
        cy_stc_sysint_t cfg =
        {
            .intrSrc = Internal0_IRQn,   /*  Set configuration parameters for NVIC */
            .intrPriority = 3ul,
        };
       /*  (4) Set NVIC registers and interrupt handler. See Code Listing 15. */
        Cy_SysInt_Init(&cfg, SoftwareInterupt0Handler);

        /* Enable software interrupt IRQ */
        NVIC_EnableIRQ(Internal0_IRQn);
        /*  (6) Interrupt Enable  */
        /* Force set pending status in the NVIC */
       /*  (7) Set software interrupt.  See Code Listing 17. */
       Cy_SysInt_SoftwareTrig(Internal0_IRQn);
    }

    for(;;)
    {
    }
}
```

Code Listing 15 Cy_SysInt_Init() 機能

```
cy_en_sysint_status_t Cy_SysInt_Init(const cy_stc_sysint_t * config, cy_israddress userIsr)
{
    cy_en_sysint_status_t status = CY_SYSINT_SUCCESS;

    if(NULL != config)   /*  Check if configuration parameter values are valid */
    {
        /* Only set the new vector if it is CPU interrupt (not internal SW interrupt) */
        if ((config->intrSrc < CPUIntIdx0_IRQn) || (config->intrSrc >= Internal0_IRQn))
        {
            /* Only set the new vector if it was moved to __ramVectors */
            if (SCB->VTOR == (uint32_t)&__ramVectors)
            {
                / * Set the software interrupt handler. See Code Listing 16. */
                (void)Cy_SysInt_SetVector(config->intrSrc, userIsr);
            }
        }
        else
        {
            return CY_SYSINT_BAD_PARAM; // Vector of system handler have to be set at ROM table.
        }
        NVIC_SetPriority(config->intrSrc, config->intrPriority);   /*  (5) Set priority */
    }
    else
    {
        status = CY_SYSINT_BAD_PARAM;
    }

    return(status);
}
```

Code Listing 16 Cy_SysInt_SetVector() 機能

```
void  Cy_SysInt_SetVector(IRQn_Type intrSrc, cy_israddress userIsr)
{
    /* Only set the new vector if it was moved to __ramVectors */
    if (SCB->VTOR == (uint32_t)&__ramVectors)
    {
        __ramVectors[CY_INT_IRQ_BASE + intrSrc] = userIsr;   /*  Set the interrupt handler  */
    }
}
```

Code Listing 17 Cy_SysInt_SoftwareTrig() 機能

```
__STATIC_INLINE void Cy_SysInt_SoftwareTrig(IRQn_Type intrSrc)
{
    NVIC->STIR = intrSrc & CY_SYSINT_STIR_MASK;      /*  Set STIR  */
}
```

- 割込み処理

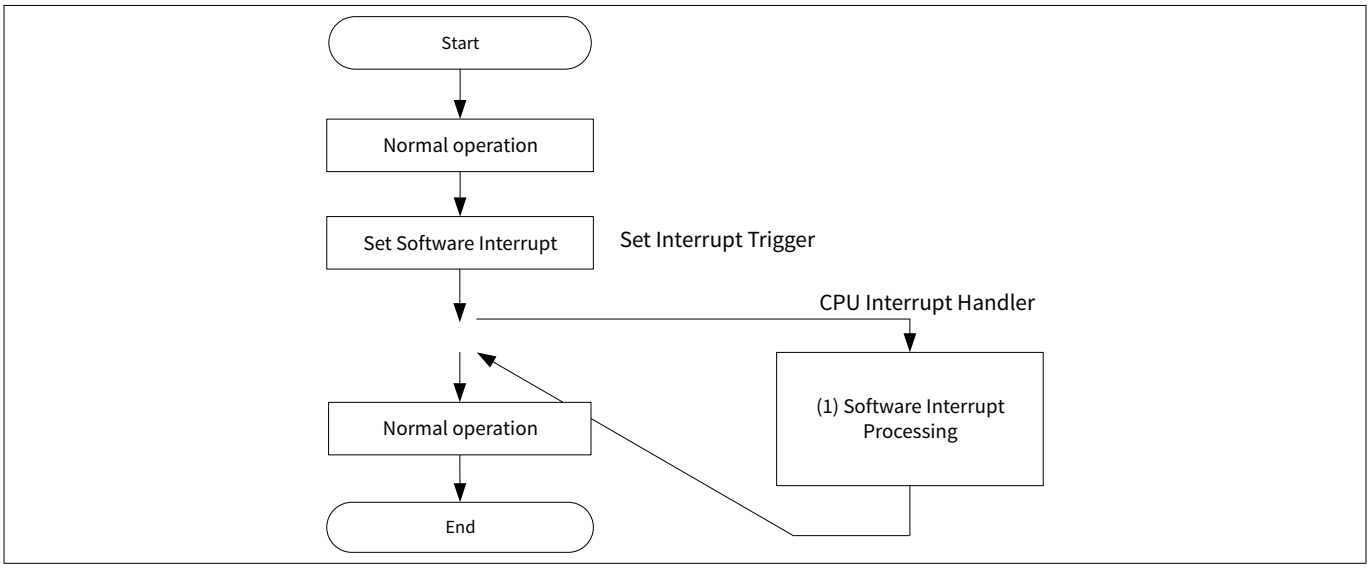

図 17　　ソフトウェア割込み処理フロー例

ペンディングフラグは、ハンドラの遷移によってクリアされます。したがって、セットペンディングレジスタを使用してソフトウェア割込みを生成する場合、割込みハンドラからの復帰時にペンディングフラグをクリアする必要はありません。

Code Listing 18 にソフトウェア割込みでの CPU 割込みハンドラ例を示します。

Code Listing 18 ソフトウェア割込みでの CPU 割込みハンドラ

```
void SoftwareInterupt0Handler(void)
{
    __NOP();      /*  Software interrupt processing  */

}
```

2.6.5　　ソフトウェア割込み (周辺機能を使用)

本シリーズは未使用の周辺機能を利用してソフトウェア割込みを発生できます。周辺機能のソフトウェア割込みは、周辺機能割込み構造の INTR_SET レジスタを使用します。ソフトウェアは、INTR_SET に '1'を書込むことで INTR レジスタの対応するビットをセットします。

ここでは、周辺機能を使用したソフトウェア割込み生成について説明し、設定手順と復帰処理について示します。割込みは CM4 によって処理されます。

2.6.5.1 ユースケース

- 割込み設定
  - システム割込みソース: TCPWM Group#0 Counter#1(IDX: 275) TC 割込み
  - CPU 割込みへ割当て: IRQ4
  - CPU 割込み優先順位: 3

このタイプの割込みでは、TCPWM Group#0 Counter#1 のタイマ機能は使用されません。

図 18 に、この設定の割込み動作を示します。

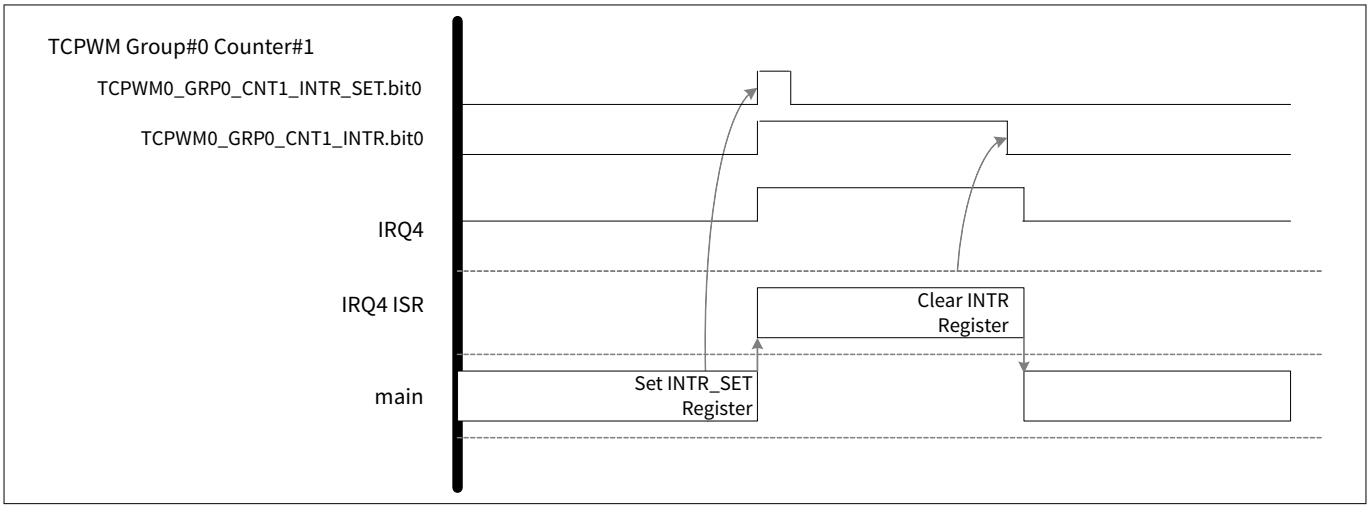

図 18 未使用 TCPWM を利用したソフトウェア割込み動作例

2.6.5.2 コンフィグレーション

- 初期設定手順

  初期設定手順については、初期設定を参照してください。周辺機能 (TCPWM Group#0 Counter#1) の設定後、周辺機能割込みは、CPU 割込みに接続されます。次に、各 CPU 割込みのレベルと許可を NVIC に設定します。割込み接続設定後、各周辺機能を起動します。TCPWM の設定については、アーキテクチャ TRM [2] の “Timer, Counter, and PWM”章を参照してください。

2 割込みの概要

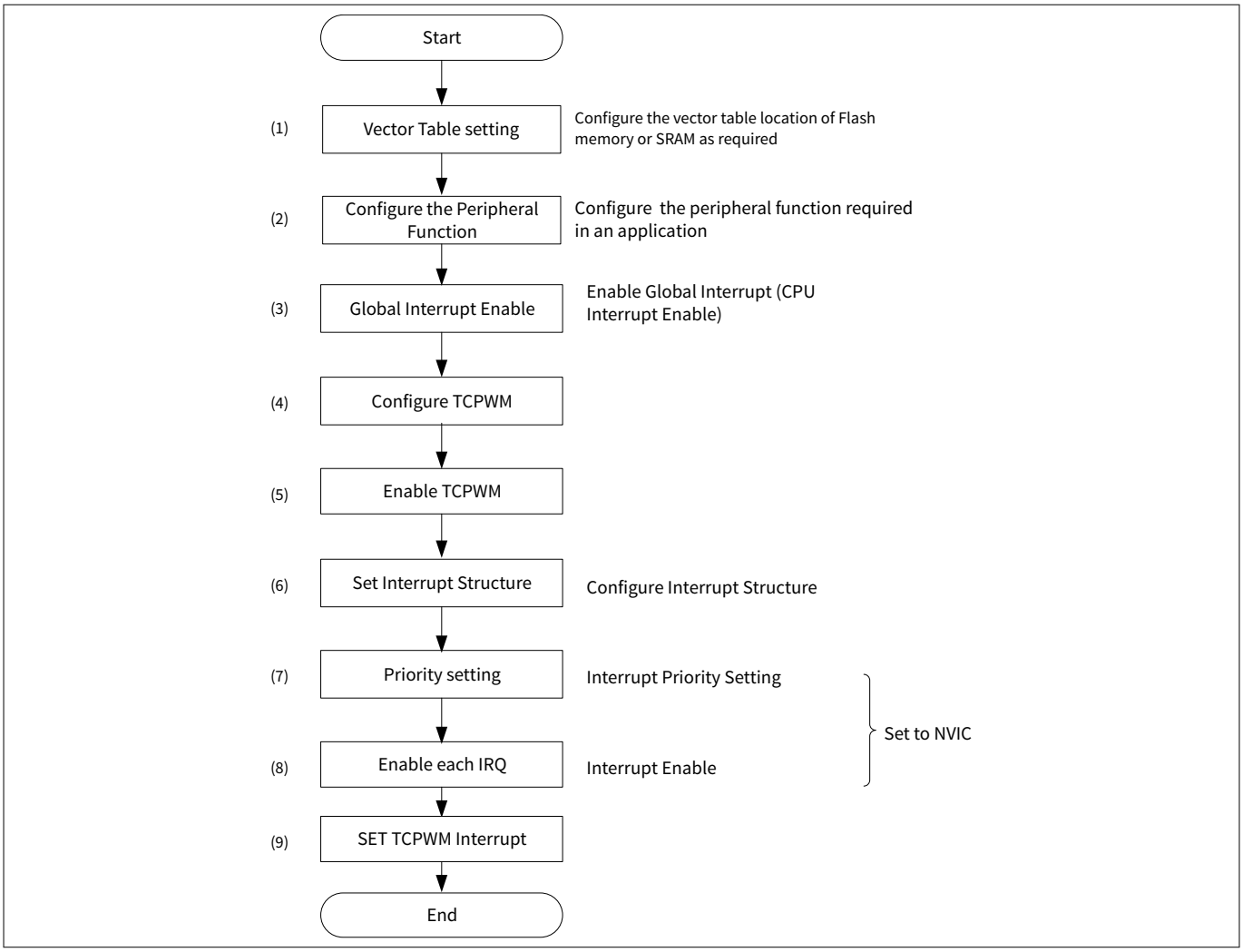

図 19　　ソフトウェア割込み初期設定手順 (周辺機能を使用)

表 7 および表 8 に、割込み初期化のための SDL 設定部のパラメータと関数を示します。

表 7　　割込み初期化パラメータ

パラメータ	説明	値
irq_cfg_1.sysIntSrc	システム割込みインデックス番号設定 [0: 1022]	tcpwm_0_interrupts_1_IRQn TCPWM Group#0 Counter#1 に割当て (インデックス番号 = 275)

(続く)

2 割込みの概要

表 7 (続き) 割込み初期化パラメータ

パラメータ	説明	値
irq_cfg_1.intIdx	CPU 割込み設定 [0: 7] CPUIntIdx0_IRQn は IRQ 0 に割当て CPUIntIdx1_IRQn は IRQ 1 に割当て CPUIntIdx2_IRQn は IRQ 2 に割当て CPUIntIdx3_IRQn は IRQ 3 に割当て CPUIntIdx4_IRQn は IRQ 4 に割当て CPUIntIdx5_IRQn は IRQ 5 に割当て CPUIntIdx6_IRQn は IRQ 6 に割当て CPUIntIdx7_IRQn は IRQ 7 に割当て	CPUIntIdx4_IRQn
irq_cfg_1.isEnabled	割込みイネーブル設定 False: ディセーブル True: イネーブル	True

表 8 割込み初期化関数

関数	説明	値
Cy_Tcpwm_Counter_SetTC_IntrMask (Count_num)	TCPWM 割込みイネーブル Count_num: 割込み有効タイマカウンタ番号	TCPWMx_GRPx_CNTx_COUNTER_1 Counter#1 に割当て
Cy_SysInt_InitIRQ(irq_cfg)	割込みを設定 Irq_cfg: 割込み設定パラメータアドレス	&irq_cfg_1
Cy_SysInt_SetSystemIrqVector (sysintSrc, Handler)	システム割込みハンドラをベクタテーブルに設定 sysintSrc: システム割込みインデックス番号 Handler: 割込みハンドラアドレス	sysintSrc: irq_cfg_1.sysIntSrc Handler: Counter_Handler、Code Listing 21 を参照してください。
NVIC_SetPriority(intrSrc, intPriority)	割込み優先順位設定: intrSrc: CPU 割込み番号 intPriority: 割込み優先順位レベル	intrSrc: irq_cfg_1.intIdx intPriority: 3ul
NVIC_EnableIRQ(intSrc)	割込みイネーブル設定 intSrc: CPU 割込み番号	intrSrc: irq_cfg_1.intIdx

Code Listing 19 にソフトウェア割込みの割込み設定部のプログラム例を示します。(1)のコンフィグレーションについては初期設定を参照してください。

2 割込みの概要

Code Listing 19 割込みの設定例

```
   / * Interrupt structure configuration parameters */
cy_stc_sysint_irq_t irq_cfg_1 =
{
    .sysIntSrc  = tcpwm_0_interrupts_1_IRQn,
    .intIdx     = CPUIntIdx4_IRQn,
    .isEnabled  = true,
};

void Counter_Handler(void)   /* System interrupt handler */
{
:
}

int main(void)
{
    SystemInit();   / * (2) Peripheral function setting */

   / * (3) Enable global interrupt */
    __enable_irq();    /* Enable global interrupts. */

   / * (4) Configure the TCPWM. See Architecture TRM  [2]. */
    /* Configure TCPWM Group#0 Counter#1 */
:

    /* Enable Interrupt */
   / * (5) Enable the TCPWM counter#1 interrupt.  See Code Listing 20. */
    Cy_Tcpwm_Counter_SetTC_IntrMask(TCPWMx_GRPx_CNTx_COUNTER_1);

    /*  Interrupt setting for TCPWMs  */
    Cy_SysInt_InitIRQ(&irq_cfg_1);
   / * (6) Configure interrupt structure */
    Cy_SysInt_SetSystemIrqVector(irq_cfg_1.sysIntSrc, Counter_Handler);

    /* Set the Interrupt Priority & Enable the Interrupt */
   / * (7) Set priority (*1) */
    NVIC_SetPriority(irq_cfg_1.intIdx, 3ul);
   / * (8) Interrupt Enable (*1) */
    NVIC_EnableIRQ(irq_cfg_1.intIdx);

    /* Set INTR_SET register, TCPWM Group#0 Counter#1 */
   / * (9) Set TCPWM Interrupt */
    TCPWMx_GRPx_CNTx_COUNTER_1->unINTR_SET.u32Register = 0x1ul;

    for(;;)
    {
    }
}
```

Code Listing 20 Cy_Tcpwm_Counter_SetTC_IntrMask () 機能

```
void Cy_Tcpwm_Counter_SetTC_IntrMask(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unINTR_MASK.stcField.u1TC = 1ul;     /*  Interrupt enable  */
}
```

- 割込み処理

  割込みは、ソフトウェアによってトリガされます。ただし、割込み処理は周辺割込みの処理と同じです。割込みハンドラの設定については、割込み処理を参照してください。

  割込みが発生すると、CPU はベクタテーブルで指定された割込みハンドラにジャンプします。CPU 割込みハンドラ内で CPU 割込みをトリガしたシステム割込みを識別し、対応する ISR にジャンプします。CPU 割込みハンドラのシステム割込みフラグをクリアし、ISR を実行後、関連する CPU ペンディングレジスタビットをクリアしてメインルーチンに復帰します。

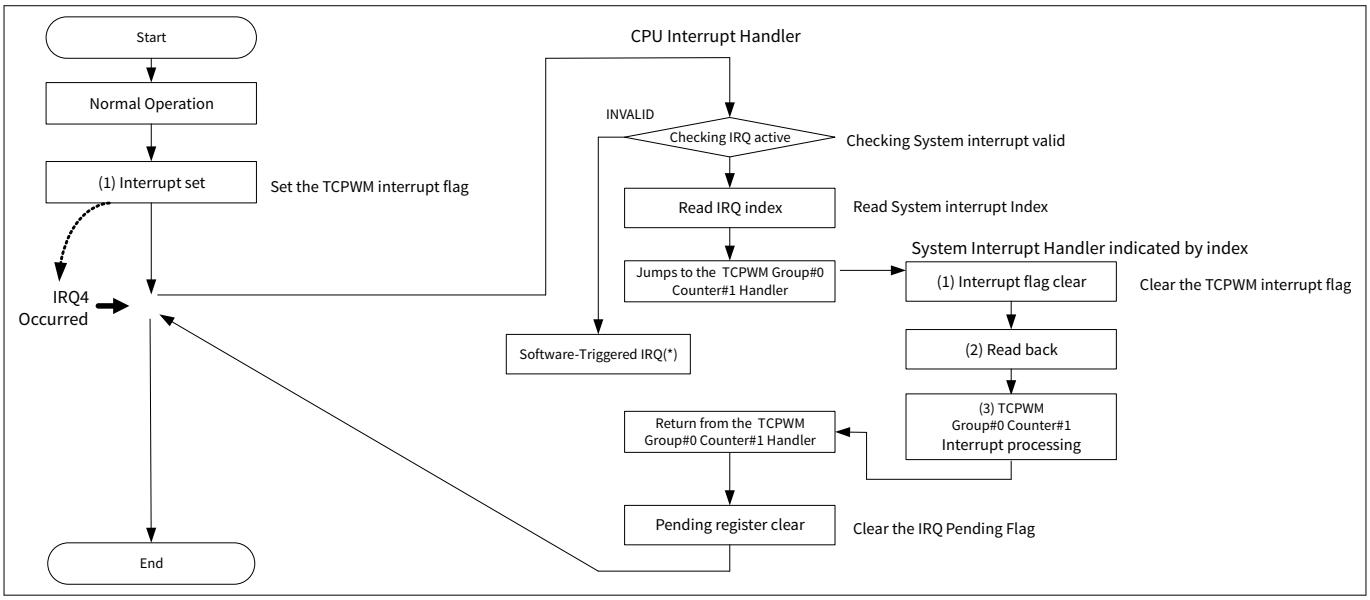

図 20　　未使用 TCPWM を利用したソフトウェア割込み処理

***注**:　(*)ソフトウェアトリガ割込みレジスタ(STIR) を使用してソフトウェア割込みが発生した場合、SYSTEM_INT_VALID ビットはセットされません。ソフトウェア割込みについてはソフトウェア割込み(CPU レジスタを使用)を参照してください。*

Code Listing 21 にシステム割込みハンドラの例を示します。CPU 割込みハンドラについては Code Listing 5 を参照してください。

Code Listing 21 システム割込みハンドラの例

```
    /*  Interrupt handler registered in the configuration part  */
void Counter_Handler(void)
{
    /*  Check the source that generated the interrupt */
    if(Cy_Tcpwm_Counter_GetTC_IntrMasked(TCPWMx_GRPx_CNTx_COUNTER_1))
    {
        /* Clear TCPWM TC interrupt flag Group#0 Counter#1 */
    /*  Clear the peripheral interrupt flag that became the interrupt source. See Code Listing
22 */
        Cy_Tcpwm_Counter_ClearTC_Intr(TCPWMx_GRPx_CNTx_COUNTER_1);

        /*  (3) Interrupt Processing.  */
        /* User program here.. */

    }
}
```

Code Listing 22 Cy_Tcpwm_Counter_ClearTC_Intr()

```
void Cy_Tcpwm_Counter_ClearTC_Intr(volatile stc_TCPWM_GRP_CNT_t *ptscTCPWM)
{
    ptscTCPWM->unINTR.stcField.u1TC = 1ul;    /*  (1) Clear interrupt flag   */
    ptscTCPWM->unINTR.u32Register;           /*  (2) Read back   */
}
```

3 Fault report structure 概要

3.1 Fault report structure

Fault report structure は、ソフトウェア実行中に発生する一時的な故障に関する情報を集約します。また、故障に関する情報を格納し、リセットを発生できます。この集中型故障通知構造によって、システム全体にわたって以下のように故障の一貫した処理が可能なため、ソフトウェア開発が容易になります。単一の故障割込みハンドラのみ必要です。

- 単一のレジスタセットから故障要因と故障特有の追加情報を提供します。
- つまり、故障要因と障害情報の取得のために複数の検索は必要ありません。
- すべての保留中の故障情報は単一のレジスタセットから読み出せます。

Fault report structure は、MPU/SMPU/PPU 保護違反、ECC エラーなどのメモリ固有の故障や周辺機能固有故障などの障害をキャプチャします。Fault report structure は、故障情報のみをキャプチャします。

図 21 に fault report structure のブロックダイヤグラムを示します。

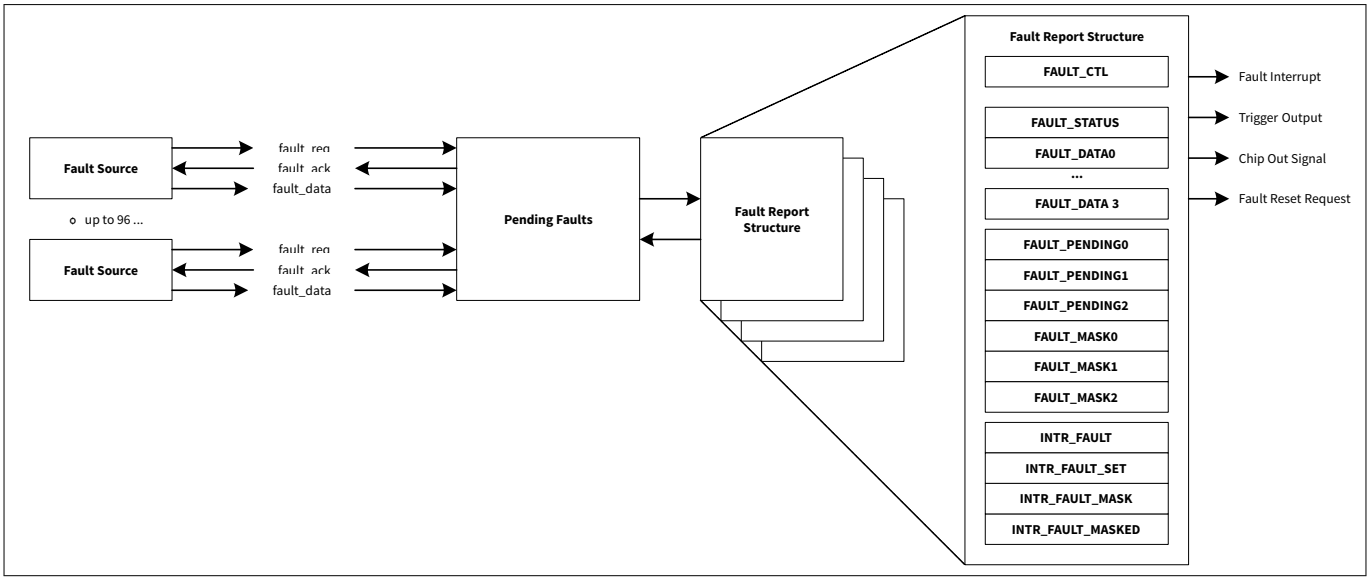

図 21 Fault report structure

本シリーズは、4 つの Fault report structure があります。各 Fault report structure は専用の制御レジスタとステータスレジスタのレジスタセットがあり、単一の故障をキャプチャします。

Fault report structure は故障をキャプチャすると、以下の情報を格納します。

- 故障をキャプチャしたことを示すフラグ
- 故障要因を特定するインデックス
- 故障詳細を示す追加情報

Fault report structures によってキャプチャ可能な故障と追加情報については、アーキテクチャ TRM [2]を参照してください。

Fault report structure は、故障をキャプチャすると、以下の信号を生成できます。

- 故障割込み
- トリガ出力
- チップ出力
- 故障リセット要求

各信号出力は、レジスタによって許可または禁止できます。故障割込みは、システム割込みとして CPU に通知できます。

保留中の故障では、Fault report structure によってキャプチャされていない故障が保持されます。故障が Fault report structure によってキャプチャされると、関連する故障保留ビットは‘0’にクリアされます。Fault report structure は Active または Sleep 電源モードで使用できます。

3.2 初期設定手順

ここではサンプルドライバライブラリ (SDL) を使用して、割込みの初期設定を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。SDL 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。

図 22 に、Fault report structure の初期設定手順を示します。Fault report structure を初期化し、故障の接続、および故障時の動作を設定します。

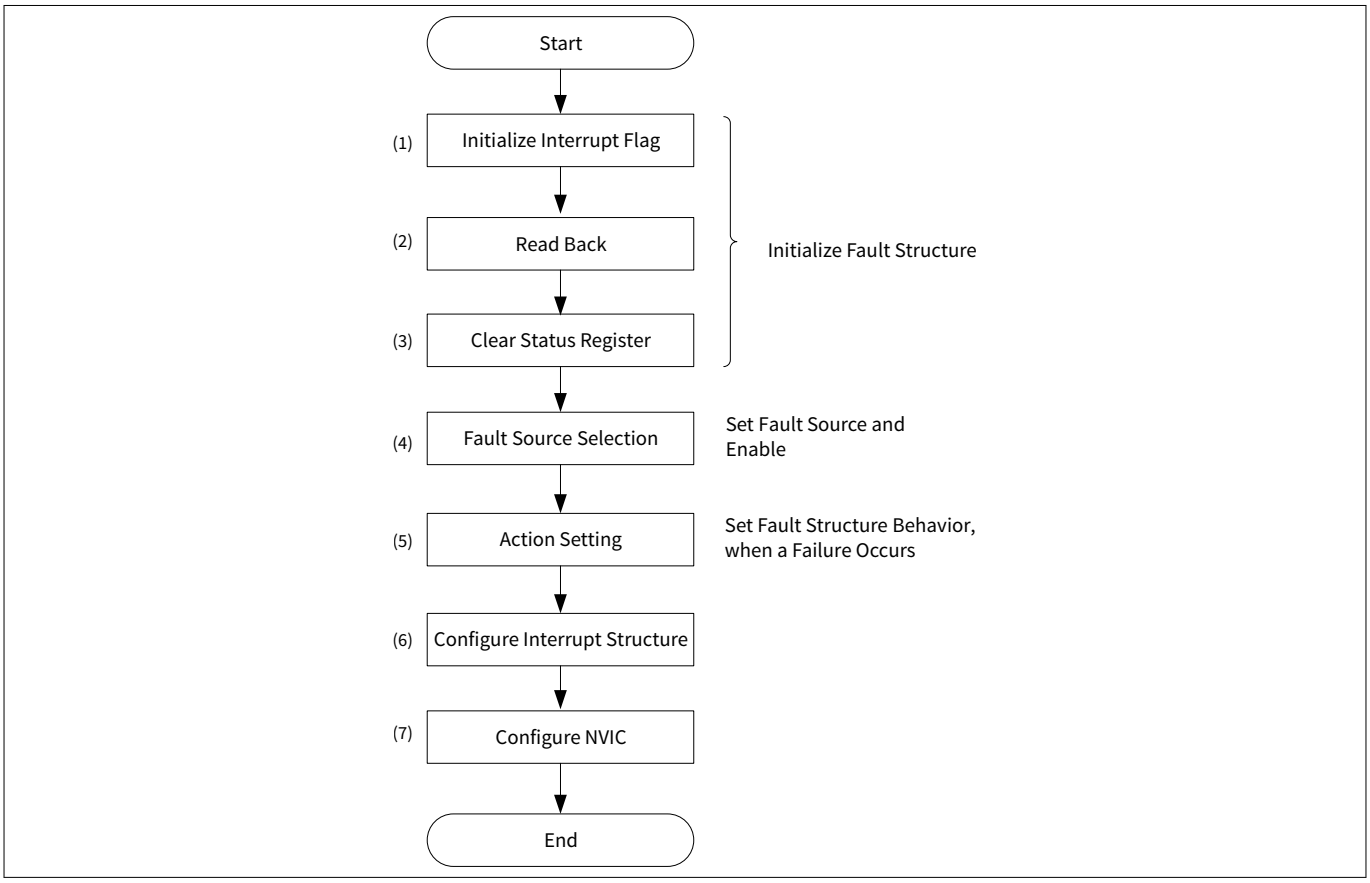

図 22 Fault report structure 初期設定フロー

3.2.1 ユースケース

ここでは、SRAM0 で修正可能な ECC 違反が検出されると、割込みコントローラを介して CPU に通知されます。

ここで使用される IDX 番号は、システム割込みインデックス番号を示します。IDX 番号は、CYT2B シリーズのものです。実際に使用するインデックス番号についてはデバイスデータシート [1]を参照してください。

- 使用する Fault report structure: #0
- 故障要因: CPUSS_RAM0_C_ECC (IDX: 58)
- 故障時の動作: 割込み生成

- CPU 割込みへ割当て: IRQ2
- CPU 割込み優先順位: 0

3.2.2 コンフィグレーション

表 9 および表 10 に、Fault report structure 初期化のための SDL 設定部のパラメータと関数を示します。

表 9 Fault report structure 初期化パラメータ

パラメータ	説明	値
irq_cfg.sysIntSrc	システム割込みインデックス番号設定 [0: 1022]	cpuss_interrupts_fault_0_IRQn Fault structure #0 に割当て (インデックス番号 = 8)
irq_cfg.intIdx	CPU 割込み番号設定 [0: 7] CPUIntIdx0_IRQn は IRQ 0 に割当て CPUIntIdx1_IRQn は IRQ 1 に割当て CPUIntIdx2_IRQn は IRQ 2 に割当て CPUIntIdx3_IRQn は IRQ 3 に割当て CPUIntIdx4_IRQn は IRQ 4 に割当て CPUIntIdx5_IRQn は IRQ 5 に割当て CPUIntIdx6_IRQn は IRQ 6 に割当て CPUIntIdx7_IRQn は IRQ 7 に割当て	CPUIntIdx2_IRQn
irq_cfg.isEnabled	割込みイネーブル設定 False: ディセーブル True: イネーブル	True
tFlt_Temp.ResetEnable	リセット要求設定 False: ディセーブル True: イネーブル	False
tFlt_Temp.OutputEnable	IO 信号出力許可設定: False: ディセーブル True: イネーブル	
tFlt_Temp.TriggerEnable	トリガ出力許可設定: False: ディセーブル True: イネーブル	

3 Fault report structure 概要

表 10 Fault report structure 初期化関数

		値
Cy_SysFlt_ClearStatus (FAULT_STRUCT)	Fault structure ステータスクリア FAULT_STRUCT: Fault report structure 番号 FAULT_STRUCT0 = Fault report structure 0 FAULT_STRUCT1 = Fault report structure 1 FAULT_STRUCT2 = Fault report structure 2 FAULT_STRUCT3 = Fault report structure 3	FAULT_STRUCT0
Cy_SysFlt_ClearInterrupt (FAULT_STRUCT)	Fault structure 割込みフラグクリア	FAULT_STRUCT0
Cy_SysFlt_SetInterruptMask (FAULT_STRUCT)	割込み許可設定: FAULT_STRUCT: Fault report structure 番号	FAULT_STRUCT0
Cy_SysFlt_SetMaskByIdx (FAULT_STRUCT, faultSrc)	故障要因選択 FAULT_STRUCT: Fault report structure 番号 faultSrc: 故障要因	FAULT_STRUCT: FAULT_STRUCT0 faultSrc: CY_SYSFLT_RAMC0_C_ECC CPUSS_RAM0_C_ECC に割当て (Fault number = 58)
Cy_SysFlt_Init (FAULT_STRUCT, tFlt_Temp)	Fault structure 動作設定: FAULT_STRUCT: Fault report structure 番号 tFlt_Temp: 動作設定パラメータ	FAULT_STRUCT0

Code Listing 23 に Fault report 設定部のプログラム例を示します。

以下の説明は、SDL ドライバ部の表記例を示します。

- ptsc**SYSFLT** は、Fault report structure レジスタのベースアドレスポインタを示します。
- ptsc**SYSFLT**->un**STATUS**.u32Register は、レジスタ TRM [2]に記載される。FAULT_STRUCTx_STATUS レジスタです。他のレジスタも同様に記述されます。“**x**”は Fault report structure 番号を示します。

レジスタの共用体および構造体の詳細については `hdr/rev_x/ip` の `cyip_fault_v2.h` を参照してください。

3 Fault report structure 概要

Code Listing 23 Fault report structure の設定例

```
cy_stc_sysint_irq_t irq_cfg =    /*  Configure interrupt structure parameters  */
{
    .sysIntSrc  = cpuss_interrupts_fault_0_IRQn,
    .intIdx     = CPUIntIdx2_IRQn,
    .isEnabled  = true,
};

int main(void)
{
    cy_stc_sysflt_t tFlt_Temp = {0ul};
    :
    /* Clear Status register */
   /*  (1) Initialize interrupt flag. See  Code Listing 24.  */
    Cy_SysFlt_ClearInterrupt(FAULT_STRUCT0);
   /*  (2) Read back  */
     FAULT_STRUCT0->unINTR.u32Register;
   /*  (3) Initialize status register.See  Code Listing 25  */
     Cy_SysFlt_ClearStatus(FAULT_STRUCT0);

    /* Set the signal interface (Interrupt generation) */
   /*  (4) Fault source selection. See  Code Listing 26.  */
    Cy_SysFlt_SetInterruptMask(FAULT_STRUCT0);

   /*  (4) Fault source selection. See  Code Listing 26.  */
    /* Fault source enable 58 System memory controller 0 correctable ECC violation */
   /*  (4) Fault source selection. See  Code Listing 26.  */
    Cy_SysFlt_SetMaskByIdx(FAULT_STRUCT0, CY_SYSFLT_RAMC0_C_ECC);

    /* Init Sysflt */
   /*  (5) Action Setting. See Code Listing 27.  */
    tFlt_Temp.ResetEnable = false;
   /*  (5) Action Setting. See Code Listing 27.  */
    tFlt_Temp.OutputEnable = false;
   /*  (5) Action Setting. See Code Listing 27.  */
    tFlt_Temp.TriggerEnable = false;
   /*  (5) Action Setting. See Code Listing 27.  */
    Cy_SysFlt_Init(FAULT_STRUCT0, &tFlt_Temp);

    /*  Interrupt setting */
   /*  (6) Configure interrupt setting. See Code Listing 3 and Code Listing 4.  */
    Cy_SysInt_InitIRQ(&irq_cfg);
   /*  Set interrupt handler  */
    Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, irqFaultReport0Handler);

    /* Set the Interrupt Priority & Enable the Interrupt */
   /*  (7) Configure NVIC. Priority sets to “0”.  */
    NVIC_SetPriority(irq_cfg.intIdx, 0ul);
   /*  (7) Configure NVIC. Priority sets to “0”.  */
    NVIC_EnableIRQ(irq_cfg.intIdx);
    :
    for(;;)
```

```
    {
    }
}
```

Code Listing 24 Cy_SysFlt_ClearInterrupt () 機能

```
void Cy_SysFlt_ClearInterrupt(volatile stc_FAULT_STRUCT_t *ptscSYSFLT)
{
    ptscSYSFLT->unINTR.stcField.u1FAULT = true;   /* Fault structure interrupt flag clear */
}
```

Code Listing 25 Cy_SysFlt_ClearStatus () 機能

```
void Cy_SysFlt_ClearStatus(volatile stc_FAULT_STRUCT_t *ptscSYSFLT)
{
    ptscSYSFLT->unSTATUS.u32Register = 0x00000000UL;   /* Fault structure status clear */
}
```

Code Listing 26 Cy_SysFlt_SetMaskByIdx()および Cy_SysFlt_SetInterruptMask() 機能

```
void Cy_SysFlt_SetMaskByIdx(volatile stc_FAULT_STRUCT_t *ptscSYSFLT, cy_en_sysflt_source_t idx)
{
    switch(idx / 32)
    {
    case 0:
        ptscSYSFLT->unMASK0.u32Register |= (1UL << (uint32_t)idx);   /* Fault source select */
        break;
    case 1:
        ptscSYSFLT->unMASK1.u32Register |= (1UL << ((uint32_t)idx - 32UL));
        break;
    case 2:
        ptscSYSFLT->unMASK2.u32Register |= (1UL << ((uint32_t)idx - 64UL));
        break;
    default:
        break;
    }
    return;
}

void Cy_SysFlt_SetInterruptMask(volatile stc_FAULT_STRUCT_t *ptscSYSFLT)
{
    ptscSYSFLT->unINTR_MASK.stcField.u1FAULT = true;    /* Fault structure interrupt enable */
}
```

Code Listing 27 Cy_SysFlt_Init() 機能

```
void Cy_SysFlt_Init(volatile stc_FAULT_STRUCT_t *ptscSYSFLT,const cy_stc_sysflt_t * config)
{
    if (config->TriggerEnable)   /* Fault action select */
    {
        ptscSYSFLT->unCTL.stcField.u1TR_EN =  true;   /* Trigger output enable  */
    }
    else
    {
        ptscSYSFLT->unCTL.stcField.u1TR_EN = false;   /* Trigger output enable  */
    }

    if (config->OutputEnable)
    {
        ptscSYSFLT->unCTL.stcField.u1OUT_EN = true;    /* IO output signal enable  */
    }
    else
    {
        ptscSYSFLT->unCTL.stcField.u1OUT_EN = false;   /* IO output signal enable  */
    }

    if (config->ResetEnable)
    {
        ptscSYSFLT->unCTL.stcField.u1RESET_REQ_EN = true;   /* Reset request enable  */
    }
    else
    {
        ptscSYSFLT->unCTL.stcField.u1RESET_REQ_EN = false;   /* Reset request enable  */
    }
}
```

3.3 故障検出時の処理

前述のとおり、Fault report structure が故障をキャプチャすると、故障の発生をシステムに通知できます。加えて、故障解析のための追加情報をキャプチャします。

図 23 に割込み通知されたときの処理例を示します。

3 Fault report structure 概要

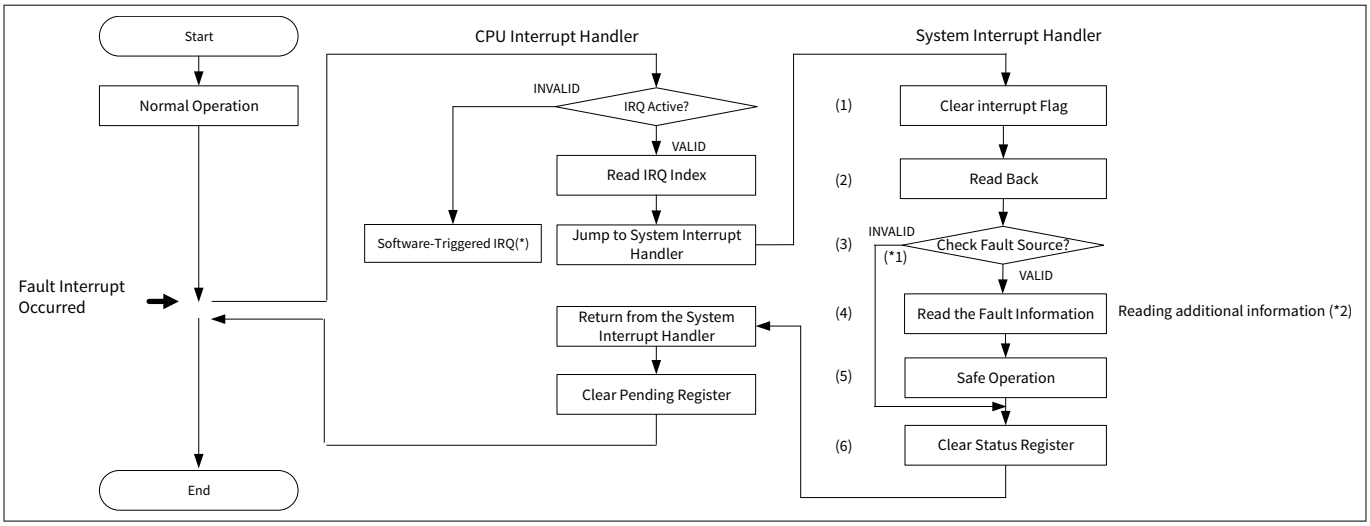

図 23　　故障検出時の処理例

***注**:　(*) ソフトウェアトリガ割込みレジスタ (STIR) を使用してソフトウェア割込みが発生した場合、SYSTEM_INT_VALID ビットはセットされません。ソフトウェア割込みについてはソフトウェア割込み (CPU レジスタを使用) を参照してください。*

***注**:　(*1) 故障要因が有効でない場合 (FAULT_STRUCT_STATUS.VALID = 0)、FAULT_STRUCT_STATUS_IDX と FAULT_STRUCT_DATA は無効です。*

***注**:　(*2) 追加情報として使用されるレジスタの数は、検出された故障によって異なります。詳細についてはアーキテクチャ TRM [2] およびレジスタ TRM [2] を参照してください。*

ソフトウェアは、追加情報から故障の原因を確認し、故障状況に応じて、適切な安全動作または診断プログラムを実行します。安全動作後、通常動作に復帰できる場合は、割込みから復帰してください。

Code Listing 28 に故障割込みハンドラのプログラム例を示します。

3 Fault report structure 概要

Code Listing 28 故障割込みハンドラ

```
   /*  Interrupt handler registered in the configuration part */
void irqFaultReport0Handler(void)
{
    cy_en_sysflt_source_t status;

    /* Clear Interrupt flag */
   /*  (1) Clear interrupt flag. See  Code Listing 24 */
    Cy_SysFlt_ClearInterrupt(FAULT_STRUCT0);
    FAULT_STRUCT0->unINTR.u32Register; /* Read Back */   /*  (2) Read back */

   /*  (3) Check fault source. See Code Listing 29. */
    status = Cy_SysFlt_GetErrorSource(FAULT_STRUCT0);

   /*  (3) Check fault source. See Code Listing 29. */
    if(status == CY_SYSFLT_RAMC0_C_ECC)
    {

   /*  (4) Read the fault information. See Code Listing 30. */
        violatingAddr = Cy_SysFlt_GetData0(FAULT_STRUCT0);
        violatingInfo.u32 = Cy_SysFlt_GetData1(FAULT_STRUCT0);

   /*  (5) Safe Operation. (Fault handling)  */
        /* User program here.. */

    }
    /* Clear Status register */
   /*  (6) Clear status register. See Code Listing 25. */
    Cy_SysFlt_ClearStatus(FAULT_STRUCT0);
}
```

Code Listing 29 CySysFlt_GetErrorSource() 機能

```
cy_en_sysflt_source_t Cy_SysFlt_GetErrorSource(volatile stc_FAULT_STRUCT_t *ptscSYSFLT)
{
    if(ptscSYSFLT->unSTATUS.stcField.u1VALID == 1u)   /*  Check Fault is valid  */
    {
       /*  Read Fault Index  */
        return((cy_en_sysflt_source_t)(ptscSYSFLT->unSTATUS.stcField.u7IDX));
    }
    else
    {
        return CY_SYSFLT_NO_FAULT;
    }
}
```

Code Listing 30 Cy_SysFltGetData0 and 1 () 機能

```
uint32_t Cy_SysFlt_GetData0(volatile stc_FAULT_STRUCT_t *ptscSYSFLT)
{
    return(ptscSYSFLT->unDATA[0].u32Register);
}     /*  Read additional information of fault  */

uint32_t Cy_SysFlt_GetData1(volatile stc_FAULT_STRUCT_t *ptscSYSFLT)
{
    return(ptscSYSFLT->unDATA[1].u32Register);     /*  Read additional information of fault  */
}
```

3.4 Fault report structure の使用例

Fault report structure の使用例をユースケースに従って説明します。

ここでの IDX 番号は、障害インデックス番号を示します。IDX 番号は、CYT2B シリーズのものです。実際に使用するインデックス番号についてはデバイスデータシート [1]を参照してください。

3.4.1 リセット生成

ここでは、故障検出時にリセット信号生成するケースについて説明し、その動作と初期設定を示します。

3.4.1.1 ユースケース

この設定では、CM0+が SMPU 保護に違反するとリセットが発生します。

- 使用する Fault report structure: #0
- 故障要因: CPUSS_MPU_VIO_0 (IDX: 0)
- 故障時の動作: リセット生成

注: (*) 故障の割当てと追加情報については、アーキテクチャ TRM [2]およびレジスタ TRM [2]を参照してください。

故障リセットにより、warm/soft リセットが発生します。故障解析のため、FAULT_STATUS と FAULT_DATA レジスタのような故障情報は、warm/soft リセット中は保持されます。

3.4.1.2 コンフィグレーション

- 初期設定手順
  初期設定手順に従って設定してください。使用する Fault report structure (Fault Report Structure#0)を初期化し、検出時の動作 (Reset) と故障要因 (MPU_VIO_0) を設定します。表 11 および表 12 に、Fault report structure 初期化のための SDL 設定部のパラメータと関数を示します。

表 11 Fault report structure 初期化パラメータ

パラメータ	説明	値
tFlt_Temp.ResetEnable	リセット要求設定 False: ディセーブル True: イネーブル	True

(続く)

表 11 (続き) Fault report structure 初期化パラメータ

パラメータ	説明	値
tFlt_Temp.OutputEnable	IO 信号出力許可設定: False: ディセーブル True: イネーブル	False
tFlt_Temp.TriggerEnable	トリガ出力許可設定: False: ディセーブル True: イネーブル	False

表 12 Fault report structure 初期化関数

関数	説明	値
Cy_SysFlt_SetMaskByIdx (FAULT_STRUCT, faultSrc)	故障要因選択 FAULT_STRUCT: Fault report structure 番号 faultSrc: 故障要因	FAULT_STRUCT: FAULT_STRUCT0 faultSrc: CY_SYSFLT_MPU_0 MPU_VIO_0 に割当て (Fault number = 0)

Code Listing 31 にリセット生成用の Fault Report Structure 設定プログラム例を示します。

Code Listing 31 Fault Report Structure の設定例

```
    :
/* Fault source enable 58 System memory controller 0 correctable ECC violation */
    /*  Fault source selection. See  Code Listing 26. */
     Cy_SysFlt_SetMaskByIdx(FAULT_STRUCT0, CY_SYSFLT_MPU_0);
:
/* Init Sysflt */
     tFlt_Temp.ResetEnable = true;     /*  Action Setting. See Code Listing 27. */
     tFlt_Temp.OutputEnable = false;   /*  Action Setting. See Code Listing 27. */
     tFlt_Temp.TriggerEnable = false;   /*  Action Setting. See Code Listing 27. */
     Cy_SysFlt_Init(FAULT_STRUCT0, &tFlt_Temp);    /*  Action Setting. See Code Listing 27. */
     :
```

3.4.2 割込み構造を使用した NMI 生成

ここでは、故障検出時、CPU への NMI 生成について説明し、その動作, 初期設定, および故障処理について説明します。

図 24 に、Fault report structure の初期設定手順を示します。Fault report structure を初期化し、故障の接続、および故障時の動作を設定します。

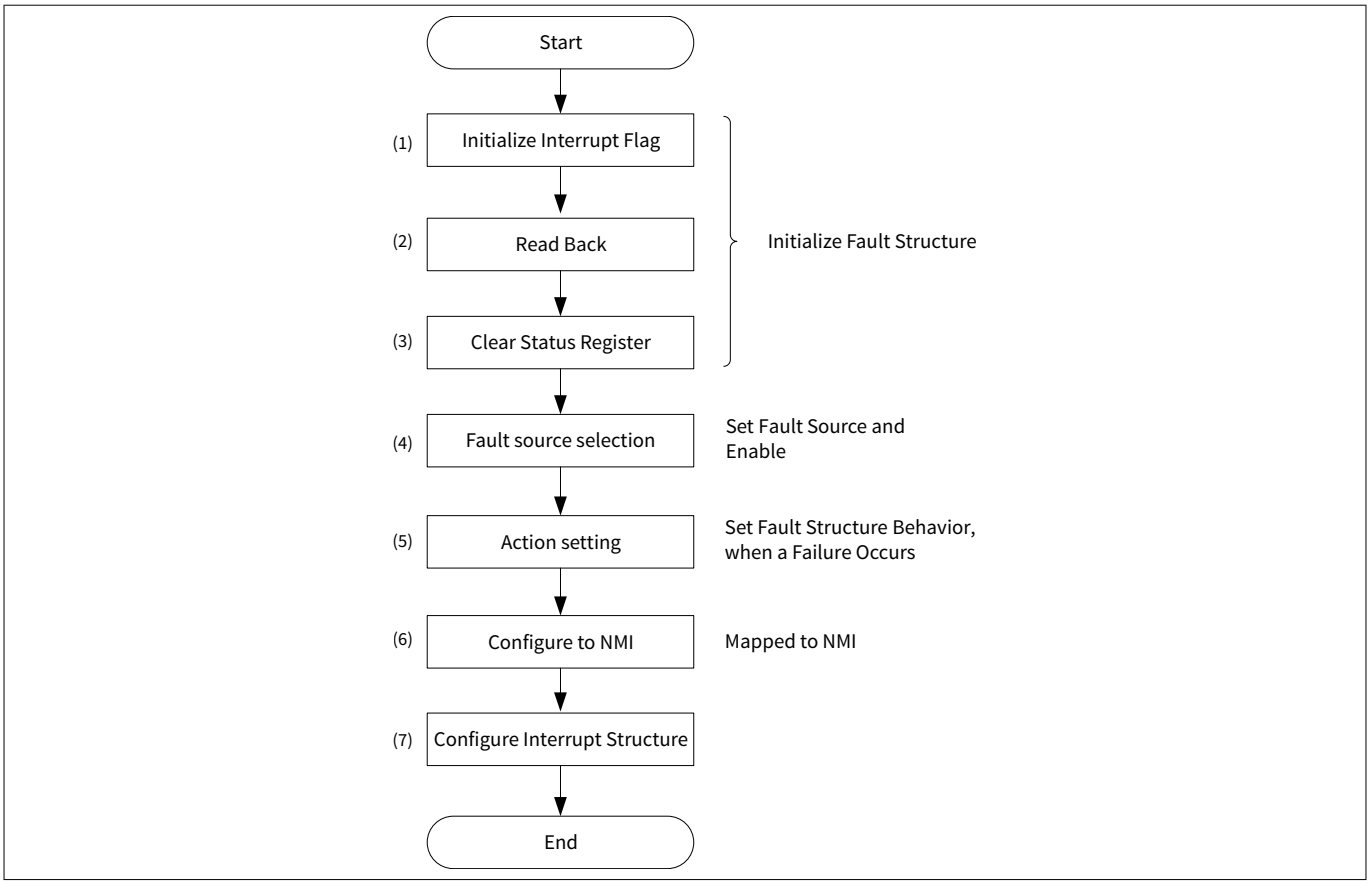

図 24　　NMI 生成用の Fault report structure 初期設定フロー

3.4.2.1　ユースケース

注:　この設定では、CM0+ が SMPU 保護に違反するとリセットが発生します。CPUSS_BUFF_CTL レジスタの設定よっては、PPU 保護違反によりバスエラーが発生する場合があります。詳細については、アーキテクチャ TRM [2] とレジスタ TRM [2] を参照してください。

- Fault report structure を使用: #0
- 故障要因: PERI_MS_VIO_1 (IDX: 29) CM4 周辺マスタインターフェース、PPU 違反 [1)]
- 追加情報:
  - DATA0[31:0] 違反を検出したアドレス [1)]
  - DATA1[0]はユーザ読み出し [1)]
  - DATA1[1]はユーザ書き込み [1)]
  - DATA1[2]はユーザ実行 [1)]
  - DATA1[3]は特権読み出し [1)]
  - DATA1[4]は特権書き込み [1)]
  - DATA1[5]は特権実行 [1)]
  - DATA1[6]は非セキュア [1)]
  - DATA1[11:8]はマスタの識別子 [1)]

[1] フォルトの割り当てと追加情報については、アーキテクチャ TRM [2] とレジスタ TRM [2] を参照してください。

  - DATA1[15:12]は保護コンテキストの識別子 [1)]
  - DATA1[31:28]は故障識別子コード [1)]
- 割込み生成
- CPU NMI に割当て

3.4.2.2 コンフィグレーション

初期設定手順に従って設定してください。表 13 および表 14 に、Fault report structure 初期化のための SDL 設定部のパラメータと関数を示します。

表 13 Fault report structure 初期化パラメータ

パラメータ	説明	値
irq_cfg.sysIntSrc	システム割込みインデックス番号設定 [0: 1022]	cpuss_interrupts_fault_0_IRQn Fault structure #0 に割当て (インデックス番号 = 8)
irq_cfg.intIdx	CPU 割込み番号設定 [0: 7] CPUIntIdx0_IRQn は IRQ 0 に割当て CPUIntIdx1_IRQn は IRQ 1 に割当て CPUIntIdx2_IRQn は IRQ 2 に割当て CPUIntIdx3_IRQn は IRQ 3 に割当て CPUIntIdx4_IRQn は IRQ 4 に割当て CPUIntIdx5_IRQn は IRQ 5 に割当て CPUIntIdx6_IRQn は IRQ 6 に割当て CPUIntIdx7_IRQn は IRQ 7 に割当て	CPUIntIdx4_IRQn
irq_cfg.isEnabled	割込みイネーブル設定 False: ディセーブル True: イネーブル	True
tFlt_Temp.ResetEnable	リセット要求設定 False: ディセーブル True: イネーブル	False
tFlt_Temp.OutputEnable	IO 信号出力許可設定: False: ディセーブル True: イネーブル	False
tFlt_Temp.TriggerEnable	トリガ出力許可設定: False: ディセーブル True: イネーブル	False

1 フォルトの割り当てと追加情報については、アーキテクチャ TRM [2] とレジスタ TRM [2] を参照してください。

3 Fault report structure 概要

表 14　　Fault report structure 初期化関数

関数	説明	値
Cy_SysFlt_ClearStatus (FAULT_STRUCT)	Fault structure ステータスをクリア FAULT_STRUCT: Fault report structure 番号 FAULT_STRUCT0 = Fault report structure 0 FAULT_STRUCT1 = Fault report structure 1 FAULT_STRUCT2 = Fault report structure 2 FAULT_STRUCT3 = Fault report structure 3	FAULT_STRUCT0
Cy_SysFlt_ClearInterrupt (FAULT_STRUCT)	Fault structure 割込みフラグをクリア	FAULT_STRUCT0
Cy_SysFlt_SetInterruptMask (FAULT_STRUCT)	割込み許可設定: FAULT_STRUCT: Fault report structure 番号	FAULT_STRUCT0
Cy_SysFlt_SetMaskByIdx (FAULT_STRUCT, faultSrc)	故障要因選択 FAULT_STRUCT: Fault report structure 番号 faultSrc: 故障要因	FAULT_STRUCT: FAULT_STRUCT0 faultSrc: CY_SYSFLT_MS_PPU_1 PERI_MS_VIO_1 に割当て (Fault number = 29)
Cy_SysFlt_Init (FAULT_STRUCT, tFlt_Temp)	Fault structure 動作設定: FAULT_STRUCT: Fault report structure 番号 tFlt_Temp: 動作パラメータ	FAULT_STRUCT0
Cy_SysInt_SetIntSourceNMI (Reg_num, sysintSrc)	NMI レジスタ設定 Reg_num: CM4 NMI 制御レジスタ番号 CPUSS_CM4_NMI_CTLx レジスタ (x = 0 から 3) sysintSrc: システム割込みインデックス番号	Reg_nim: 0ul CPUSS_CM4_NMI_CTL0 レジスタ使用 sysintSrc: irq_cfg.sysIntSrc

Code Listing 32 に NMI 生成の Fault report structure 設定部のプログラム例を示します。

3 Fault report structure 概要

Code Listing 32 Fault report structure の設定例

```
cy_stc_sysint_irq_t irq_cfg =    /*  Configure interrupt structure parameters. */
{
    .sysIntSrc  = cpuss_interrupts_fault_0_IRQn,
    .intIdx     = CPUIntIdx4_IRQn,
    .isEnabled  = true,
};

int main(void)
{
    cy_stc_sysflt_t tFlt_Temp = {0ul};
    :
    /* Clear Status register */
   /*  (1) Initialize interrupt flag. See  Code Listing 24. */
    Cy_SysFlt_ClearInterrupt(FAULT_STRUCT0);
   /*  (2) Read back. */
     FAULT_STRUCT0->unINTR.u32Register;
   /*  (3) Initialize status register. See  Code Listing 25. */
     Cy_SysFlt_ClearStatus(FAULT_STRUCT0);

    /* Set the signal interface (Interrupt generation) */
   /*  (4) Fault source selection. See Code Listing 26.. */
    Cy_SysFlt_SetInterruptMask(FAULT_STRUCT0);

    /* Fault source enable 29 CM4 Peripheral Master Interface PPU violation */
   /*  (4) Fault source selection. See Code Listing 26. */
    Cy_SysFlt_SetMaskByIdx(FAULT_STRUCT0, CY_SYSFLT_MS_PPU_1);

    /* Init Sysflt */
   /*  (5) Action Setting. See Code Listing 27. */
    tFlt_Temp.ResetEnable = false;
   /*  (5) Action Setting. See Code Listing 27. */
    tFlt_Temp.OutputEnable = false;
   /*  (5) Action Setting. See Code Listing 27. */
    tFlt_Temp.TriggerEnable = false;
   /*  (5) Action Setting. See Code Listing 27. */
    Cy_SysFlt_Init(FAULT_STRUCT0, &tFlt_Temp);

    /* Set NMI mapping of CM4 */
   /*  (6) Configure NMI See Code Listing 10.. */
    Cy_SysInt_SetIntSourceNMI(0ul, irq_cfg.sysIntSrc);

    /*  Interrupt setting */
    Cy_SysInt_InitIRQ(&irq_cfg);
   /*  (7) Configure interrupt setting. See Code Listing 3 and Code Listing 4.. */
    Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, NMI_Handler);   /*  Set NMI handler. */

    /* CPUSS_BUFF_CTL register setting */
    CPUSS->unBUFF_CTL.stcField.u1WRITE_BUFF = 1ul;
   :
    for(;;)
    {
```

```
        }
    }
```

- 故障検出時の処理例
  PPU#1 マスタへのアクセス違反が発生すると、Fault report structure は故障情報をキャプチャし、割込みを出力します。割込みコントローラは、Fault report structure からの割込みを NMI として CPU に通知します。

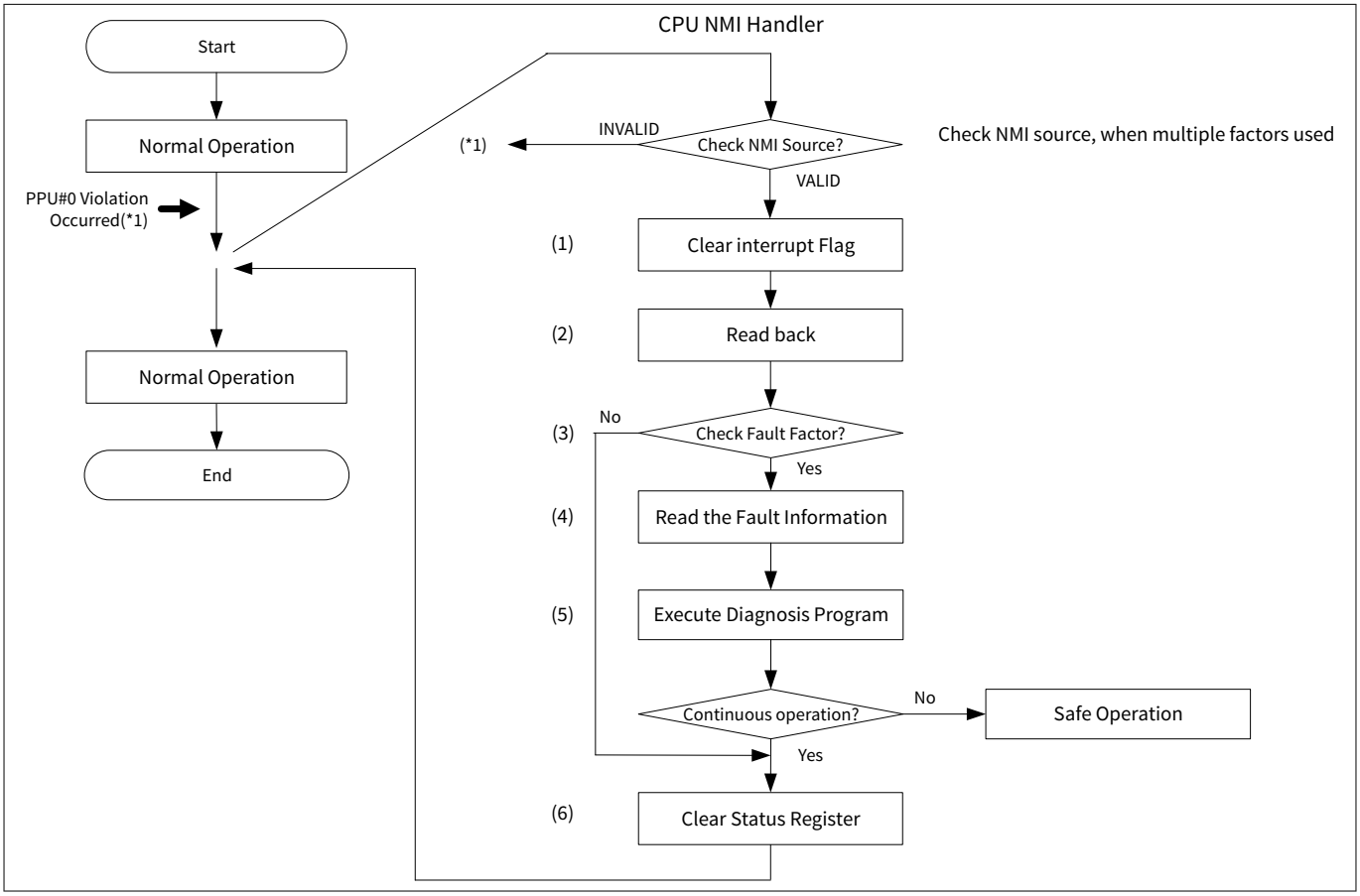

図 25　　故障処理フロー例

*注:　(*1) システムによって複数 (4) の NMI 要因がある場合、NMI ハンドラは CPU NMI のソースを特定するために、選択されたすべてのシステム割込みソースの確認が必要な場合があります。*

ソフトウェアは、追加情報から故障の原因を確認し、故障状況に応じて、適切な安全動作または診断プログラムを実行します。安全動作後、通常動作に復帰できる場合は、割込みから復帰して h ください。

Code Listing 33 に故障割込みハンドラのプログラム例を示します。

Code Listing 33 NMI 生成の故障割込みハンドラ

```
void NMI_Handler(void)    /*  NMI handler registered in the configuration part. */
{
    cy_en_sysflt_source_t status;

    /* Clear Interrupt flag */
    Cy_SysFlt_ClearInterrupt(FAULT_STRUCT0);   /* (1) Clear interrupt flag. See Code Listing
24.*/
    FAULT_STRUCT0->unINTR.u32Register; /* Read Back */   /*  (2) Read back. */

    status = Cy_SysFlt_GetErrorSource(FAULT_STRUCT0);   /*  (3) Check fault factor. See Code
Listing 29. */

    if(status == CY_SYSFLT_MS_PPU_1)    /*  (3) Check fault factor. See Code Listing 29. */
    {
   /*  (4) Read the fault information. See Code Listing 30. */
        violatingAddr = Cy_SysFlt_GetData0(FAULT_STRUCT0);
        violatingInfo.u32 = Cy_SysFlt_GetData1(FAULT_STRUCT0);

   /*  (5) Execute diagnosis and Safe Operation (Fault handling). */
            /* User program here.. */

    }
    /* Clear Status register */
    Cy_SysFlt_ClearStatus(FAULT_STRUCT0);    /*  (6) Clear status register. See Code Listing
25. */
}
```

4 用語集

表 15 用語集

用語	説明
CPU	Central Processing Unit (中央処理装置)
NVIC	Nested vectored interrupt controller (Nested vectored 割込みコントローラ)
WIC	Wakeup interrupt controller (Wakeup 割込みコントローラ)
NMI	Non-maskable interrupt (Non-maskable 割込み)
DeepSleep	TRAVEO™ T2G ファミリの低消費電力モード詳細については、アーキテクチャ TRM [2] の“Device Power Mode”章を参照してください。
ISR	Interrupt service routine (割込みサービスルーチン)
IRQ	Interrupt request (割込み要求)
WFI	Wait for interrupt (割込み待ち命令)
MMIO	メモリマップド I/O
MPU	Memory Protection Unit (メモリ保護ユニット)。詳細はアーキテクチャ TRM [2]の“Protection Unit”章を参照してください。
SMPU	Shared Memory Protection Unit (共有メモリ保護ユニット)。詳細はアーキテクチャ TRM [2]の“Protection Unit”章を参照してください。
PPU	Peripheral Protection Unit (周辺保護ユニット)。詳細はアーキテクチャ TRM [2]の“Protection Unit”章を参照してください。
GPIO	General purpose input/output (汎用入力/出力)。詳細はアーキテクチャ TRM [2]の“IO System”章を参照してください。
TCPWM	Timer (タイマ), Counter (カウンタ), および Pulse Width Modulator (パルス幅変調器)。詳細はアーキテクチャ TRM [2]の“Timer, Counter, and PWM”章を参照してください。
RTC	Real-time clock (リアルタイムクロック)。詳細はアーキテクチャ TRM [2]の“Real-time Clock”章を参照してください。

5 参考資料

以下は、TRAVEO™ T2G ファミリ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。これらの資料を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。

[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] ボディコントローラ エントリファミリ

- 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] ボディコントローラ ハイファミリ

- 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] クラスタ 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] クラスタ Entry ファミリ

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

[6] アプリケーションノート

- AN220193 - TRAVEO™ T2G ファミリ GPIO の使用方法
- AN220224 – How to use timer, counter, and PWM (TCPWM) in TRAVEO™ T2G family (Japanese)

6 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
**	2018-06-05	これは英語版 002-19842 Rev. **を翻訳した日本語版 002-23842 Rev. **です。
*A	2018-12-21	これは英語版 002-19842 Rev. *A を翻訳した日本語版 002-23842 Rev. *A です。主な変更点は以降のとおりです。対象製品番号を変更しました。(CYT2B シリーズ) 初期設定に VTOR 説明のための Note (*1) を追加しました。 周辺機能割込み処理の IDX 番号を変更しました。17, 18, 19 -> 21, 22, 23 システム割込みによる NMI の生成の IDX 番号を変更しました。274, 275 -> 273, 274 ソフトウェア割込み (周辺機能を使用)の IDX 番号を変更しました。275 -> 274 割込み構造を使用した NMI 生成の IDX 番号を変更しました。47 -> 8
*B	2019-03-06	これは英語版 002-19842 Rev. *B を翻訳した日本語版 002-23842 Rev. *B です。主な変更点は以降のとおりです。対象の製品番号を追加しました。(CYT4B シリーズ)
*C	2019-08-22	これは英語版 002-19842 Rev. *C を翻訳した日本語版 002-23842 Rev. *C です。主な変更点は以降のとおりです。対象の製品番号を追加しました。(CYT4D シリーズ)
*D	2020-04-14	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) 対象の製品番号を追加しました。(CYT3 シリーズ)
*E	2021-03-01	これは英語版 002-19842 Rev. *E を翻訳した日本語版 002-23842 Rev. *E です。主な変更点は以降のとおりです。図 3 を変更しました。 セクション 2.3, 2.4, 2.6, 3.2, 3.3, 3.4 を変更しました。 - フローチャートを変更しました。 - ユースケースを変更しました。 - 設定およびコードを追加しました。 参考資料を更新しました。 その他の参考資料にサンプルドライバーライブラリの情報を追加しました。
*F	2021-10-11	これは英語版 002-19842 Rev. *F を翻訳した日本語版 002-23842 Rev. *F です。主な変更点は以降のとおりです。セクション 2.3 に Note を追加しました。 図 4 および図 12 を変更しました。 対応するデバイスを追加しました。(CYT3DL)
*G	2024-03-27	これは英語版 002-19842 Rev. *G を翻訳した日本語版 002-23842 Rev. *G です。
*H	2025-09-17	これは英語版 002-19842 Rev. *H を翻訳した日本語版 002-23842 Rev. *H です。主な変更点は以降のとおりです。CYT6BJ の関連箇所を更新および修正しました。

Trademarks

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

Edition 2025-09-18

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-bwl1682064216032

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。