TRAVEO™ T2G ファミリ MCU でのウォッチドッグタイマの使用

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 TRAVEO™ T2G ファミリ MCU に搭載されているウォッチドッグタイマの操作方法について説明します。このドキュメントはベーシックウォッチドッグタイマとマルチカウンタウォッチドッグタイマの機能、およびフォールト, 割込み, リセットを生成するために必要な設定を紹介します。

対象者

本書は、TRAVEO™ T2G ファミリ MCU 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は、TRAVEO™ T2G ファミリ MCU のウォッチドッグタイマ (WDT) について説明します。WDT は警告割込み, フォールト, またはリセットの生成によって、予期しないファームウェア実行パスを検出します。これにより、システムはアプリケーションプログラムによる安全ではない実行から回復できます。

所定の周期を観測するために使用される異なるカウンタを含み、定期的にタイマをクリアすることによって、WDT はアプリケーションソフトウェアの通常動作を監視します。WDT はあらかじめ定められた周期に達すると、その状態を異常として検出し、リセットまたは割込みまたは障害イベントを生成します。TRAVEO™ T2G は、基本 WDT とマルチカウンタ WDT (MCWDT) の 2 種類の WDT をサポートしています。どちらの WDT もウィンドウモードをサポートしており、ウォッチドッグタイマが処理されなければならない上限時間と下限時間を定義できます。

ベーシック WDT は、ハードウェアのリセット解除後に起動されます。その動作モードは、初期設定時にアプリケーションソフトウェアによって設定されます。Active, Sleep, DeepSleep, および Hibernate の電力モードでカウントされます。

MCWDT は、アプリケーションソフトウェアによって起動と動作モードを設定します。Active, Sleep, DeepSleep のパワーモードでカウントされます。このドキュメントでは、CYT2 シリーズ, CYT3 シリーズ, CYT4 シリーズ, および CYT6 シリーズデバイスについて記載します。図 1 に WDT のブロックダイヤグラムを示します。ベーシック WDT, MCWDT および両方のサブ構造が含まれています。

このアプリケーションノートで使用されてい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 (TRM) の“Watchdog Timer”章を参照してください。

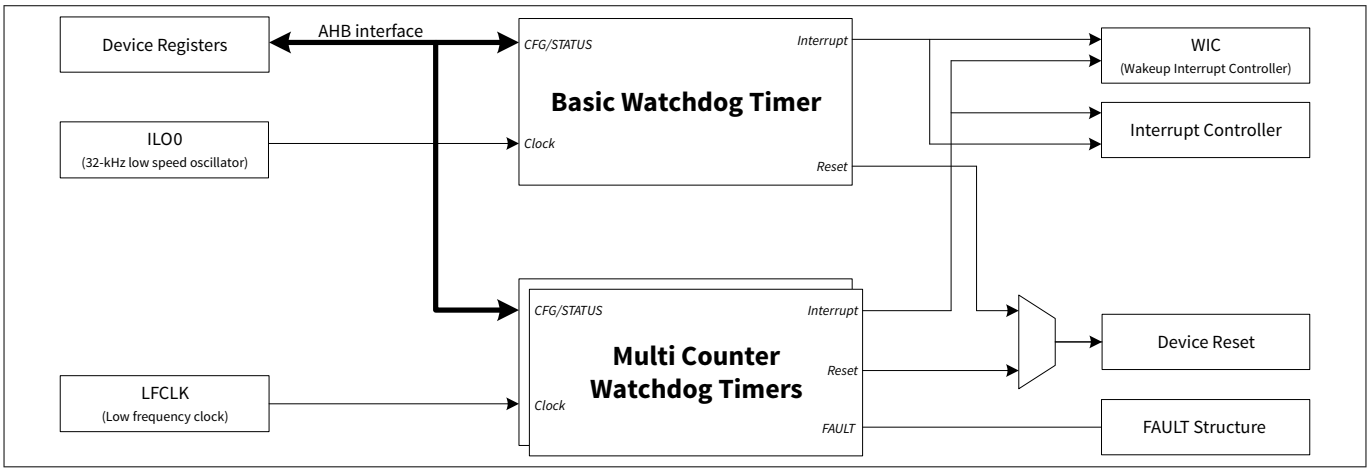

図 1 WDT ブロックダイヤグラム

2 ベーシック WDT

図 2 にベーシック WDT のブロックダイヤグラムを示します。ベーシック WDT は、1 つの 32 ビットフリーランカウンタをサポートします。それは、WDT_CTL レジスタの ENABLE[31]ビットが'1'に設定されている場合、ILO0 クロックでカウントアップします。

WDT ロジックと ILO0 は外部の高電圧電源 (V_{DDD}) によって電源供給されるため、Hibernate での動作が可能です。WDT リセットはチップをアクティブモードに遷移させます。デフォルトでは、ベーシック WDT はイネーブル、UPPER_ACTION はリセット、UPPER_LIMIT は 0x8000 に設定され、保護可能なレジスタはすべてロックされています。UPPER_ACTION と UPPER_LIMIT は、時間内にサービスされない場合にリセット実行するようなベーシック WDT 動作の定義に使用するコンフィギュレーションレジスタです。

WDT コンフィギュレーションレジスタは、WDT を処理するために使用されるレジスタとは別の保護領域にあります。保護領域は周辺保護ユニット (PPU) によって処理されます。詳細は、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“CPU Subsystem (CPUSS)”章を参照してください。

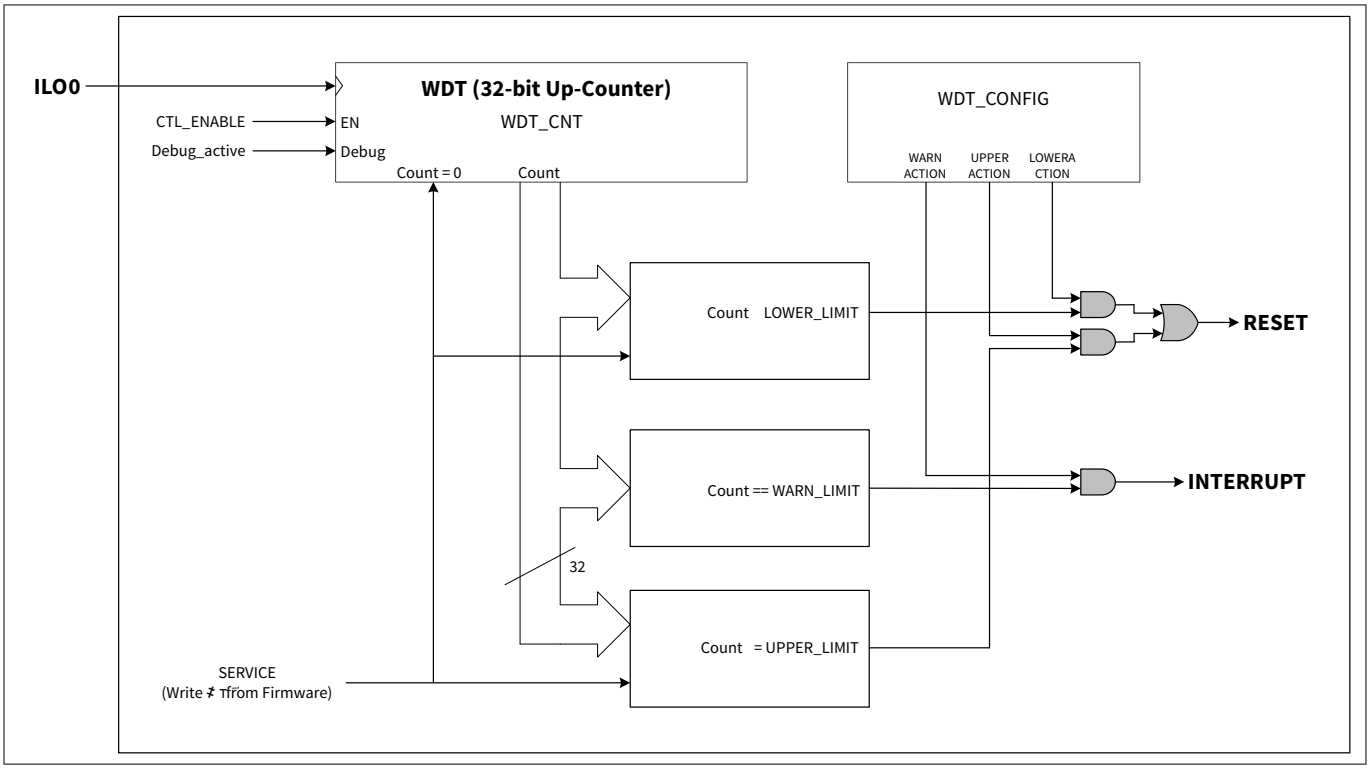

図 2 ベーシック WDT ブロックダイヤグラム

WDT_CONFIG レジスタの設定に応じて、カウンタが関連するカウンタリミットに達すると、割込みまたはリセットイベントが生成できます。以下のアクションには、3 つの閾値を設定できます。

- LOWER-LIMIT: WDT_CONFIG レジスタで LOWER_ACTION[0]ビットが'1'に設定されている場合、WDT が LOWER_LIMIT 値に達する前にウォッチドッグルーチンが処理されると、リセットが発行されます。
- UPPER-LIMIT: WDT_CONFIG レジスタの UPPER_ACTION[4]ビットが'1'にセットされると、WDT が処理される前に WDT が UPPER_LIMIT 値に達するとリセットが発行されます。
- WARN-LIMIT: WDT_CONFIG レジスタの WARN_ACTION[8]ビットが'1'に設定されている場合、WDT が WARN_LIMIT 値に達すると割込みが発生します。

UPPER-LIMIT と LOWER-LIMIT を組み合わせにより、ベーシック WDT のウィンドウモードを構築できます。

WDT_CONFIG レジスタの ACTION ビットで定義されるベーシック WDT モードに応じて、ウォッチドッグカウンタを異なる方法で処理する必要があります。ウィンドウモードでは、ファームウェアはウィンドウタイミング条件を満たすのに十分なウォッチドッグ・サービスタイミングを確保する必要があります。LOWER_ACTION ビットが設定されていない場合、ベーシック WDT は UPPER_LIMIT 値に達する前にいつでも処理できます。

2.1 ソースクロック:

ベーシック WDT に選択可能なソースクロックは、ILO0 クロックに固定されています: 32.768 kHz。

2.2 WDT タイマカウンタ

ベーシック WDT のカウント幅は 32 ビットです。したがって、設定可能なタイマ時間は 30.518 µs～131,072 s です。これらの値は、ILO0 の標準タイミングで計算されています。誤差も考慮する必要があります。詳細については、デバイスのデータシートを参照してください。

2.3 レジスタ保護

ベーシック WDT の設定に使用するレジスタ値を変更するには、LOCK レジスタにある WDT_LOCK[1:0] ビットでの UNLOCK シーケンスが必要です。CNT, CTL, LOWER_LIMIT, UPPER_LIMIT, WARN_LIMIT, CONFIG, および SERVICE レジスタのアンロックには、WDT_LOCK ビットフィールドへの次の書込みアクセスシーケンスを実行する必要があります。

- WDT_LOCK = 1
- WDT_LOCK = 2

ベーシック WDT レジスタのアンロックには、LOCK レジスタへの 1 回の単独アクセスが必要です。

- WDT_LOCK = 3

WDT_LOCK レジスタの読出しによってロック状態を確認してください。読出し値が"0"でない場合は、ベーシック WDT レジスタがロック状態であることを示します。

DeepSleep モードまたは Hibernate モードから Active モードへの移行後、すべてのベーシック WDT レジスタはロックされます。

2.4 警告割込み

ベーシック WDT は、特定のタイミングの割込みを定義するのに使用できる WARN リミットをサポートします。以下のようにさまざまな目的で使用できます。

- **事前警告イベント**: WARN_LIMIT 値を UPPER_LIMIT 値よりも低く設定します。CONFIG レジスタの WARN_ACTION[8] ビットを'1'に設定すると有効になります。時間内に WARN 割込みを実行するために、適切な制限時間を設定してください。
- **ウェイクアップイベント**: ベーシック WDT は、所望のウェイクアップ時間間隔のために、警告割込みを設定することによって、単純なウェイクアップタイマとして使用できます。ウォッチドッグカウンタは Sleep および DeepSleep モードでウェイクアップ割込みコントローラ (WIC) に割込み要求を送信できます。さらに、ベーシック WDT は、デバイスを Hibernate 電力モードから復帰できます。これは通常のウォッチドッグリセット動作の有無によらず使用できます。Hibernate からのウェイクアップは、PWR_HIBERNATE レジスタで設定します。詳細はアーキテクチャ テクニカルリファレンスマニュアル (TRM) の Systems Resources Registers 章を参照してください。ベーシック WDT は、CONFIG レジスタの AUTO_SERVICE[12] ビットを'1'にセットすると自動的に処理できます。ベーシック WDT カウンタがタイムアウトリセット機能付きのウォッチドッグタイマとして使用されない場合、自動サービス設定したベーシック WDT は、定期的な割込みを生成します。これは、CONFIG レジスタの LOWER_ACTION[0] ビットと UPPER_ACTION[1] ビットが'0'に設定されることを意味します。対応する割込みサービスルーチン (ISR) でのベーシック WDT カウンタの保守は不要です。ベーシック WDT カウンタは、ハードウェアによって処理されます。

図 3 は、自動サービスが有効化された 500 ミリ秒の定期ウェイクアップタイミング例を示します。計算は以下の式を使用して行われます。

WARN_LIMIT = 32768 Hz × 500 ms = 16384 = 0x00004000

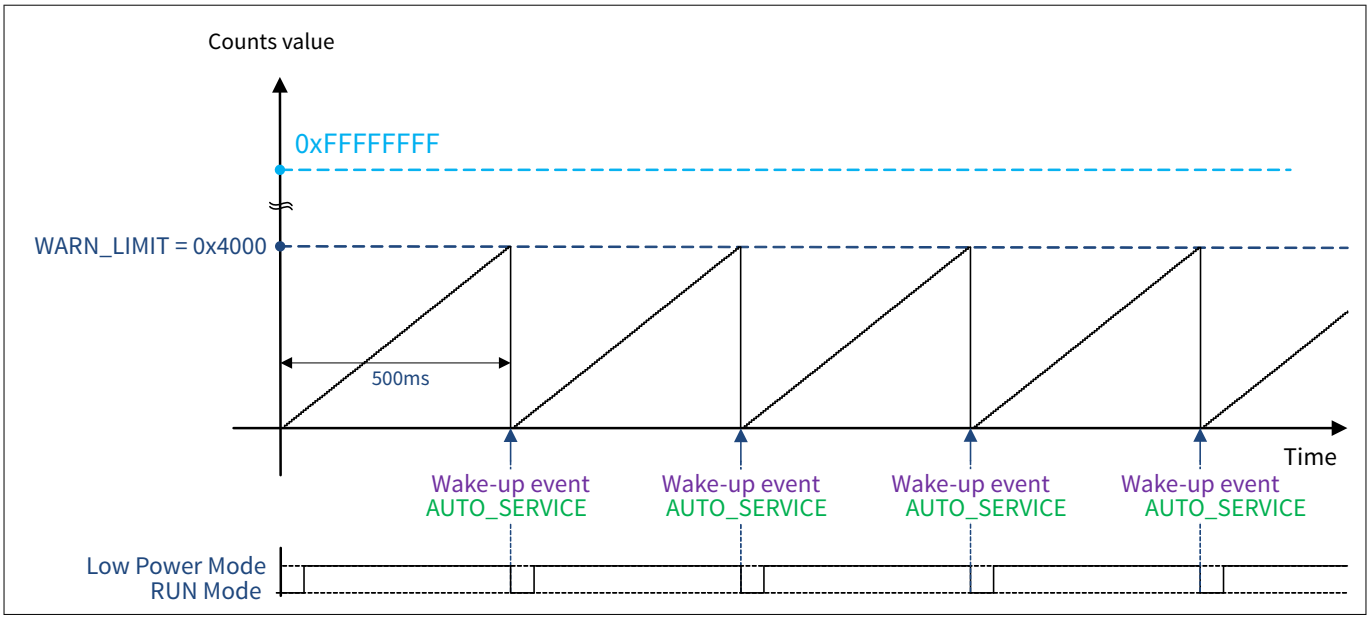

図 3　　ベーシック WDT による定期ウェイクアップ

2.5　タイムアウトモード

ベーシック WDT のレガシーモードは、MCU リセットのために、タイムアウト条件を備えた標準のウォッチドッグ動作です。それは、UPPER_LIMIT レジスタを使用して、ベーシック WDT が時間内に処理されない場合、リセットを生成します。ウォッチドッグ カウンタが UPPER_LIMIT 値と一致したときにリセットをトリガするために、CONFIG レジスタの UPPER_ACTION[4] ビットを'1'に設定します。

WARN_LIMIT レジスタは誤ったウォッチドッグ カウンタ サービスタイミングを示す事前警告イベントとして使用できます。警告割込みを有効にするために、CONFIG レジスタの WARN_ACTION[8] ビットを‘1’に設定します。

図 4 に、1 秒の上限タイムアウト時間および 875 ミリ秒の事前警告割込みタイミングを定義した場合のベーシック WDT の例を示します。対応するレジスタ値は以下のように計算されます。

UPPER_LIMIT = 32768 Hz × 1 sec = 32768 = 0x00008000

WARN_LIMIT = 32768 Hz × 875 ms = 28672 = 0x00007000

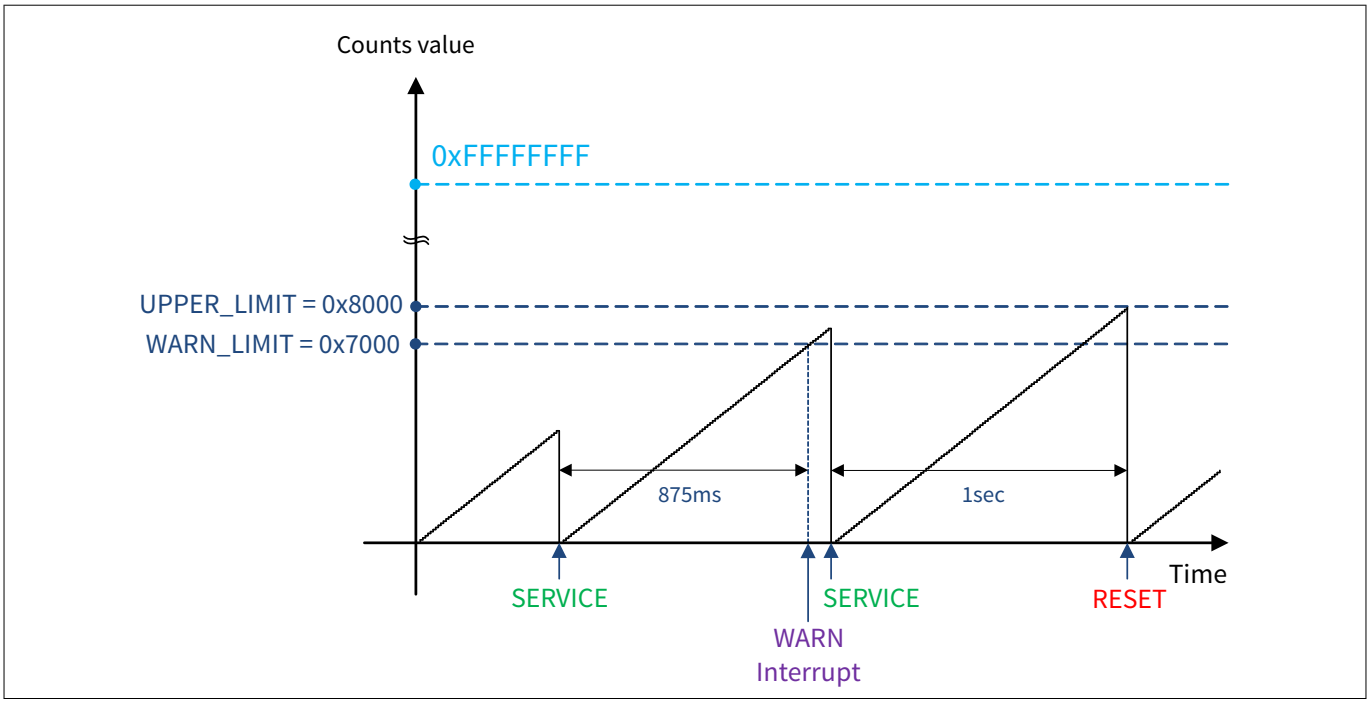

図 4　　ベーシック WDT のタイムアウトと事前警告

この例では、次の 3 つのシナリオを示します。

- ベーシック WDT カウンタが WARN_LIMIT に達する前にサービスします。
- 事前警告 ISR 内でベーシック WDT カウンタを処理します。
- ベーシック WDT カウンタが時間内に処理されない場合、1 秒後に RESET が発行されます。

2.6　　ウインドウモード

TRAVEO™ T2G MCU は、WDT ウィンドウモードを可能にする低いカウンタ閾値を定義するオプションをサポートします。WDT ウィンドウモードは下限値と上限値の 2 つのカウンタリミットの監視をサポートします。カウンタが LOWER_LIMIT レジスタの設定された下限値に達する前にウォッチドッグが処理されると、リセットが発行されます。ベーシック WDT カウンタの上限に達する前にウォッチドッグが処理されないと、リセットが発行されます。2 つの閾値は、ベーシック WDT が処理されなければいけないウィンドウタイミングを定義します。この機能を有効にするために、CONFIG レジスタの LOWER_ACTION[0]ビットを'1'に設定し、LOWER_LIMIT レジスタに適切な下限時間を定義する必要があります。

次の例は、LOWER_LIMIT 値が 150 ミリ秒の場合の計算を示します。

LOWER_LIMIT = 32.768 kHz × 150 ms = 4915 = 0x00000CCC

2.7　　ベーシック WDT の設定

ここでは、インフィニオンが提供する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 WDT 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 ノートのプログラムコードは SDL の一部です。詳細については、その他の関連資料を参照してください。

SDL には設定部とドライバ部があります。設定部は、目的の操作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値に基づいて各レジスタを設定します。

設定部は、お客様のシステムに合わせて設定できます。

図 5 にベーシック WDT の設定フロー例を示します。

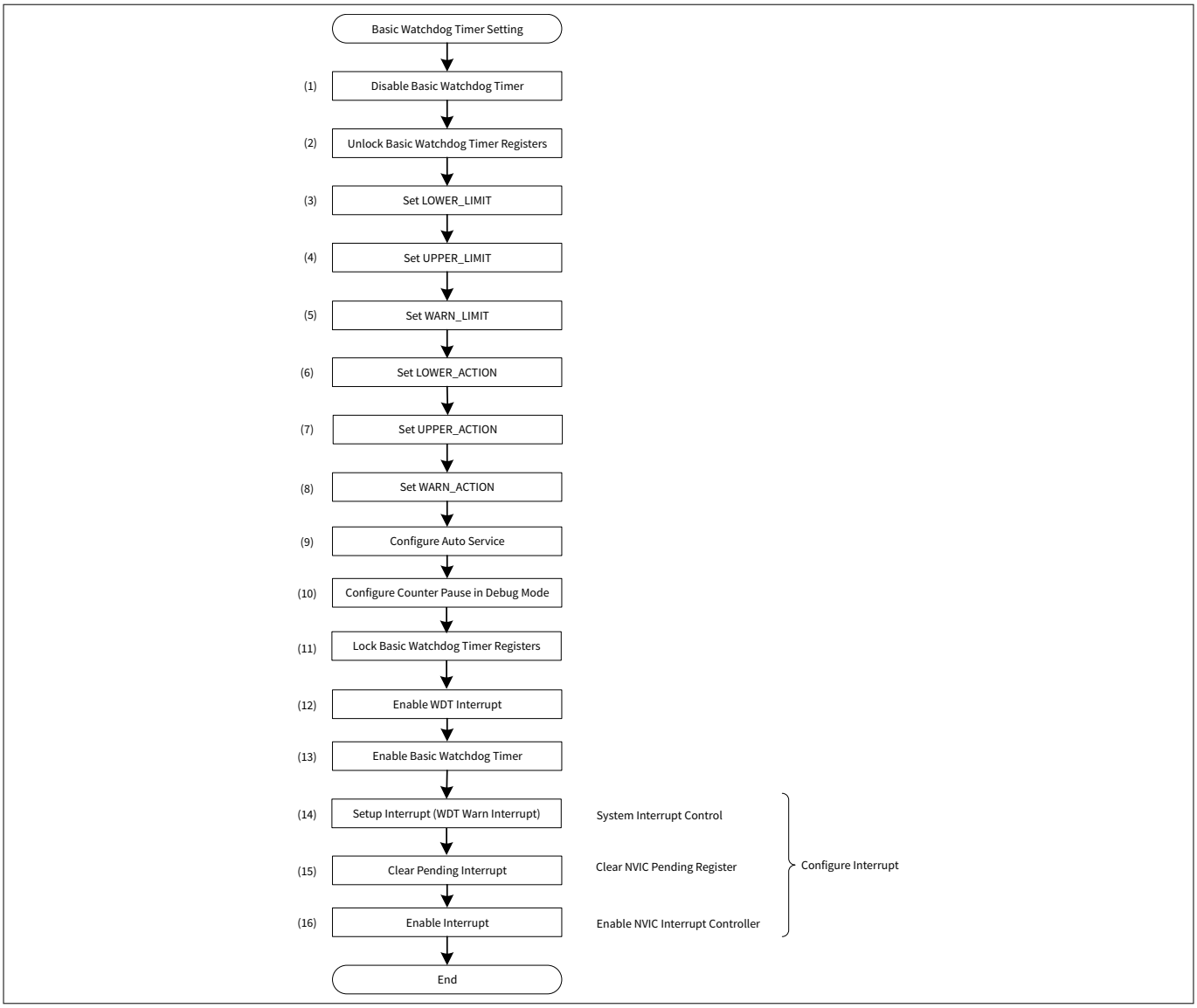

図 5　　ベーシック WDT の設定フロー例

2.7.1　　使用事例

ここでは、次の使用例によってベーシック WDT の例を説明します。ベーシック WDT は、警告割込みハンドラでクリアされます。ベーシック WDT が LOWER_LIMIT と UPPER_LIMIT の間でクリアされない場合、リセットがトリガされます。

使用例:

- LOWER_LIMIT: 125ms
- UPPER_LIMIT: 1 秒
- WARN_LIMIT: 875ms
- ウインドウモード: 使用する
- 警告割込み: 使用する (IRQ 番号: 2)
- Auto service: 未使用
- デバッガ設定: デバッグモード中にカウンタを一時停止するために WDT のトリガ入力をイネーブルする

2.7.2 ベーシック WDT 設定

表 1 に、ベーシック WDT の SDL の設定部のパラメータを示します。

表 1 ベーシック WDT パラメータリスト

機能	説明	値
`Cy_WDT_SetLowerLimit()`	下限設定 (符号なし整数 32 ビット)	4096ul
`Cy_WDT_SetUpperLimit()`	上限設定 (符号なし整数 32 ビット)	32768ul
`Cy_WDT_SetWarnLimit()`	警告制限設定 (符号なし整数 32 ビット)	28672ul
`Cy_WDT_SetLowerAction()`	lower action を “no action”または “reset”に設定: CY_WDT_LOW_UPP_ACTION_NONE = 0ul CY_WDT_LOW_UPP_ACTION_RESET = 1ul	CY_WDT_LOW_UPP_ACTION_RESET
`Cy_WDT_SetUpperAction()`	upper action を “no action”または “reset”に設定: CY_WDT_LOW_UPP_ACTION_NONE = 0ul CY_WDT_LOW_UPP_ACTION_RESET = 1ul	CY_WDT_LOW_UPP_ACTION_RESET
`Cy_WDT_SetWarnAction()`	warn action を “no action”または “interrupt”に設定: CY_WDT_WARN_ACTION_NONE = 0ul CY_WDT_WARN_ACTION_INT = 1ul	CY_WDT_WARN_ACTION_INT
`Cy_WDT_SetAutoService()`	カウント値が WARN_LIMIT に達したときにベーシック WDT を自動的にクリアするように設定: CY_WDT_DISABLE = 0ul CY_WDT_ENABLE = 1ul	CY_WDT_DISABLE
`Cy_WDT_SetDebugRun()`	デバッガを設定 (デバッガを使用する場合に必要) CY_WDT_DISABLE = 0ul CY_WDT_ENABLE = 1ul	CY_WDT_ENABLE

Code Listing 1 に、ベーシック WDT 設定部のプログラム例を示します。割込みの初期設定手順の詳細については、関連ドキュメントに記載されている AN219842 の“Interrupt Structure”章を参照してください。

2 ベーシック WDT

Code Listing 1 ベーシック WDT の設定例

```
cy_stc_sysint_irq_t stc_sysint_irq_cfg_wdt =
{
    .sysIntSrc = srss_interrupt_wdt_IRQn,
    .intIdx    = CPUIntIdx2_IRQn,
    .isEnabled = true,
};

int main(void)
{
    SystemInit();

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

    /*-----------------------*/
    /* Configuration for WDT */
    /*-----------------------*/
    Cy_WDT_Disable();    /* (1) Disable Basic WDT */

    Cy_WDT_Unlock();    /* (2) Unlock Basic WDT registers */

    Cy_WDT_SetLowerLimit(4096ul);    /* (3) Set LOWER_LIMIT */

    Cy_WDT_SetUpperLimit(32768ul);    /* (4) Set UPPER_LIMIT */

    Cy_WDT_SetWarnLimit (28672ul);    /* (5) Set WARN_LIMIT */

    Cy_WDT_SetLowerAction(CY_WDT_LOW_UPP_ACTION_RESET);    /* (6) Set LOWER_ACTION */

    Cy_WDT_SetUpperAction(CY_WDT_LOW_UPP_ACTION_RESET);    /* (7) Set UPPER_ACTION */

    Cy_WDT_SetWarnAction (CY_WDT_WARN_ACTION_INT);    /* (8) Set WARN_ACTION */

    Cy_WDT_SetAutoService(CY_WDT_DISABLE);    /* (9) Disable Auto Service */

    Cy_WDT_SetDebugRun(CY_WDT_ENABLE);    /* (10) Enable counter pause in debug mode */

    Cy_WDT_Lock();    /* (11) Lock Basic WDT registers */

    Cy_WDT_MaskInterrupt();    /* (12) Enable Interrupt */

    Cy_WDT_Enable();    /* (13) Enable Basic WDT */

    /*----------------------------------*/
    /* Interrupt Configuration for WDT  */
    /*----------------------------------*/
    Cy_SysInt_InitIRQ(&stc_sysint_irq_cfg_wdt);   /*(14) Setup Interrupt (WDT Warn Interrupt)*/
    Cy_SysInt_SetSystemIrqVector(stc_sysint_irq_cfg_wdt.sysIntSrc, Wdt_Warn_IntrISR);

    NVIC_ClearPendingIRQ(stc_sysint_irq_cfg_wdt.intIdx);    /* (15) Clear Pending Interrupt */
    NVIC_EnableIRQ(stc_sysint_irq_cfg_wdt.intIdx);    /* (16) Enable Interrupt */
```

```
    for(;;);
}
```

2.7.3 ドライバ部のベーシック WDT 設定プログラム例

Code Listing 2～Code Listing 14 は、ドライバ部でベーシック WDT を設定するプログラム例を示します。

以下は、SDL のドライバ部分のレジスタ表記を説明します。

- **WDT**->un***CTL***.stcField.ul***ENABLE*** は、レジスタ TRM に記載されている WDT_CTL.ENABLE レジスタです。その他のレジスタについても、同様の意味です。

Code Listing 2 ドライバ部のベーシック WDT のディセーブル例

```
void Cy_WDT_Disable(void)
{
    Cy_WDT_Unlock();
    /*  (1) Disable Basic WDT. WDT should be unlocked before being disabled.  */
    WDT->unCTL.stcField.u1ENABLE = 0ul;
    Cy_WDT_Lock();
}
```

Code Listing 3 Example to unlock basic WDT in driver part

```
void Cy_WDT_Unlock(void)
{
    uint32_t interruptState;
    interruptState = Cy_SysLib_EnterCriticalSection();

    /* The WDT lock is to be removed by two writes */
    /*  (2) Unlock Basic WDT registers when interrupts are disabled */
    WDT->unLOCK.stcField.u2WDT_LOCK = 1ul;
    WDT->unLOCK.stcField.u2WDT_LOCK = 2ul;

    Cy_SysLib_ExitCriticalSection(interruptState);
}
```

Code Listing 4 ドライバ部の Lower Limit 設定例

```
__STATIC_INLINE void Cy_WDT_SetLowerLimit(uint32_t match)
{
    WDT->unLOWER_LIMIT.stcField.u32LOWER_LIMIT = match;    /*  (3) Set LOWER_LIMIT */
}
```

Code Listing 5 ドライバ部の Upper Limit 設定例

```
__STATIC_INLINE void Cy_WDT_SetUpperLimit(uint32_t match)
{
    WDT->unUPPER_LIMIT.stcField.u32UPPER_LIMIT = match;   /*  (4) Set UPPER_LIMIT */
}
```

Code Listing 6 ドライバ部の Warn Limit 設定例

```
__STATIC_INLINE void Cy_WDT_SetWarnLimit(uint32_t match)
{
    WDT->unWARN_LIMIT.stcField.u32WARN_LIMIT = match;   /*  (5) Set WARN_LIMIT */
}
```

Code Listing 7 ドライバ部の Lower Action 設定例

```
   typedef enum
{
    CY_WDT_LOW_UPP_ACTION_NONE,
    CY_WDT_LOW_UPP_ACTION_RESET
} cy_en_wdt_lower_upper_action_t;

__STATIC_INLINE void Cy_WDT_SetLowerAction(cy_en_wdt_lower_upper_action_t action)
{
    WDT->unCONFIG.stcField.u1LOWER_ACTION = action;   /*  (6) Set LOWER_ACTION */
}
```

Code Listing 8 ドライバ部の Upper Action 設定例

```
__STATIC_INLINE void Cy_WDT_SetUpperAction(cy_en_wdt_lower_upper_action_t action)
{
    WDT->unCONFIG.stcField.u1UPPER_ACTION = action;   /*  (7) Set UPPER_ACTION */
}
```

Code Listing 9 ドライバ部の Warn Action 設定例

```
typedef enum
{
    CY_WDT_WARN_ACTION_NONE,
    CY_WDT_WARN_ACTION_INT
} cy_en_wdt_warn_action_t;

__STATIC_INLINE void Cy_WDT_SetWarnAction(cy_en_wdt_warn_action_t action)   /*  (8) Set
WARN_ACTION */
{
    WDT->unCONFIG.stcField.u1WARN_ACTION = action;
}
```

Code Listing 10 ドライバ部の Auto Service 設定例

```
typedef enum
{
    CY_WDT_DISABLE,
    CY_WDT_ENABLE
} cy_en_wdt_enable_t;

__STATIC_INLINE void Cy_WDT_SetAutoService(cy_en_wdt_enable_t enable)
{
    WDT->unCONFIG.stcField.u1AUTO_SERVICE = enable;   /*  (9) Configure Auto Service */
}
```

Code Listing 11 ドライバ部のデバッガ設定例

```
__STATIC_INLINE void Cy_WDT_SetDebugRun(cy_en_wdt_enable_t enable)
{
    WDT->unCONFIG.stcField.u1DEBUG_RUN = enable;   /8  (10) Set Debugger Configuration */
}
```

Code Listing 12 ドライバ部のベーシック WDT ロック例

```
void Cy_WDT_Lock(void)
{
    uint32_t interruptState;
    interruptState = Cy_SysLib_EnterCriticalSection();

    WDT->unLOCK.stcField.u2WDT_LOCK = 3ul;   /*  (11) Lock Basic WDT registers during
interrupts disabled */

    Cy_SysLib_ExitCriticalSection(interruptState);
}
```

Code Listing 13 ドライバ部の WDT 割込みイネーブル例

```
__STATIC_INLINE void Cy_WDT_MaskInterrupt(void)
{
    WDT->unINTR_MASK.stcField.u1WDT = 1ul;   /*  (12) Enable WDT Interrupt- */
}
```

Code Listing 14 ドライバ部のベーシック WDT イネーブル例

```
void Cy_WDT_Enable(void)
{
    Cy_WDT_Unlock();
    WDT->unCTL.stcField.u1ENABLE = 1ul;   /*  (13) Enable Basic WDT during WDT unlocked */
    Cy_WDT_Lock();
}
```

2.8 ベーシック WDT のクリア

ベーシック WDT のクリアは、SERVICE レジスタの SERVICE[0]ビットを'1'に設定することによって実行されます。ファームウェアは、このビットに'1'を書き込む前に、このビットが'0'になるまで読み出す必要があります。

ベーシック WDT カウンタの保守は、安定したソフトウェアフローを確保するために定期的に行う必要があります。使用されるソフトウェアコンセプトとは無関係に、ソフトウェアコンポーネントのランタイム計算は、クリアされるカウンタの閾値を定義するために重要です。ウィンドウモードは、ソフトウェアがベーシック WDT 処理として予期していない最小期間を決定する必要があり、さらに複雑になります。この最小期間は、例えば、優先度の低い主機能の最小実行時間とすることができます。

図 6 は、タスクが異なるシステム内でウォッチドッグカウンタをクリアできる場合の例を示します。各サービスの計算では、次の条件を考慮する必要があります。

1. ウィンドウモードでは、カウンタが LOWER_LIMIT に達する前にウォッチドッグを処理しないでください。
2. リセットイベントを回避するため、UPPER_LIMIT に達する前にウォッチドッグを処理する必要があります。

以下の条件が定義されています。

- UPPER_LIMIT = 0x8000: 上限リセットの閾値は 1 秒です
- LOWER_LIMIT = 0x1000: 下限リセットの閾値は 125 ミリ秒です

- Task 1 期間: 100ms
- Task 2 期間: 300ms
- Task 3 期間: 200ms
- Task 4 期間: 150ms
- Task 5 期間: 200ms

異なるタイミングを想定した異なるシーケンスがあります。

- シーケンス 1: $t_{Task1} + t_{Task2} + t_{Task3} + t_{Task4}$ = 100 ms + 300 ms + 200 ms + 150 ms = 750 ms
- シーケンス 2: $t_{Task1} + t_{Task4}$ = 100 ms + 150 ms = 250 ms
- シーケンス 3: $t_{Task1} + t_{Task4} + t_{Task5}$ = 100 ms + 150 ms + 200 ms = 450 ms

すべてのケースで次の条件が満たされます。

$t_{LOWER\_LIMIT} < t_{SEQUENCE} < t_{UPPER\_LIMIT}$

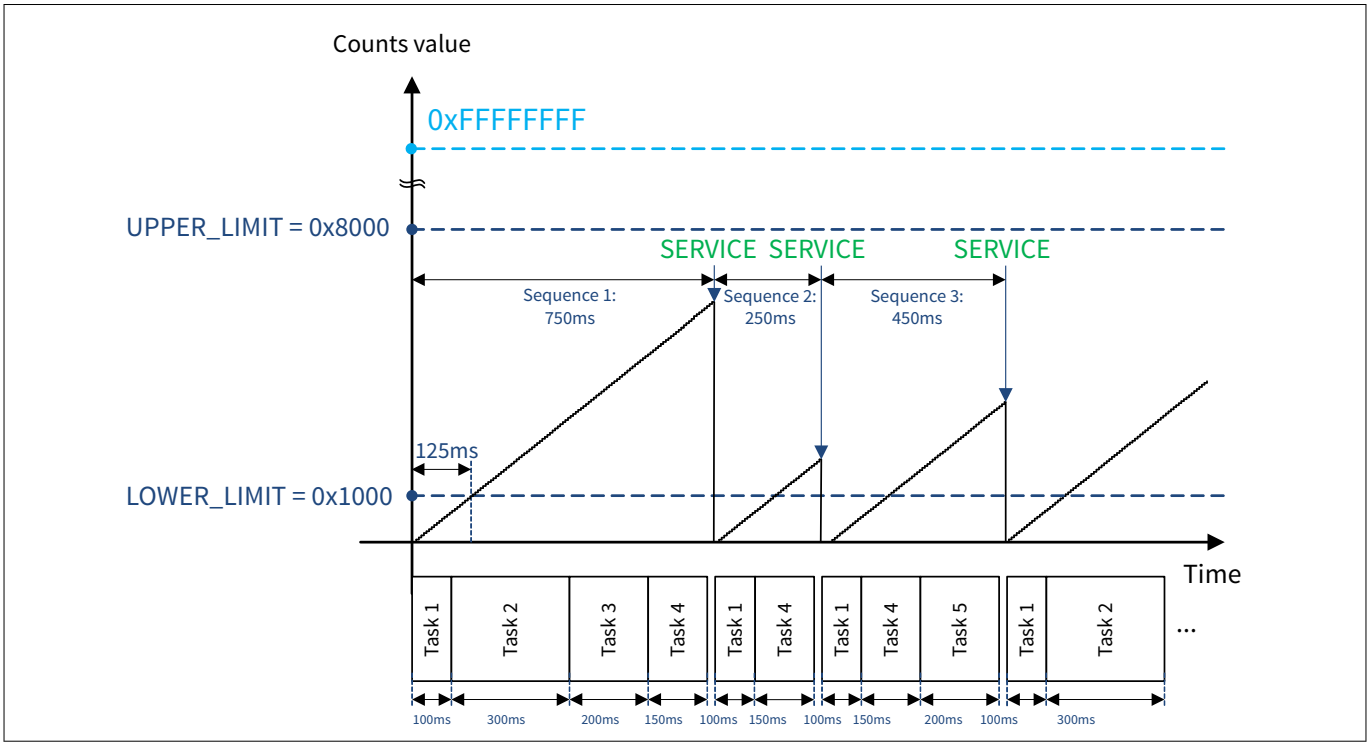

図 6　　ウィンドウモードでのベーシック WDT の処理例

2.8.1　使用事例

ここでは、第 2.7.1 章 使用事例で説明した使用例を使用してベーシック WDT のクリア例について説明します。

2.8.2　ベーシック WDT クリアのフロー例

図 7 に、ベーシック WDT をクリアするフロー例を示します。

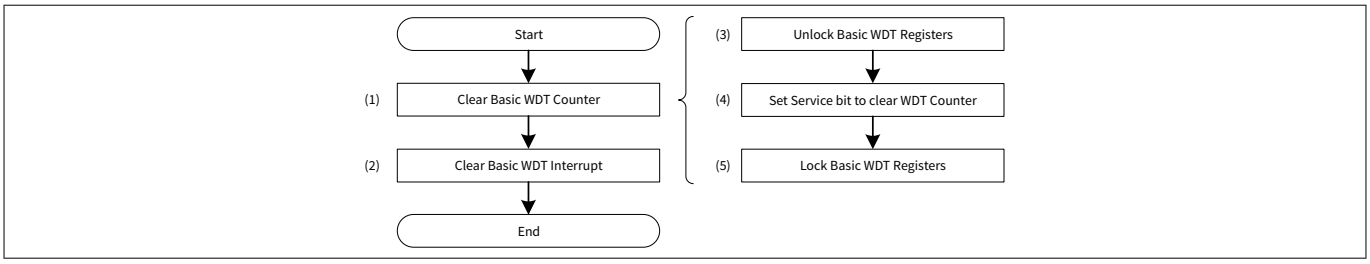

図 7　　ベーシック WDT クリアのフロー例

2.8.3　　ベーシック WDT クリアのプログラム例

Code Listing 15 に、ベーシック WDT をクリアするプログラム例を示します。

Code Listing 15 ベーシック WDT クリアのプログラム例

```
void Wdt_Warn_IntrISR(void)
{
    Cy_WDT_ClearWatchdog();    /* (1) Clear Basic WDT Counter */
    Cy_WDT_ClearInterrupt();    /* (2) Clear Basic WDT Interrupt */
}
```

Code Listing 16 に、ドライバ部のベーシック WDT をクリアするプログラム例を示します。

Code Listing 16 ドライバ部のベーシック WDT クリアのプログラム例

```
void Cy_WDT_ClearWatchdog(void)
{
    Cy_WDT_Unlock();    /* (3) Unlock Basic WDT Registers */
    Cy_WDT_SetService();
    Cy_WDT_Lock();    /* (5) Unlock Basic WDT Registers */
}

__STATIC_INLINE void Cy_WDT_SetService()
{
    WDT->unSERVICE.stcField.u1SERVICE = 1ul;    /* (4) Set Service bit to clear Basic WDT
Counter */
}
```

Code Listing 17 に、ドライバ部のベーシック WDT 割込みをクリアするプログラム例を示します。

Code Listing 17 ドライバ部のベーシック WDT 割込みクリアのプログラム例

```
void Cy_WDT_ClearInterrupt(void)
{
    WDT->unINTR.stcField.u1WDT = 1ul;    /* (2) Clear Basic WDT Interrupt */

    /* Read the interrupt register to ensure that the initial clearing write has
    * been flushed out to the hardware.
    */
    (void) SRSS->unSRSS_INTR;
}
```

2.9 ベーシック WDT のリセット要因表示

ベーシック WDT が処理されない、または早すぎる場合、システム全体のリセットが発行されます。リセットイベントは、RES_CAUSE レジスタの RESET_WDT[0] ビットに格納されます。ハードウェアは、パワーオンリセット (POR) によって、このビットをクリアすることに注意してください。リセットが LOWER_LIMIT または UPPER_LIMIT 違反によって引き起こされたかの区別はできません。

2.10 ベーシック WDT レジスタ

表 2 ベーシック WDT レジスタ

名称	説明
WDT_CTL	ウォッチドッグカウンタ制御レジスタ
WDT_LOWER_LIMIT	WDT 下限レジスタ
WDT_UPPER_LIMIT	WDT 上限レジスタ
WDT_WARN_LIMIT	WDT 警告レジスタ
WDT_CONFIG	WDT コンフィギュレーション レジスタ
WDT_CNT	WDT カウントレジスタ
WDT_LOCK	WDT ロックレジスタ
WDT_SERVICE	WDT サービスレジスタ
WDT_INTR	WDT 割込みレジスタ
WDT_INTR_SET	WDT 割込みセット レジスタ
WDT_INTR_MASK	WDT 割込みマスクレジスタ
WDT_INTR_MASKED	WDT 割込みマスクレジスタ
CLK_SELECT	クロック選択レジスタ
CLK_ILO_CONFIG	ILO の構成
RES_CAUSE	リセット要因レジスタ

3 マルチカウンター WDT

MCWDT には 3 つのサブカウンタがあります: Subcounters 0, 1, および 2。

Subcounter 0 と Subcounter 1 は、ベーシック WDT のように動作する 16 ビットのカウンタです。ウィンドウモードと事前警告割込みがサポートされています。ウィンドウタイミング違反が発生した場合、タイムアウトのタイミング内で処理されないと、FAULT または FAULT 後にリセットを生成します。

Subcounter 2 は 32 ビットのカウンタで、あらかじめ定義されたカウンタビットの 1 つがトグルしたときに割込みを生成するように設定できます。両方のタイプのカウンタは、Active, Sleep, および DeepSleep モードで動作します。Hibernate モードでは使用できません。

3 つのサブカウンタすべてを備えた MCWDT のブロックダイヤグラムを図 8 に示します。

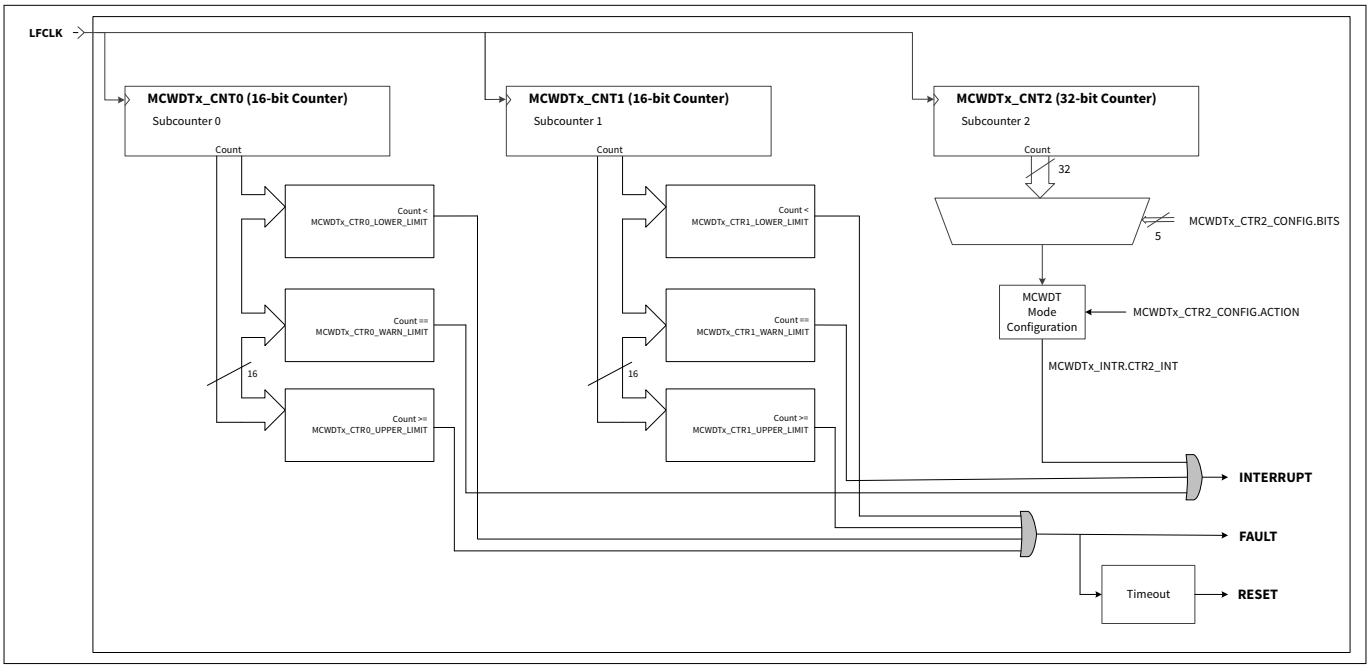

図 8 マルチカウンタ WDT ブロックダイヤグラム

3.1 ソースクロック

MCWDT で選択できるソースクロックは LFCLK で、以下のクロックソースのいずれかです。

- ILO0/1: 比較的低精度の内部低速発振器 (公称値: 32.768 kHz)
- WCO: 低周波時計用水晶発振器 (公称値: 32.768 kHz)
- ECO: 高周波水晶発振器 (公称値: 4 – 33.33 MHz)

3.2 MCWDT のレジスタ保護

MCWDT に関連するレジスタを変更するには、LOCK レジスタにある MCWDT_LOCK[1:0] ビットの UNLOCK シーケンスが必要です。アンロックのためには、以下のアクセスシーケンスが必要です。

Subcounter 2: CTR2_CTL, CTR2_CONFIG, および CTR2_CNT レジスタ

Subcounter 0 および Subcounter 1: CTL, LOWER_LIMIT, UPPER_LIMIT, WARN_LIMIT, CONFIG, SERVICE, および CNT レジスタ

- MCWDT_LOCK = 1
- MCWDT_LOCK = 2

MCWDT レジスタを保護するには、LOCK レジスタへの 1 回の書込みアクセスが必要です。

- MCWDT_LOCK = 3

3.3 MCWDT 割込み

MCWDT は、さまざまな種類の割込みをサポートします。

3.3.1 事前警告割込み

Subcounter 0 と Subcounter 1 はベーシック WDT の事前警告割込みと似た動作をします。第 2.4 章 警告割込みを参照してください。唯一の違いは、WARN_LIMIT が 16 ビット値であることです。割込みタイミングは次の式で計算されます。

$$t_{WARN\_IRQ} = \frac{WARN\_LIMIT}{f_{LFCLK}}$$

図 9

この割込みは、FAULT イベントが発行される前に MCWDT カウンタを処理する必要があることを示す事前警告イベントとして使用できます。

CONFIG レジスタの WARN_ACTION[8] ビットを'1'に設定すると、関連する CPU に対して割込みがトリガできます。

MCWDT は CONFIG レジスタの AUTO_SERVICE[12] ビットによって自動的に処理されます。これにより、このカウンタがウォッチドッグとして必要のない場合、定期的な割込みを作成できます。

3.3.2 MCWDT Subcounter 2 割込み

Subcounter 2 割込みは、異なる動作をします。専用の事前に定義されたカウンタビットがトグルすると、割込みタイミングが生成されます。割込みタイミングは、次の式で計算されます。

$$t_{IRQ} = 2^n \frac{1}{f_{LFCLK}}$$

図 10

例:

LFCLK = ILO0 = 32.768 kHz

Toggle-Bit = Bit 12

$$t_{IRQ} = 2^{12} \frac{1}{32768} = 125\ ms$$

図 11

Toggle-Bit は、CTR2_CONFIG レジスタの BITS[20:16] によって設定されます。CTR2_CONFIG レジスタの ACTION[0] ビットが'1'に設定されているときに、関連する CPU に対して割込みがトリガされます。

3.4 タイムアウトモード

このモードは、Subcounter 0 と Subcounter 1 のみに関連し、ベーシック WDT と似ています。第 2.5 章 タイムアウトモードを参照してください。違いは、UPPER_LIMIT が 16 ビット値であることです。サブカウンタが UPPER_LIMIT 値と一致すると、FAULT ストラクチャで処理される FAULT が生成されます。

CONFIG レジスタの UPPER_ACTION[1:0]ビットフィールドで、FAULT の処理方法を指定します。

- 何もしない

- FAULT ストラクチャによって処理される FAULT のみを生成
- FAULT を生成し、この FAULT が 3 クロックサイクル未満に処理されない場合は RESET をトリガします

3.5 ウインドウモード

このモードは、Subcounter 0 と Subcounter 1 のみに関連し、ベーシック WDT と似ています。第 2.6 章 ウインドウモードを参照してください。違いは、LOWER_LIMIT が 16 ビット値であることです。カウンタが LOWER_LIMIT 値に達する前にサブカウンタが処理された場合、FAULT が FAULT ストラクチャで処理されるように生成されます。

CONFIG レジスタの UPPER_ACTION[5:4] および LOWER_ACTION[1:0] ビットフィールドで、FAULT の処理方法を次のように指定します。

- 何もしない
- FAULT ストラクチャによって処理される FAULT のみを生成
- FAULT を生成し、この FAULT が 3 クロックサイクル未満に処理されない場合は、RESET をトリガします

FAULT_THEN_RESET が LOWER_ACTION および UPPER_ACTION として選択されたときのウィンドウモードを図 12 に示します。LOWER_ACTION, WARN_ACTION, および UPPER_ACTION がアクティブ化されている場合、それに応じて 4 つのシナリオが考えられます。

- LOWER_LIMIT と WARN_LIMIT の間でカウンタを処理: これは MCWDT の通常の動作です。WARN 割込みは発行されず、RESET も行われません。
- WARN_LIMIT と UPPER_LIMIT の間でカウンタを処理: 処理は遅れて終了します。WARN 割込みが発行されますが、RESET は実行されません。
- すべてのカウンタ処理なし: WARN 割込みが発行されますが、それでも CTR0/1_SERVICE ビットは設定されません。カウンタが UPPER_LIMIT に達すると、FAULT が発行されます。ファームウェアがこの FAULT を処理してシステムを安全な状態に戻さない場合、固定数の LFCLK サイクル後に RESET が発行されます。
- LOWER_LIMIT に達する前にカウンタを処理: カウンタの処理が早すぎます。FAULT が発行され、続いて FAULT がファームウェアによって時間内に処理されない場合には RESET が発行されます。

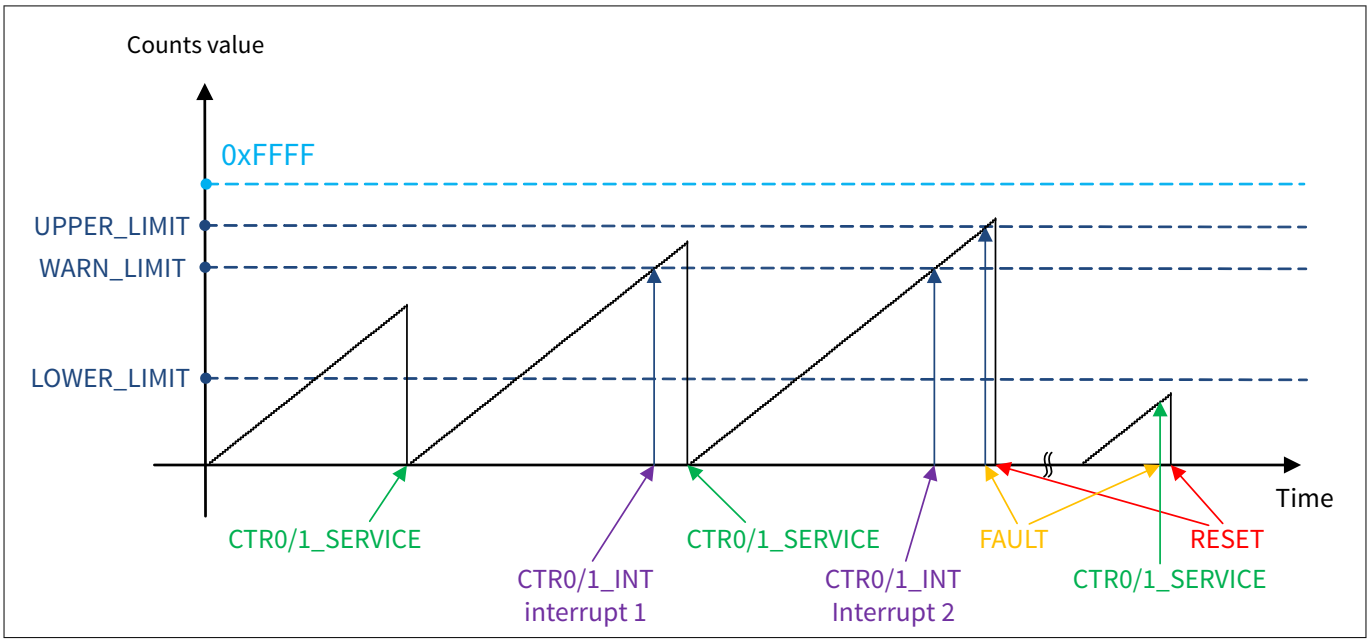

図 12 ウィンドウモードでの Subcounter 0/1 の FAULT および RESET 動作

3.6 CPU 選択

マルチ CPU システムでは、1 つの MCWDT を専用 CPU に割り当てて、それぞれの CPU 低電力モードでのカウンタ動作を制御する SLEEPDEEP を選択する必要があります。CTR2_CONFIG レジスタで SLEEPDEEP_PAUSE[30] ビットが‘1’に設定されている場合、各 CPU が低電力モードにある間、カウンタは一時停止します。

調整が複雑なため、1 つの MCWDT を複数の CPU で同時に使用することはできません。

CPU_SELECT レジスタの CPU_SEL[1:0] ビットは、表 3 で定義されています。

表 3 CPU への MCWDT 割当て

CPU_SEL[0:4]	CYT2 CPU	CYT3 CPU	CYT4 CPU	CYT6 CPU
0	CM0+	CM0+	CM0+	CM0+
1	CM4	CM7-0	CM7-0	CM7-0
2	-	-	CM7-1	CM7-1
3	-	-	-	CM7-2

***注**: CYT6BJ は CM7_3 コアです。しかし、CM7_3 はどの MCWDT にも接続されていません。*

3.7 MCWDT 設定

図 13 に、MCWDT を設定するフロー例を示します。

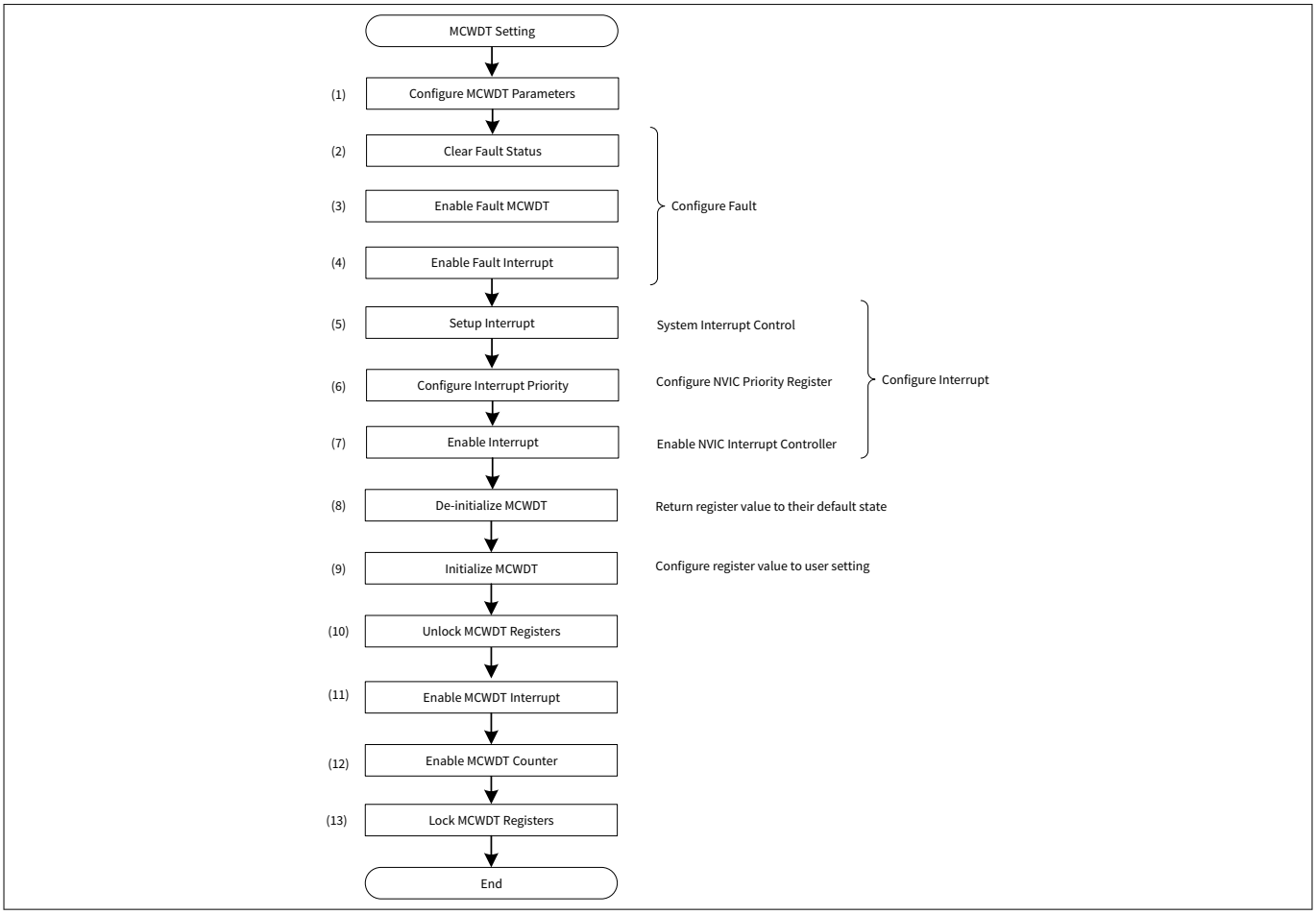

図 13 マルチカウンタ WDT 設定手順

3.7.1 使用事例

ここでは、次の使用例によって MCWDT の例を説明します。MCWDT はメインタスクループでクリアされます。MCWDT が UPPER_LIMIT 以内にクリアされない場合、フォールト割込みがトリガされます。

使用例:

- MCWDT 番号: 1
- CPU: CM4
- Subcounter 0
  - LOWER_LIMIT: 未使用
  - UPPER_LIMIT: 1 秒
  - WARN_LIMIT: 未使用
  - Window mode: 未使用
  - 上限動作: フォールト割込み (IRQ 番号: 2)
  - Auto service: 未使用
  - デバッガ設定: デバッグモード中にカウンタを一時停止するために MCWDT のトリガ入力をイネーブルします
- Subcounter 1: 未使用
- Subcounter 2: 未使用
- Fault report: Fault structure 1

3.7.2 MCWDT 設定

表 4 に、MCWDT の SDL 設定部のパラメータを示します。

表 4 MCWDT パラメータリスト

パラメータ	説明	値
`.coreSelect`	SleepDeepPause に使用する CPU を選択 CY_MCWDT_PAUSED_BY_DPSLP_CM0 = 0ul CY_MCWDT_PAUSED_BY_DPSLP_CM4_CM7_0 = 1ul CY_MCWDT_PAUSED_BY_DPSLP_CM7_1 = 2ul CY_MCWDT_PAUSED_BY_NO_CORE = 3ul	CY_MCWDT_PAUSED_BY_DPSLP_CM4_CM7_0
`.c0LowerLimit`	Subcounter 0 下限設定 (符号なし整数 32 ビット)	0ul
`.c0UpperLimit`	Subcounter 0 上限設定 (符号なし整数 32 ビット)	32768ul
`.c0WarnLimit`	Subcounter 0 警告限界設定 (符号なし整数 32 ビット)	0ul
`.c0LowerAction`	Subcounter 0 下位アクションを“no action”, “fault”, または“fault then reset”に設定: CY_MCWDT_ACTION_NONE = 0ul CY_MCWDT_ACTION_FAULT = 1ul CY_MCWDT_ACTION_FAULT_THEN_RESET = 2ul	CY_MCWDT_ACTION_NONE

(続く)

表 4 (続き) MCWDT パラメータリスト

パラメータ	説明	値
.c0UpperAction	Subcounter 0 上位アクションを “no action”, “fault”, または“fault then reset”に設定: CY_MCWDT_ACTION_NONE = 0ul CY_MCWDT_ACTION_FAULT = 1ul CY_MCWDT_ACTION_FAULT_THEN_RESET = 2ul	CY_MCWDT_ACTION_FAULT
.c0WarnAction	Subcounter 0 警告アクションを “no action”, または“interrupt”に設定: CY_MCWDT_WARN_ACTION_NONE = 0ul CY_MCWDT_WARN_ACTION_INT = 1ul	CY_MCWDT_WARN_ACTION_NONE
.c0AutoService	Subcounter 0 の値が WARN_LIMIT に達したときに MCWDT を自動的にクリアするように設定: CY_MCWDT_DISABLE = 0ul CY_MCWDT_ENABLE = 1ul	CY_MCWDT_DISABLE
.c0SleepDeepPause	対応する CPU が DeepSleep にあるときに Subcounter 0 を一時停止できるようにする: CY_MCWDT_DISABLE = 0ul CY_MCWDT_ENABLE = 1ul	CY_MCWDT_ENABLE
.c0DebugRun	デバッガを設定デバッガを使用する場合に必要 CY_MCWDT_DISABLE = 0ul CY_MCWDT_ENABLE = 1ul	CY_MCWDT_ENABLE
.c1LowerLimit	Subcounter 1 下限設定 (符号なし整数 32 ビット)	0ul
.c1UpperLimit	Subcounter 1 上限設定 (符号なし整数 32 ビット)	0ul
.c1WarnLimit	Subcounter 1 警告限界設定 (符号なし整数 32 ビット)	0ul
.c1LowerAction	Subcounter 1 下限アクションは “no action”, “fault”, または“fault then reset”に設定: CY_MCWDT_ACTION_NONE = 0ul CY_MCWDT_ACTION_FAULT = 1ul CY_MCWDT_ACTION_FAULT_THEN_RESET = 2ul	CY_MCWDT_ACTION_NONE
.c1UpperAction	Subcounter 1 上限アクションは “no action”, “fault”, または“fault then reset”に設定: CY_MCWDT_ACTION_NONE = 0ul CY_MCWDT_ACTION_FAULT = 1ul CY_MCWDT_ACTION_FAULT_THEN_RESET = 2ul	CY_MCWDT_ACTION_NONE
.c1WarnAction	Subcounter 1 警告アクションは “no action”, または“interrupt”に設定: CY_MCWDT_WARN_ACTION_NONE = 0ul CY_MCWDT_WARN_ACTION_INT = 1ul	CY_MCWDT_WARN_ACTION_NONE

(続く)

表 4 (続き) MCWDT パラメータリスト

パラメータ	説明	値
.c1AutoService	Subcounter 1 の値が WARN_LIMIT に達したときに MCWDT を自動的にクリアするように設定: CY_MCWDT_DISABLE = 0ul CY_MCWDT_ENABLE = 1ul	CY_MCWDT_DISABLE
.c1SleepDeepPause	対応する CPU が DeepSleep にあるときに Subcounter 1 を一時停止できるようにする: CY_MCWDT_DISABLE = 0ul CY_MCWDT_ENABLE = 1ul	CY_MCWDT_DISABLE
.c1DebugRun	デバッガを設定 (デバッガを使用する場合に必要) CY_MCWDT_DISABLE = 0ul CY_MCWDT_ENABLE = 1ul	CY_MCWDT_DISABLE

(続く)

表 4 (続き) MCWDT パラメータリスト

パラメータ	説明	値
.c2ToggleBit	トグルを監視するビットを選択: CY_MCWDT_CNT2_MONITORED_BIT0 = 0ul CY_MCWDT_CNT2_MONITORED_BIT1 = 1ul CY_MCWDT_CNT2_MONITORED_BIT2 = 2ul CY_MCWDT_CNT2_MONITORED_BIT3 = 3ul CY_MCWDT_CNT2_MONITORED_BIT4 = 4ul CY_MCWDT_CNT2_MONITORED_BIT5 = 5ul CY_MCWDT_CNT2_MONITORED_BIT6 = 6ul CY_MCWDT_CNT2_MONITORED_BIT7 = 7ul CY_MCWDT_CNT2_MONITORED_BIT8 = 8ul CY_MCWDT_CNT2_MONITORED_BIT9 = 9ul CY_MCWDT_CNT2_MONITORED_BIT10 = 10ul CY_MCWDT_CNT2_MONITORED_BIT11 = 11ul CY_MCWDT_CNT2_MONITORED_BIT12 = 12ul CY_MCWDT_CNT2_MONITORED_BIT13 = 13ul CY_MCWDT_CNT2_MONITORED_BIT14 = 14ul CY_MCWDT_CNT2_MONITORED_BIT15 = 15ul CY_MCWDT_CNT2_MONITORED_BIT16 = 16ul CY_MCWDT_CNT2_MONITORED_BIT17 = 17ul CY_MCWDT_CNT2_MONITORED_BIT18 = 18ul CY_MCWDT_CNT2_MONITORED_BIT19 = 19ul CY_MCWDT_CNT2_MONITORED_BIT20 = 20ul CY_MCWDT_CNT2_MONITORED_BIT21 = 21ul CY_MCWDT_CNT2_MONITORED_BIT22 = 22ul CY_MCWDT_CNT2_MONITORED_BIT23 = 23ul CY_MCWDT_CNT2_MONITORED_BIT24 = 24ul CY_MCWDT_CNT2_MONITORED_BIT25 = 25ul CY_MCWDT_CNT2_MONITORED_BIT26 = 26ul CY_MCWDT_CNT2_MONITORED_BIT27 = 27ul CY_MCWDT_CNT2_MONITORED_BIT28 = 28ul CY_MCWDT_CNT2_MONITORED_BIT29 = 29ul CY_MCWDT_CNT2_MONITORED_BIT30 = 30ul CY_MCWDT_CNT2_MONITORED_BIT31 = 31ul	CY_MCWDT_CNT2_MONITORED_BIT0
.c2Action	Subcounter 2 のアクションを“no action” または “interrupt”に設定: CY_MCWDT_CNT2_ACTION_NONE = 0ul CY_MCWDT_CNT2_ACTION_INT = 1ul	CY_MCWDT_CNT2_ACTION_NONE

(続く)

表 4 (続き) MCWDT パラメータリスト

パラメータ	説明	値
.c2SleepDeepPause	対応する CPU が DeepSleep にあるときに Subcounter 2 を一時停止できるようにする: CY_MCWDT_DISABLE = 0ul CY_MCWDT_ENABLE = 1ul	CY_MCWDT_DISABLE
.c2DebugRun	デバッガを設定 (デバッガを使用する場合に必要) CY_MCWDT_DISABLE = 0ul CY_MCWDT_ENABLE = 1ul	CY_MCWDT_DISABLE

Code Listing 18 に、MCWDT 設定部のプログラム例を示します。割込みとフォールトの初期設定手順の詳細については、関連ドキュメントに記載されている AN219842 の“Interrupt and Fault Report Structure”章を参照してください。

Code Listing 18 MCWDT を設定するプログラム例

```
cy_stc_sysint_irq_t irq_cfg =
{
    .sysIntSrc  = cpuss_interrupts_fault_1_IRQn,
    .intIdx     = CPUIntIdx2_IRQn,
    .isEnabled  = true,
};
cy_stc_mcwdt_config_t mcwdtConfig =    /*  (1) Configure MCWDT Parameters  */
{
    .coreSelect       = CY_MCWDT_PAUSED_BY_DPSLP_CM4_CM7_0,    /*  Select CPU to be used for
SleepDeepPause.  */
    .c0LowerLimit     = 0,    /*  Configure WDT Subcounter 0 Parameters  */
    .c0UpperLimit     = 32768,    /*  Configure WDT Subcounter 0 Parameters  */
    .c0WarnLimit      = 0,    /*  Configure WDT Subcounter 0 Parameters  */
    .c0LowerAction    = CY_MCWDT_ACTION_NONE,    /*  Configure WDT Subcounter 0 Parameters  */
    .c0UpperAction    = CY_MCWDT_ACTION_FAULT,    /*  Configure WDT Subcounter 0 Parameters  */
    .c0WarnAction     = CY_MCWDT_WARN_ACTION_NONE,    /*  Configure WDT Subcounter 0
Parameters  */
    .c0AutoService    = CY_MCWDT_DISABLE,    /*  Configure WDT Subcounter 0 Parameters  */
    .c0SleepDeepPause = CY_MCWDT_ENABLE,    /*  Configure WDT Subcounter 0 Parameters  */
    .c0DebugRun       = CY_MCWDT_ENABLE,    /*  Configure WDT Subcounter 0 Parameters  */
    .c1LowerLimit     = 0,    /*  Configure WDT Subcounter 1 Parameters  */
    .c1UpperLimit     = 0,    /*  Configure WDT Subcounter 1 Parameters  */
    .c1WarnLimit      = 0,    /*  Configure WDT Subcounter 1 Parameters  */
    .c1LowerAction    = CY_MCWDT_ACTION_NONE,    /*  Configure WDT Subcounter 1 Parameters  */
    .c1UpperAction    = CY_MCWDT_ACTION_NONE,    /*  Configure WDT Subcounter 1 Parameters  */
    .c1WarnAction     = CY_MCWDT_WARN_ACTION_NONE,    /*  Configure WDT Subcounter 1
Parameters  */
    .c1AutoService    = CY_MCWDT_DISABLE,    /*  Configure WDT Subcounter 1 Parameters  */
    .c1SleepDeepPause = CY_MCWDT_DISABLE,    /*  Configure WDT Subcounter 1 Parameters  */
    .c1DebugRun       = CY_MCWDT_DISABLE,    /*  Configure WDT Subcounter 1 Parameters  */
    .c2ToggleBit      = CY_MCWDT_CNT2_MONITORED_BIT0,    /*  Configure WDT Subcounter 2
Parameters  */
    .c2Action         = CY_MCWDT_CNT2_ACTION_NONE,    /*  Configure WDT Subcounter 2
Parameters  */
    .c2SleepDeepPause = CY_MCWDT_DISABLE,    /*  Configure WDT Subcounter 2 Parameters  */
    .c2DebugRun       = CY_MCWDT_DISABLE,    /*  Configure WDT Subcounter 2 Parameters  */
};
int main(void)
{
    SystemInit();
    __enable_irq(); /* Enable global interrupts. */
    /*******************************************************************/
    /*****                 Fault report settings                   *****/
    /*******************************************************************/
    Cy_SysFlt_ClearStatus(FAULT_STRUCT1);    /* (2) Clear Fault Status. */
    Cy_SysFlt_SetMaskByIdx(FAULT_STRUCT1, CY_SYSFLT_SRSS_MCWDT1);    /* (3) Enable Fault MCWDT.
*/
    Cy_SysFlt_SetInterruptMask(FAULT_STRUCT1);    /* (4) Enable Fault Interrupt. */
    /*******************************************************************/
    /*****                   Interrupt setting                     *****/
```

```
    /*****************************************************************/
    Cy_SysInt_InitIRQ(&irq_cfg);    /* (5) Setup Interrupt. */
    Cy_SysInt_SetSystemIrqVector(irq_cfg.sysIntSrc, irqFaultReport1Handler);
    NVIC_SetPriority(CPUIntIdx2_IRQn, 0);    /* (6) Configure Interrupt Priority. */
    NVIC_EnableIRQ(CPUIntIdx2_IRQn);    /* (7) Enable Interrupt. */
    /*****************************************************************/
    /*****                Configuration for MCWDT                *****/
    /*****************************************************************/
    Cy_MCWDT_DeInit(MCWDT1);    /* (8) De-initialize MCWDT. */
    Cy_MCWDT_Init(MCWDT1, &mcwdtConfig);    /* (9) Initialize MCWDT. */
    Cy_MCWDT_Unlock(MCWDT1);    /* (10) Unlock MCWDT. */
    Cy_MCWDT_SetInterruptMask(MCWDT1, CY_MCWDT_CTR0);    /* (11) Enable MCWDT Interrupt. */
    Cy_MCWDT_Enable(MCWDT1,
                    CY_MCWDT_CTR0,    /* (12) Enable MCWDT Counter. */
                    0);
    Cy_MCWDT_Lock(MCWDT1);    /* (13) Unlock MCWDT. */
    for(;;)
    {
    :
    }
}
```

3.7.3 ドライバ部の MCWDT 設定プログラム例

Code Listing 19～Code Listing 24 に、MCWDT を設定するドライバ部のプログラム例を示します。

以下は、SDL のドライバ部分のレジスタ表記を説明します。

- base は、MCWDT レジスタのベースアドレスへのポインタを示します。counters は、MCWDT 内の Subcounter を指定します。表 5 参照
- レジスタ設定手順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、事前にビット書込み可能-なバッファに生成され、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。

```
tempCNT2ConfigParams.stcField.u5BITS    = config->c2ToggleBit;
tempCNT2ConfigParams.stcField.u1ACTION                  = config->c2Action;
tempCNT2ConfigParams.stcField.u1SLEEPDEEP_PAUSE         = config->c2SleepDeepPause;
tempCNT2ConfigParams.stcField.u1DEBUG_RUN               = config->c2DebugRun;
base->unCTR2_CONFIG.u32Register                         = tempCNT2ConfigParams.u32Register;
```

レジスタ表記の結合および構造表現の詳細については、`hdr/rev_x/ip` の `cyip_srss_v2.h` を参照してください。

表 5 ドライバ部の MCWDT パラメータリスト

パラメータ	説明	値
base	MCWDT 番号を指定して、そのレジスタを設定 MCWDT0 MCWDT1 MCWDT2 (CYT4 用のみ) MCWDT3 (CYT6 用のみ)	MCWDT1

(続く)

表 5 (続き) ドライバ部の MCWDT パラメータリスト

パラメータ	説明	値
カウンタ	レジスタを設定する Subcounter を指定: CY_MCWDT_CTR0: Subcounter 0 CY_MCWDT_CTR1: Subcounter 1 CY_MCWDT_CTR2: Subcounter 2 CY_MCWDT_CTR_Msk: すべての Subcounters	CY_MCWDT_CTR0

Code Listing 19 ドライバ部の MCWDT 初期化解除プログラム例

```
/*  (8) De-initializes the MCWDT block, returns register values to their default state. */
void Cy_MCWDT_DeInit(volatile stc_MCWDT_t *base)
{
    Cy_MCWDT_Unlock(base);    /* Unlock MCWDT Registers */

    // disable all counter
    for(uint32_t loop = 0ul; loop < CY_MCWDT_NUM_OF_SUBCOUNTER; loop++)
    {
        base->CTR[loop].unCTL.u32Register = 0ul;
    }
    base->unCTR2_CTL.u32Register    = 0ul;

    for(uint32_t loop = 0ul; loop < CY_MCWDT_NUM_OF_SUBCOUNTER; loop++)
    {
        while(base->CTR[loop].unCTL.u32Register != 0x0ul); // wait until enabled bit become 1
        base->CTR[loop].unLOWER_LIMIT.u32Register = 0x0ul;
        base->CTR[loop].unUPPER_LIMIT.u32Register = 0x0ul;
        base->CTR[loop].unWARN_LIMIT.u32Register  = 0x0ul;
        base->CTR[loop].unCONFIG.u32Register      = 0x0ul;
        base->CTR[loop].unCNT.u32Register         = 0x0ul;
    }

    while(base->unCTR2_CNT.u32Register != 0ul); // wait until enabled bit become 1
    base->unCPU_SELECT.u32Register  = 0ul;
    base->unCTR2_CONFIG.u32Register = 0ul;
    base->unSERVICE.u32Register     = 0x00000003ul;
    base->unINTR.u32Register        = 0xFFFFFFFFul;
    base->unINTR_MASK.u32Register   = 0ul;

    Cy_MCWDT_Lock(base);    /* Lock MCWDT Registers */
}
```

Code Listing 20 ドライバ部の MCWDT 初期化プログラム例

```
/*  (9) Initializes the MCWDT block according to the MCWDT configuration  */
cy_en_mcwdt_status_t Cy_MCWDT_Init(volatile stc_MCWDT_t *base, cy_stc_mcwdt_config_t const
*config)
{
    cy_en_mcwdt_status_t ret = CY_MCWDT_BAD_PARAM;
    if ((base != NULL) && (config != NULL))   /*  Validate configuration parameter */
    {
        Cy_MCWDT_Unlock(base);   /*  Unlock MCWDT Registers */
        un_MCWDT_CTR_CONFIG_t  tempConfigParams     = { 0ul };
        un_MCWDT_CTR2_CONFIG_t tempCNT2ConfigParams = { 0ul };

       /*  Configure CPU to be used for SLEEPDEEP_PAUSE.  */
        base->unCPU_SELECT.u32Register                    = config->coreSelect;


       /*  Configure Subcounter 0  */
        base->CTR[0].unLOWER_LIMIT.stcField.u16LOWER_LIMIT = config->c0LowerLimit;
        base->CTR[0].unUPPER_LIMIT.stcField.u16UPPER_LIMIT = config->c0UpperLimit;
        base->CTR[0].unWARN_LIMIT.stcField.u16WARN_LIMIT   = config->c0WarnLimit;
        tempConfigParams.stcField.u2LOWER_ACTION           = config->c0LowerAction;
        tempConfigParams.stcField.u2LOWER_ACTION           = config->c0LowerAction;
        tempConfigParams.stcField.u2UPPER_ACTION           = config->c0UpperAction;
        tempConfigParams.stcField.u1WARN_ACTION            = config->c0WarnAction;
        tempConfigParams.stcField.u1AUTO_SERVICE           = config->c0AutoService;
        tempConfigParams.stcField.u1SLEEPDEEP_PAUSE        = config->c0SleepDeepPause;
        tempConfigParams.stcField.u1DEBUG_RUN              = config->c0DebugRun;
        base->CTR[0].unCONFIG.u32Register                  = tempConfigParams.u32Register;


       /*  Configure Subcounter 1.  */
        base->CTR[1].unLOWER_LIMIT.stcField.u16LOWER_LIMIT = config->c1LowerLimit;
        base->CTR[1].unUPPER_LIMIT.stcField.u16UPPER_LIMIT = config->c1UpperLimit;
        base->CTR[1].unWARN_LIMIT.stcField.u16WARN_LIMIT   = config->c1WarnLimit;
        tempConfigParams.stcField.u2LOWER_ACTION           = config->c1LowerAction;
        tempConfigParams.stcField.u2UPPER_ACTION           = config->c1UpperAction;
        tempConfigParams.stcField.u1WARN_ACTION            = config->c1WarnAction;
        tempConfigParams.stcField.u1AUTO_SERVICE           = config->c1AutoService;
        tempConfigParams.stcField.u1SLEEPDEEP_PAUSE        = config->c1SleepDeepPause;
        tempConfigParams.stcField.u1DEBUG_RUN              = config->c1DebugRun;
        base->CTR[1].unCONFIG.u32Register                  = tempConfigParams.u32Register;


       /*  Configure Subcounter 2.  */
        tempCNT2ConfigParams.stcField.u5BITS            = config->c2ToggleBit;
        tempCNT2ConfigParams.stcField.u1ACTION          = config->c2Action;
        tempCNT2ConfigParams.stcField.u1SLEEPDEEP_PAUSE = config->c2SleepDeepPause;
        tempCNT2ConfigParams.stcField.u1DEBUG_RUN       = config->c2DebugRun;
        base->unCTR2_CONFIG.u32Register                 = tempCNT2ConfigParams.u32Register;

        Cy_MCWDT_Lock(base);       /*  Lock MCWDT Registers */
```

```
        ret = CY_MCWDT_SUCCESS;
    }

    return (ret);
}
```

Code Listing 21 ドライバ部の MCWDT レジスタアンロックプログラム例

```
#define CY_MCWDT_LOCK_CLR0      (1ul)
#define CY_MCWDT_LOCK_CLR1      (2ul)
__STATIC_INLINE void Cy_MCWDT_Unlock(volatile stc_MCWDT_t *base)
{
    uint32_t interruptState;

    interruptState = Cy_SysLib_EnterCriticalSection();

   /*  (10) Unlock MCWDT Registers when Interrupts are disabled. */
    base->unLOCK.stcField.u2MCWDT_LOCK = CY_MCWDT_LOCK_CLR0;
    base->unLOCK.stcField.u2MCWDT_LOCK = CY_MCWDT_LOCK_CLR1;

    Cy_SysLib_ExitCriticalSection(interruptState);
}
```

Code Listing 22 ドライバ部の MCWDT 割込みイネーブルプログラム例

```
__STATIC_INLINE void Cy_MCWDT_SetInterruptMask(volatile stc_MCWDT_t *base, uint32_t counters)
{
    if (counters & CY_MCWDT_CTR0)
    {
        base->unINTR_MASK.stcField.u1CTR0_INT = 1ul;    /* (11) Enable the MCWDT Subcounter
Interrupt.  */
    }
    if (counters & CY_MCWDT_CTR1)
    {
        base->unINTR_MASK.stcField.u1CTR1_INT = 1ul;
    }
    if (counters & CY_MCWDT_CTR2)
    {
        base->unINTR_MASK.stcField.u1CTR2_INT = 1ul;
    }
}
```

Code Listing 23 ドライバ部の MCWDT カウンタイネーブルプログラム例

```
__STATIC_INLINE void Cy_MCWDT_Enable(volatile stc_MCWDT_t *base, uint32_t counters, uint16_t
waitUs)
{
    if (counters & CY_MCWDT_CTR0)
    {
        base->CTR[0].unCTL.stcField.u1ENABLE = 1ul;   /*  (12) Enable the MCWDT Subcounter.  */
    }
    if (counters & CY_MCWDT_CTR1)
    {
        base->CTR[1].unCTL.stcField.u1ENABLE = 1ul;
    }
    if (counters & CY_MCWDT_CTR2)
    {
        base->unCTR2_CTL.stcField.u1ENABLE = 1ul;
    }
    Cy_SysLib_DelayUs(waitUs);
}
```

Code Listing 24 ドライバ部の MCWDT レジスタロックプログラム例

```
#define CY_MCWDT_LOCK_SET01     (3ul)
__STATIC_INLINE void Cy_MCWDT_Lock(volatile stc_MCWDT_t *base)
{
    uint32_t interruptState;

    interruptState = Cy_SysLib_EnterCriticalSection();

    base->unLOCK.stcField.u2MCWDT_LOCK = CY_MCWDT_LOCK_SET01;   /* (13) Lock MCWDT Registers
when interrupts are disabled.  */

    Cy_SysLib_ExitCriticalSection(interruptState);
}
```

3.8 MCWDT のクリア

MCWDT のクリアは、Subcounter 0 の場合は CTR0_SERVICE[0]ビットを'1'に設定し、Subcounter 1 の場合は CTR1_SERVICE[1]ビットを'1'に設定します。両方のビットは SERVICE レジスタにあります。ファームウェアは、'1'を設定する前に、'0'になるまでに対応するビットを読み出す必要があります。

- MCWDT カウンタの処理は、安定したソフトウェアフローを確保するために定期的に行う必要があります。使用されるソフトウェアコンセプトとは無関係に、ソフトウェアコンポーネントのランタイム計算は、クリアされるカウンタの閾値を定義するために重要です。ウィンドウモードは、ソフトウェアが MCWDT 処理として予期していない最小期間を決定する必要があるため、さらに複雑になります。この最小期間は、例えば、優先度の低い主機能の最小実行時間とすることができます。これは、他のコードを実行することなく MCWDT サービスルーチンを連続的に実行するソフトウェアなどの異常状況を検出する際に役立ちます。
- この手順は、ウィンドウモードのベーシック WDT と同じです。異なるタスクを持つシステム内でウォッチドッグカウンタをクリアできる例を、図 7 に示します。各サービス モーメントの計算において、ウィンドウモードでは

FAULT およびリセットイベント回避のため、カウンタが LOWER_LIMIT に達する前にクリアが行われず、UPPER_LIMIT に達しないことを考慮する必要があります。

3.8.1 使用事例

ここでは、使用事例で説明した使用例によって MCWDT をクリアする例について説明します。

3.8.2 MCWDT クリアフロー例

図 14 に、MCWDT をクリアするフロー例を示します。

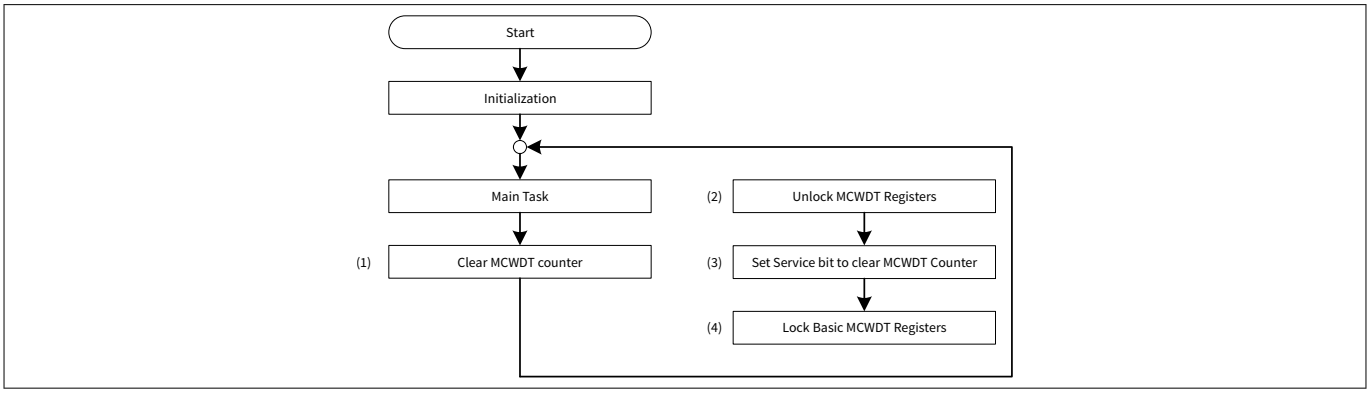

図 14 MCWDT クリアフロー例

3.8.3 MCWDT クリアプログラム例

Code Listing 25 に、MCWDT をクリアするプログラム例を示します。

Code Listing 25 MCWDT をクリアするプログラム例

```
int main(void)
{
:
for(;;)
{
       :
Cy_MCWDT_ClearWatchdog(MCWDT1, CY_MCWDT_COUNTER0);   /*  (1) Clear the MCWDT counter. */
}
}
```

Code Listing 26 に、ドライバ部の MCWDT カウンタをクリアするプログラム例を示します。

Code Listing 26 ドライバ部の MCWDT カウンタクリアプログラム例

```
void Cy_MCWDT_ClearWatchdog(volatile stc_MCWDT_t *base, cy_en_mcwdtctr_t counter)
{
    Cy_MCWDT_Unlock(base);   /*  (2) Unlock MCWDT Registers . */
    Cy_MCWDT_ResetCounters(base, (1u << (uint8_t)counter), 0u);
    Cy_MCWDT_Lock(base);   /*  (4) Lock MCWDT Registers . */
}

__STATIC_INLINE void Cy_MCWDT_ResetCounters(volatile stc_MCWDT_t *base, uint32_t counters,
uint16_t waitUs)
{
    if (counters & CY_MCWDT_CTR0)
    {
        base->unSERVICE.stcField.u1CTR0_SERVICE = 1ul;   /*  (3) Set the Service bit to clear
the MCWDT counter. */
    }
    if (counters & CY_MCWDT_CTR1)
    {
        base->unSERVICE.stcField.u1CTR1_SERVICE = 1ul;
    }
    if (counters & CY_MCWDT_CTR2)
    {
        // No reset functionality for CTR2
    }
    Cy_SysLib_DelayUs(waitUs);
}
```

3.9 MCWDT のフォールト処理

4 つのフォールトは 1 つのフォールト報告に集約されます。この報告にはフォールトがトリガされたデータが含まれ、フォールトハンドラは正しいフォールト原因を記録できます。異なる MCWDT インスタンスには独立したフォールト報告があるため、異なるプロセッサで処理できます。

フォールト報告の初期化を図 13 と Code Listing 18 に示します。例として、フォールトストラクチャ 1 が使用されています。

フォールト設定手順の詳細については、関連ドキュメントに記載されている AN219842 の“Fault Report Structure”章を参照してください。

フォールトは、FAULT report ハンドラ内で処理されます。MCWDT には、次の 4 つの FAULT ソースがあります。

- 下限 Fault Subcounter 0
- 上限 Fault Subcounter 0
- 下限 Fault Subcounter 1
- 上限 Fault Subcounter 1

Fault ステータスは、関連する Fault ストラクチャから読み出せます。

3.9.1 使用事例

ここでは、使用事例で説明した使用例によって MCWDT フォールト処理の例について説明します。

3.9.2 MCWDT フォールト処理のフロー例

図 15 に、MCWDT フォールト処理のフロー例を示します。

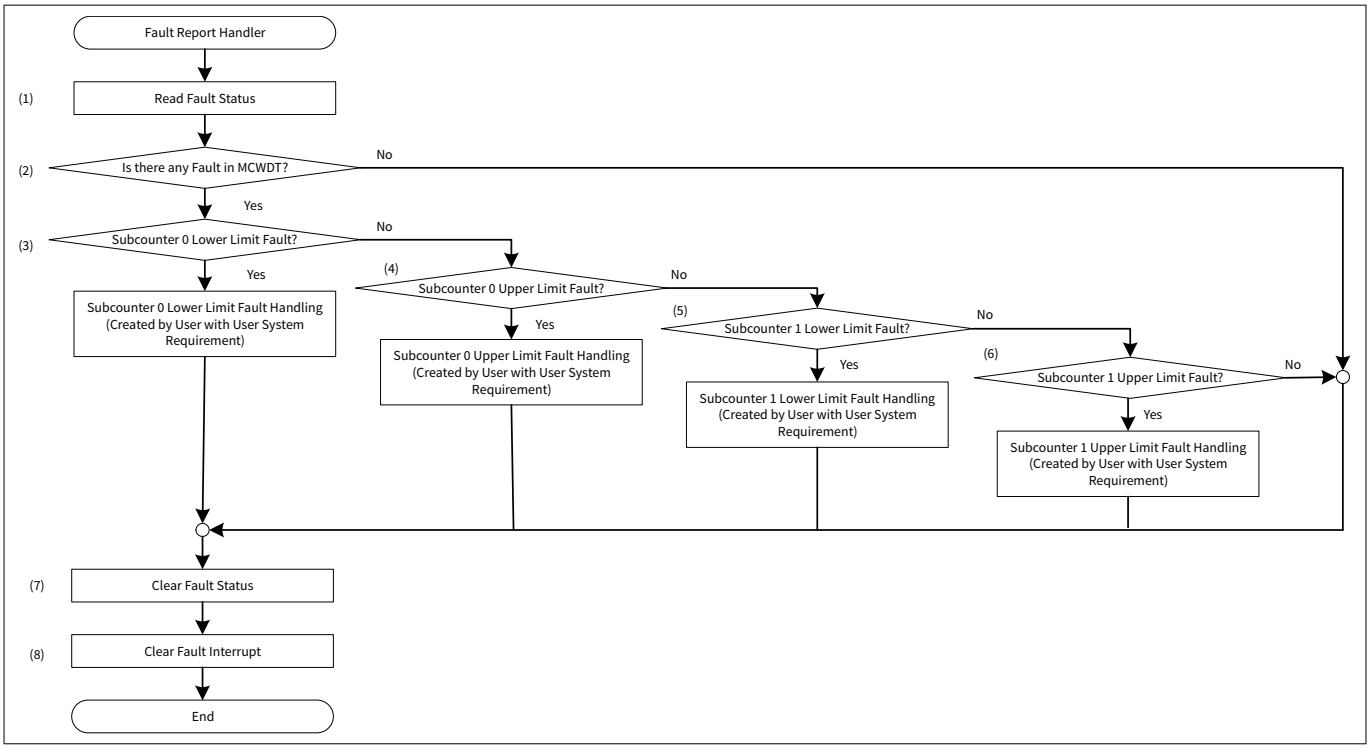

図 15 MCWDT フォールト処理のフロー例

3.9.3 MCWDT フォールト処理のプログラム例

Code Listing 27 に、MCWDT フォールト処理のプログラム例を示します。

Code Listing 27 MCWDT フォールト処理のプログラム例

```
void irqFaultReport1Handler(void)
 {
    cy_en_sysflt_source_t status;
    uint32_t faultData;

    /* Read FAULT status from FAULT structure */
    status = Cy_SysFlt_GetErrorSource(FAULT_STRUCT1);   /*  (1) Read Fault Status Register
(FAULT_STRUCT1_STATUS) */

    /* Evaluation of FAULT status */
    if(status != CY_SYSFLT_NO_FAULT)
    {
        */ MCWDT1 FAULT */
        if(status == CY_SYSFLT_SRSS_MCWDT1)   /* (2) Check if any Fault in MCWDT1
(FAULT_STRUCT1_STATUS.SRSS_MCWDT1) */
        {
            /* Read and evaluate FAULT source from FAULT structure */
            faultData = Cy_SysFlt_GetData0(FAULT_STRUCT1);   /* Check Fault Data Register
(FAULT_STRUCT1_DATA0.[0-3]) */
            if(faultData & 0x00000001ul)   /* (3) Check if Subcounter 0 Lower Limit Fault  */
            {
                // Subcounter 0 lower limit fault handling created by user
            }
            else if(faultData & 0x00000002ul)   /* (4) Check if Subcounter 0 Upper Limit Fault
*/
            {
                // Subcounter 0 upper limit fault handling created by user
            }
            else if(faultData & 0x00000004ul)   /* (5) Check if Subcounter 1 Lower Limit Fault
*/
            {
                // Subcounter 1 lower limit fault handling created by user
            }
            else if(faultData & 0x00000008ul)   /* (6) Check if Subcounter 1 Upper Limit Fault
*/
            {
                // Subcounter 1 upper limit fault handling created by user
            }
        }
    }
    /* Clear FAULT interrupt */
    Cy_SysFlt_ClearStatus(FAULT_STRUCT1);   /* (7) Clear Fault Status (FAULT_STRUCT1_STATUS =
0) */
    Cy_SysFlt_ClearInterrupt(FAULT_STRUCT1);   /* (8) Clear Fault Interrupt (FAULT_INTR.FAULT =
1) */
 }
```

3.10 MCWDT のリセット要因表示

MCWDT カウンタが処理されていない、または早すぎる場合、FAULT 処理の時間内での未完によってシステムリセットが発行されます。デバイスがリセット解除後にリセットの原因を知ることは有益です。リセット要因は RES_CAUSE レジスタに記録されます。使用された MCWDT インスタンスに応じて、リセットイベントは RES_CAUSE レジスタの RESET_MCWDT0[5], RESET_MCWDT1[6], RESET_MCWDT2[7], および RESET_MCWDT3[8]ビットに格納されます。RES_CAUSE レジスタのビットは、対応するリセットの発生時に設定され、ユーザソフトウェアまたはパワーオンリセット (POR) によってクリアされるまでセットされます。

3.11 MCWDT レジスタ

表 6 MCWDT レジスタ

名称	説明
MCWDTx_CTL	MCWDT Subcounter 0/1 制御レジスタ
MCWDTx_LOWER_LIMIT	MCWDT Subcounter 0/1 下限レジスタ
MCWDTx_UPPER_LIMIT	MCWDT Subcounter 0/1 上限レジスタ
MCWDTx_WARN_LIMIT	MCWDT Subcounter 0/1 警告レジスタ
MCWDTx_CONFIG	MCWDT Subcounter 0/1 コンフィギュレーションレジスタ
MCWDTx_CNT	MCWDT Subcounter 0/1 カウントレジスタ
MCWDT2_CTR2_CTL	MCWDT Subcounter 2 制御レジスタ
MCWDT2_CTR2_CONFIG	MCWDT Subcounter 2 コンフィギュレーションレジスタ
MCWDT2_CTR2_CNT	MCWDT Subcounter 2 カウントレジスタ
MCWDT2_LOCK	MCWDT ロックレジスタ
MCWDT2_SERVICE	MCWDT サービスレジスタ
MCWDT2_INTR	MCWDT 割込みレジスタ
MCWDT2_SET	MCWDT 割込みセット レジスタ
MCWDT2_MASK	MCWDT 割込みマスクレジスタ
MCWDT2_MASKED	MCWDT 割込みマスクレジスタ
CLK_SELECT	クロック選択レジスタ
CLK_ILO_CONFIG	ILO の構成
RES_CAUSE	リセット要因レジスタ

4 デバッグサポート

両タイプの WDT は異なるデバッグモードをサポートします。設定は DEBUG_TRIGGER_ENABLE[28] ビットと DEBUG_RUN[31] ビットで行われ、それぞれベーシック WDT と MCWDT の関連する CONFIG レジスタに配置されています。WDT リセット要求はデバッグモード中、ブロックされ、デバッグモードのブレークポイントを使用することで MCWDT リセットのデバッグが可能です。

表 7 デバッグ モード

DEBUG_RUN	DEBUG_TRIGGER_ENABLE	説明
0	0	デバッガが接続されているときカウンタが停止します。
0	1	デバッガが接続されており、ブレークポイント中に CPU が停止している場合にのみカウンタが停止します。
1	x	デバッガが接続されているときカウンタは動作します。ブレークポイント中に CPU が停止し、カウンタが停止していない場合でもリセットは発行されません。

いずれの場合も、デバッガがターゲット システムに接続されているときは、リセットまたは FAULT は発行されません。

デバッグ中にブレークポイントで一時停止するには、関連する「CPU 停止」信号を関連 WDT のトリガ入力に接続するようにトリガマトリックスを設定します。トリガ信号を処理するには、最大 2 回の LFCLK サイクルが必要です。2 回の LFCLK サイクル未満のトリガは見逃される場合があります。同期エラーは停止するたびに累積する可能性があります。

5 定義, 頭字語, および略語

表 8 定義, 頭字語, および略語

用語	定義
AHB	Advanced High-performance Bus (アドバンスドハイパフォーマンスバス)
CPU	中央演算処理装置
CPUSS	CPU subsystem (CPU サブシステム)
ECO	High-frequency crystal oscillator (高周波水晶発振器)
ILO0	32-kHz internal low-speed oscillator (32kHz 内部低速発振器)
IRQ	Interrupt request (割込み要求)
ISR	Interrupt Service Routine (割込みサービスルーチン)
kHz	kilohertz (キロヘルツ)
LFCLK	Low-frequency clock (低周波クロック)
MCWDT	マルチカウンター ウォッチドッグタイマ (Multi-counter watchdog timer の略)。
ms, msec	milliseconds (ミリ秒)
POR	Power-on reset (パワーオン リセット)
PPU	Peripheral protection unit (周辺機能保護ユニット)
s	second (秒)
SW	Software (ソフトウェア)
V_{DDD}	External high-voltage supply (外部高電圧電源)
WCO	Low-frequency watch crystal oscillator (低周波時計用水晶発振器)
WDT	ウォッチドッグタイマー
WIC	Wakeup interrupt controller (ウェイクアップ割込みコントローラ)

6 関連ドキュメント

以下は TRAVEO™ T2G ファミリのデータシートおよびテクニカルリファレンスマニュアルです。インフィニオン サポートに連絡して、これらのドキュメントを入手してください。

- 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
- ボディコントローラ 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 entry registers technical reference manual (TRM) for CYT2BL (Doc No. 002-29852)
- ボディコントローラ 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)
- 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)
- 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
- アプリケーションノート
  - AN219842 – TRAVEO™ T2G の割込みの使用方法

7 その他の関連資料

インフィニオンは、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、初期化コードを含むサンプル ドライバ ライブラリ (SDL) を提供しています。また、SDL は、AUTOSAR™ の公式製品でカバーされてい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、製造目的で使用することはできません。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 を入手するには、インフィニオン サポートに連絡してください。

改訂履歴

版数	日付	変更内容
英語版**	2018-08-21	本版は英語版のみの発行です。Initial release.
**	2019-01-16	これは英語版 002-19944 Rev. *A を翻訳した日本語版 002-25806 Rev. ** です。
*A	2019-05-23	これは英語版 002-19944 Rev. *B を翻訳した日本語版 002-25806 Rev. *A です。
*B	2019-12-17	これは英語版 002-19944 Rev. *C を翻訳した日本語版 002-25806 Rev. *B です。
英語版*D	2020-03-02	本版は英語版のみの発行です。英語版の改訂内容: Changed target part numbers from “CYT2B/CYT4B/CYT4D Series” to “CYT2/CYT4 Series” in all instances across the document. Added target part number (CYT3 series) in all instances across the document.
*C	2020-09-28	これは英語版 002-19944 Rev. *E を翻訳した日本語版 002-25806 Rev. *C です。主な変更点は以降のとおりです。セクション 2.7、2.8、3.7、3.8、3.9 にフローを追加。 セクション 2.7、2.8、3.7、3.8、3.9 のサンプルコードを更新。 セクション 6 に AN219842 を追加。 セクション 7（サンプルドライバライブラリーの情報を含む）を追加。
*D	2021-05-17	これは英語版 002-19944 Rev. *F を翻訳した日本語版 002-25806 Rev. *D です。主な変更点は以降のとおりです。 テンプレートの変更を実施。
*E	2024-04-02	これは英語版 002-19944 Rev. *G を翻訳した日本語版 002-25806 Rev. *E です。
*F	2025-06-02	これは英語版 002-19944 Rev. *H を翻訳した日本語版 002-25806 Rev. *F です。英語版の改訂内容: Added to CYT6BJ

Trademarks

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

Edition 2025-06-02

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-ova1681539638600

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。