TRAVEO™ T2G ファミリ低消費電力モードの使用方法

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 の低消費電力モードの特長および低消費電力モードへの入り方、Active モードへの復帰方法について説明します。

対象者

本書は、低消費電力モードを TRAVEO™ T2G に使用するお客様を対象とします。

関連製品ファミリ

TRAVEO™ T2G ファミリ CYT2/CYT3/CYT4/CYT6 シリーズ

目次

本書について . . . 1
目次 . . . 2
1 はじめに . . . 3
2 TRAVEO™ T2G ファミリの消費電力モード . . . 4
3 消費電力モードの遷移 . . . 6
3.1 各消費電力モードへの遷移方法 . . . 6
3.1.1 RESET/OFF 状態 . . . 7
3.1.2 低消費電力モードへの遷移 . . . 7
3.1.3 低消費電力モードからの復帰 . . . 13
3.2 低消費電力モード中の WDT の設定について . . . 15
3.2.1 特長 . . . 16
3.2.2 WDT を使ったウェイクアップ動作の使用例 . . . 16
3.2.2.1 設定とサンプルコード . . . 18
3.3 Cyclic ウェイクアップ動作 . . . 28
3.3.1 Cyclic ウェイクアップの使用例 . . . 28
3.3.2 Cyclic ウェイクアップ動作 . . . 29
3.3.3 Cyclic ウェイクアップ動作のフローチャート . . . 31
3.3.3.1 設定とサンプルコード . . . 31
3.3.4 Cyclic ウェイクアップでのスマート I/O の使用 . . . 41
3.3.4.1 Cyclic ウェイクアップにおけるスマート I/O 実装の利点 . . . 42
3.3.4.2 Cyclic ウェイクアップでのスマート I/O 設定 . . . 43
3.3.4.3 Cyclic ウェイクアップ動作でのセンサ起動回路 . . . 47
3.3.4.4 設定とサンプルコード . . . 49
3.4 CAN ウェイクアップ動作 . . . 59
3.4.1 設定とサンプルコード . . . 60
用語集 . . . 65
関連ドキュメント . . . 67
その他の参考資料 . . . 68
改訂履歴 . . . 69
免責事項 . . . 70

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 の低消費電力モードについて説明します。このシリーズは、Arm® Cortex® CPU コアと CAN FD, メモリ, アナログおよびデジタル周辺機能を 1 つのチップに搭載しています。

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+) を搭載します。CYT4 シリーズには 2 つの Arm® Cortex®-M7 ベースの CPU (CM7) と CM0+があり、CYT3 シリーズには 1 つの CM7 と CM0+があります。CYT6 シリーズには、4 つの Arm® Cortex®-M7 ベースの CPU (CM7) と CM0+があります。

TRAVEO™ T2G ファミリ MCU には、様々な消費電力モードがあります。これらのモードにより、アプリケーションの平均消費電力を最小限にすることを目的とします。

このアプリケーションノートでは、消費電力モードの特長および設定方法について説明します。

本書で記載されている機能や用語については、Architecture reference manual の Device Power Modes 章を参照してください。

2 TRAVEO™ T2G ファミリの消費電力モード

TRAVEO™ T2G ファミリ MCU には、以下の消費電力モードがあります。

- **Active モード:** すべての周辺機能が有効です。
- **Sleep モード:** CPU を除くすべての周辺機能が有効です。
- **DeepSleep モード:** 低い周波数で動作する周辺機能のみ有効です。
- **Hibernate モード:** デバイスおよび I/O の機能は停止した状態です。

図 1 に消費電力モードと消費電流の関係を示します。

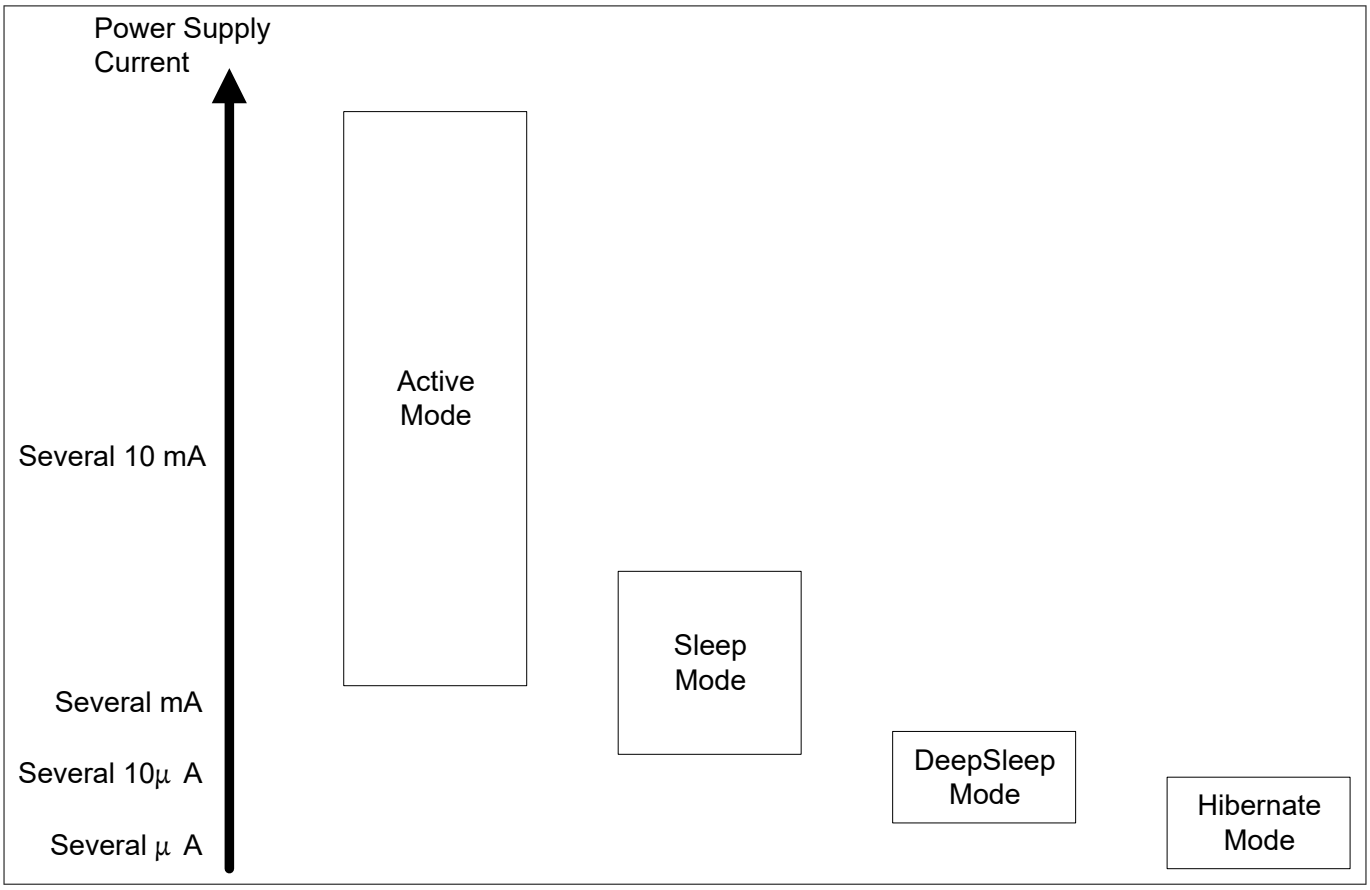

図 1 消費電力モードおよび消費電流

注: 図 1 はあくまでも各モードでの消費電流の程度を示します。実際の電流値は、使用するクロックや周辺機能の設定にも依存します。消費電流の電気的特性についてはデータシートを参照してください。

消費電力は、Active モード, Sleep モード, DeepSleep モード, Hibernate モードの順に小さくなります。各消費電力モードの切り替えにより、お客様のアプリケーションの消費電力を最適化します。

表 1 にそれぞれの消費電力モードの説明および遷移条件, 復帰条件をまとめます。消費電力モードの詳細は、Architecture reference manual を参照してください。

2 TRAVEO™ T2G ファミリの消費電力モード

表 1 TRAVEO™ T2G 消費電力モード

消費電力モード	説明	遷移条件	復帰要因	復帰動作
Active	主要な動作モード。すべての周辺機能を利用可能 (プログラム可能)	Sleep/DeepSleep モード, Hibernate リセット, またはその他のリセットからの復帰	N/A	N/A
Sleep	CPU は Sleep モード; 他のすべての周辺機能は利用可能			割込み
DeepSleep	すべての高周波クロックと周辺機能はオフになっています。低周波クロック (32kHz) と低消費電力アナログおよびデジタル周辺機能は、動作用およびウェイクアップソースとして利用できます。SRAM 値は保持可能 (設定変更可能)。	Active モード時のレジスタ書込み, もしくはデバッガセッションの終了	GPIO 割込み, イベントジェネレータ, SCB, ウォッチドッグタイマ, および RTC アラーム [1] およびデバッガ	割込みまたはデバッグ
Hibernate	GPIO の状態は凍結。ほぼすべての周辺機能とクロックも停止。復帰時にデバイスがリセットされます。	Active モードでレジスタ書込みを実施	WAKEUP ピンおよび RTC アラーム	Hibernate リセット

1) RTC (原振: WCO) は VDDD 電源で動作し、デバイスの消費電力モードに関わらず有効です。RTC アラームを使用して、各消費電力モードから復帰可能です。

TRAVEO™ T2G ファミリ MCU には以下の特長があります。

- ソフトウェアで消費電力モードを切り換えることにより、アプリケーションの消費電力を最適にすることが可能です。
- 低消費電力である DeepSleep モードからの復帰は複数の復帰要因をサポートしており、DeepSleep モード中に SRAM 値を保持するかは設定可能です。
- 超低消費電力である Hibernate モードからは、I/O または RTC アラームによって復帰可能です。

さまざまな消費電力モードでの電力消費は、次の方法で制御されます。

- 各周辺機能のクロックを有効もしくは無効に設定
- クロックの電源を ON もしくは OFF に設定
- 各周辺機能の電源を ON もしくは OFF に設定

3 消費電力モードの遷移

ここでは、サンプルドライバライブラリ (SDL) を使用して低消費電力モードの手順の使用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 には設定部とドライバ部があります。設定部では、主に目的の動作をさせるためのパラメーター値を設定します。ドライバ部は設定部のパラメーター値に基づいて各レジスタを設定します。システムに応じて設定部を設定できます。

この例では、CYT2B7 シリーズを使用しています。

3.1 各消費電力モードへの遷移方法

図 2 にデバイスの状態遷移図を示します。各消費電力モードへの遷移方法については後述します。

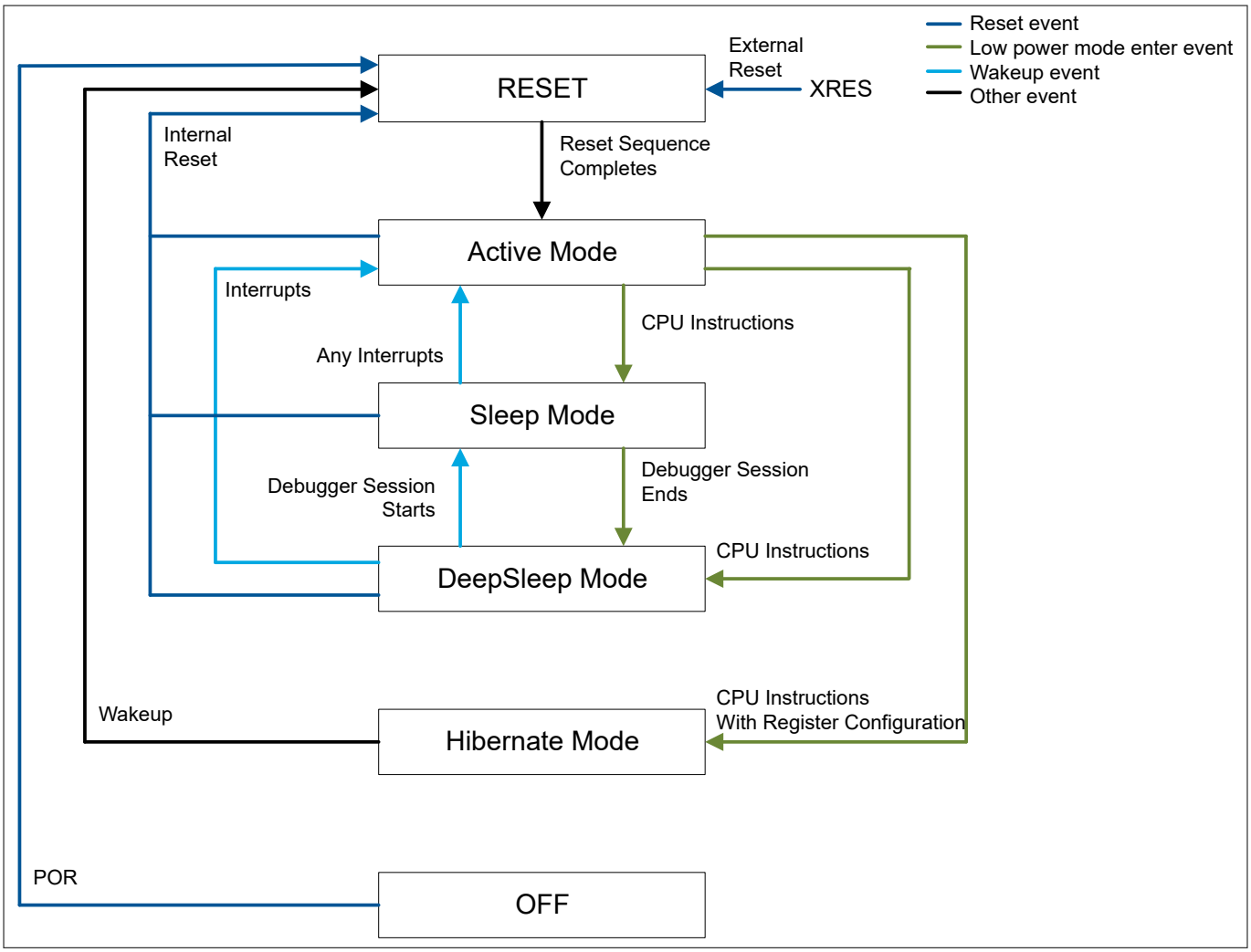

図 2 消費電力モードの状態遷移図

3.1.1 RESET/OFF 状態

- OFF 状態
  - 電源が供給されていない状態
  - 電源 ON 時、パワーオンリセットが発生し (POR イベント)、RESET 状態へ遷移
- RESET 状態
  - POR や外部リセット(XRES), 内部リセットなどリセットイベントを検出
  - リセットシーケンス完了後、Active モードへ遷移
  - IMO の発振開始
  - デバイスはどの消費電力モードであっても、XRES リセットのアサートにより、RESET 状態へ移行

3.1.2 低消費電力モードへの遷移

表 2 に低消費電力モードへの遷移方法および低消費電力モードでの動作を示します。

表 2 低消費電力モードへの遷移

遷移前の状態	遷移後の状態	トリガ	ハードウェアの動作
Active	Sleep	**ファームウェアの動作** **1.** すべての CPU (CYT2 の場合、CPU は CM4 および CM0+です。CYT3/CYT4/CYT6 の場合、CPU は CM7 および CM0+です) の SCR レジスタの SLEEPDEEP ビット[2] をクリアしてください。 **2.** CPU が割込み時にのみ動作する場合は、オプションで SCR レジスタの SLEEPONEXIT ビット[1] を設定してください。このビットが設定されると、WFI/WFE 命令が実行された後に CPU はアプリケーションコードに戻りません。CPU は有効になっている割込みまたはイベントでウェイクアップし、割込みが終了するかイベントを処理するとすぐに Sleep/DeepSleep モードに入ります。 **3.** アプリケーションが保留中の割込みから CPU をウェイクアップする必要がある場合は、オプションで SCR レジスタの SEVONPEND ビット[4]を設定してください。このビットが設定されている場合、保留状態へ移行するどんな割込みも CPU をウェイクアップします。 **4.** すべての CPU で WFI/WFE 命令を実行してください。(CYT2 の場合、CPU は CM4 および CM0+です。CYT3/CYT4/CYT6 の場合、CPU は CM7 および CM0+です。)	**1.** CPU へのクロック供給停止 **2.** CPU は割込み待ちまたは、ウェイクアップイベント待ち状態

(続く)

表 2 (続き) 低消費電力モードへの遷移

遷移前の状態	遷移後の状態	トリガ	ハードウェアの動作
Active	Deep Sleep	**ファームウェアの動作** DeepSleep モードに入るには次の手順を実行してください (これら手順の前に、PWR_CTL レジスタの LPM_READY ビット[5] が'1'であることを確認してください。) **1.** すべての CPU (CYT2 の場合、CPU は CM4 および CM0+です。CYT3/CYT4/CYT6 の場合、CPU は CM7 および CM0+です) に対し、SCR レジスタの SLEEPDEEP ビット[2]を設定してください。 **2.** CPU が割込み時にのみ動作する場合は、オプションで SCR レジスタの SLEEPONEXIT ビット[1] を設定してください。このビットが設定されると、WFI/WFE 命令が実行された後に CPU はアプリケーションコードに戻りません。CPU は有効になっている割込みまたはイベントでウェイクアップし、割込みが終了するかイベントを処理するとすぐに Sleep/DeepSleep モードに入ります。 **3.** アプリケーションが保留中の割込みから CPU をウェイクアップする必要がある場合は、オプションで SCR レジスタの SEVONPEND ビット[4]を設定してください。このビットが設定されている場合、保留状態へ移行するどんな割込みも CPU をウェイクアップします。 **4.** すべての CPU (CYT2 の場合、CPU は CM4 および CM0+です。CYT3/CYT4/CYT6 の場合、CPU は CM7 および CM0+です) で WFI/WFE 命令を実行してください。 ***注**: 低消費電力モード (LPM_READY == 1) の準備ができる前に上記のシーケンスを実行すると、まず Sleep モードに移行します。LPM_READY ビットが設定されると、デバイスの状態は自動的に DeepSleep 状態に移行します。*	**1.** CPU は低消費電力モードに遷移 **2.** 高速クロックを停止 **3.** I/O セルは自動で停止状態 **4.** 保持回路は有効になり非保持ロジック回路はリセット状態 **5.** Active モード用レギュレータが無効となり、DeepSleep モード用レギュレータに切替え

(続く)

表 2 (続き) 低消費電力モードへの遷移

遷移前の状態	遷移後の状態	トリガ	ハードウェアの動作
		注: *WFI 命令を実行する前に、行われた書込み転送が反映されていることを同じメモリ位置へのリードアクセスによって確認してください。これにより、書込み操作は確実に成功します。*	

(続く)

表 2 (続き) 低消費電力モードへの遷移

遷移前の状態	遷移後の状態	トリガ	ハードウェアの動作
Active	Hibernate	**ファームウェアの動作** **1.** PWR_HIBERNATE レジスタ (オプション) と PWR_HIB_DATA レジスタの TOKEN ビット[7:0]を、Hibernate モードからのウェイクアップイベントで使用できるアプリケーション固有の分岐データに設定してください。 **2.** PWR_HIBERNATE レジスタの FREEZE ビットおよび HIBERNATE ビットを制御するために、PWR_HIBERNATE レジスタの UNLOCK ビット[8:15]を 0x3A に設定してください。 **3.** アプリケーション要件に基づいて、PWR_HIBERNATE レジスタでウェイクアップピンの極性 (POLARITY_HIBPIN ビット[23:20]), ウェイクアップピンのマスク (MASK_HIBPIN ビット[27:24]), ウェイクアップアラームマスク (MASK_HIBALARM ビット[18]) を設定してください。 **4.** I/O ピンをフリーズするために、PWR_HIBERNATE レジスタの FREEZE ビット[17]を設定してください。 **5.** Hibernate モードに入るために、PWR_HIBERNATE レジスタの HIBERNATE ビット[31]を設定してください。 **6.** 書込みを有効にするために、PWR_HIBERNATE レジスタを読んでください。 **7.** すべての CPU で WFI 命令を実行してください。 ***注**: Hibernate モードをアトミックに起動することを推奨します。つまり Hibernate モードに入るときは、すべての割込みを無効にして PWR_HIBERNATE レジスタに書込み操作を行ってください。* ***注**: WFI 命令を実行する前に、行われた書込み転送が反映されていることを同じメモリ位置へのリードアクセスによって確認してください。これにより、書込み操作は確実に成功します。*	**1.** CPU は低消費電力モードに遷移 **2.** 高速および低速クロックを停止 **3.** 保持回路は有効になり非保持ロジック回路はリセット状態 **4.** Active モード用および DeepSleep モード用レギュレータがパワーダウン。Hibernate ドメインで動作する周辺機能は直接 VDDD 電源で動作 **5.** I/O セルが動作停止

(続く)

3 消費電力モードの遷移

表 2　　(続き) 低消費電力モードへの遷移

遷移前の状態	遷移後の状態	トリガ	ハードウェアの動作
Sleep	DeepSleep	デバッガ未接続の状態で DeepSleep モードへ移行する際、LPM_READY==0 の場合、デバイスはいったん Sleep モードに移行します。LPM_READY==1 のとき、デバイスは自動的に DeepSleep に移行します。 もしデバッガ接続の状態で DeepSleep モードへ移行した場合、デバイスは以下の条件が成立すると DeepSleep モードに遷移します。 **1.** LPM_READY==1 **2.** デバッガの切断	**1.** 高速クロックを停止 **2.** I/O セルを自動で停止 **3.** 保持回路は有効になり非保持ロジック回路はリセット状態 **4.** Active モード用のレギュレータは無効となり DeepSleep モード用レギュレータに切替え

図 3 に Active モードから DeepSleep モードへ遷移する際のソフトウェアおよびハードウェアの動作を示します。

3 消費電力モードの遷移

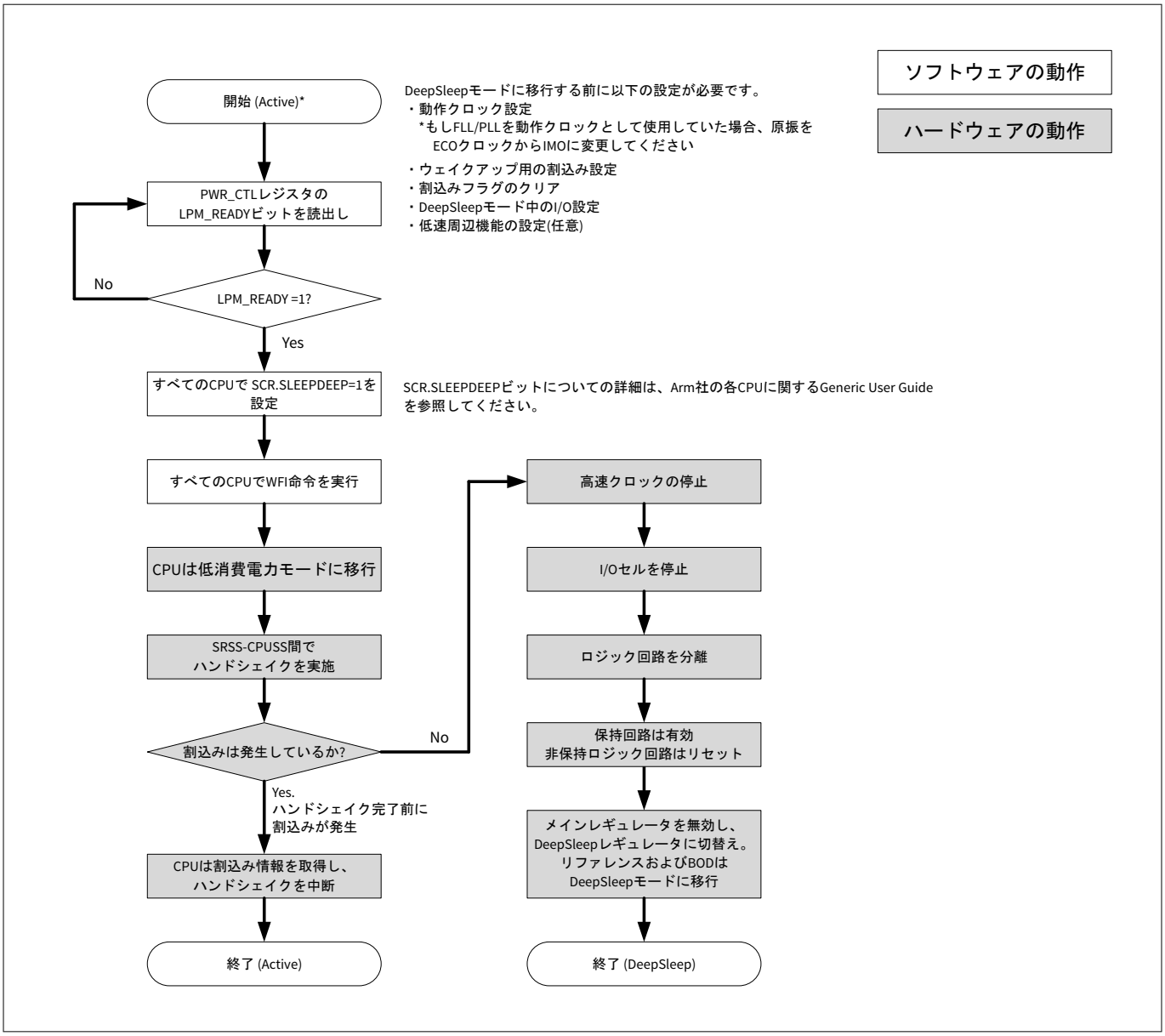

図 3　　Active モードから DeepSleep モードへの遷移

注:　　図 3 の灰色のボックスはハードウェアの動作を示します。したがって、ソフトウェアによる処理は必要ありません。

図 4 に Active モードから Hibernate モードへ遷移する際のソフトウェアおよびハードウェアの動作を示します。

3 消費電力モードの遷移

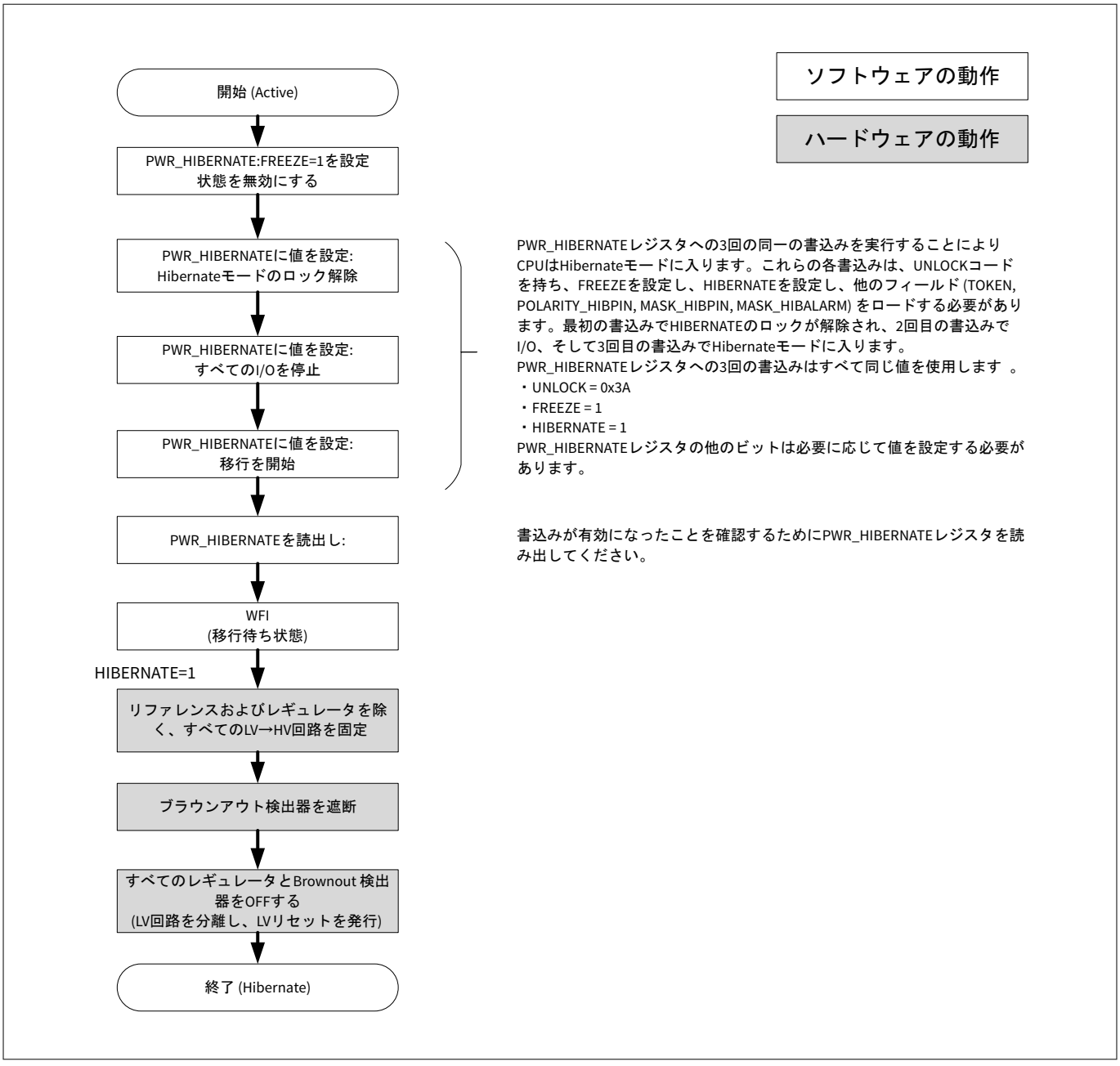

図 4　　Active モードから Hibernate モードへの遷移

注:　　図 4 中の灰色のボックスはハードウェアの動作を示します。したがって、ソフトウェアによる処理は必要ありません。

3.1.3　　低消費電力モードからの復帰

表 3 にウェイクアップ動作の起動要因およびウェイクアップ後の動作を示します。

3 消費電力モードの遷移

表 3　　ウェイクアップ動作

遷移前の状態	遷移後の状態	ウェイクアップ要因	ハードウェアの動作
Sleep	Active	Sleep モード中に有効となっている割込み	CPU は Sleep モードから復帰し、割込み処理を実行します。
DeepSleep	Active	DeepSleep モード中に有効となっている割込み	デバイスは、DeepSleep モードに遷移前の設定状態に復帰します。 (IMO/クロックは有効となり、保持機能は解除、非保持回路はリセット、停止状態は解除され、CPU は低消費電力モードから復帰し割込み情報を取得)
DeepSleep	Sleep	デバッグセッションの開始	保持機能は解除され、非保持回路はリセットされます。 停止状態は解除されます。 高速および低速クロックが有効になります。 CPU は Sleep 状態を継続します。
Hibernate	Active	ウェイクアップピン、RTC アラーム	Hibernate モードからのウェイクアップはリセット復帰となり Active モードへ移行します **1.** 低電圧レギュレータ (Active および DeepSleep モード) およびリファレンス回路が立ち上がる **2.** すべての低電圧ロジック回路 (内部レギュレータで動作する回路) がリセットされる **3.** IMO クロックが開始 **4.** コアの実行が開始

図 5 に DeepSleep モードから Active モードへ遷移する際のソフトウェアおよびハードウェアの動作を示します。

3 消費電力モードの遷移

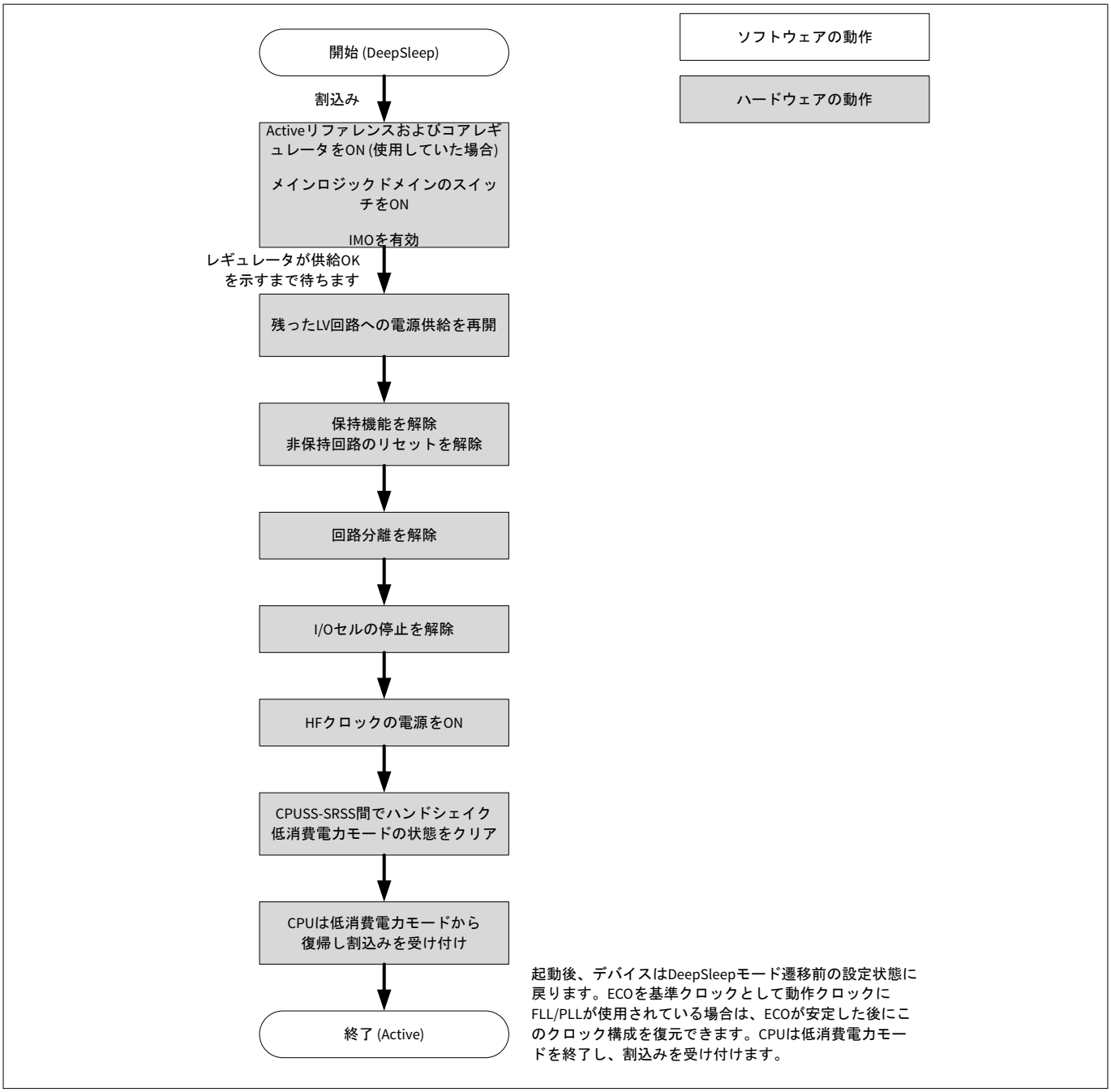

図 5　　DeepSleep モードから Active モードへの遷移

注:　　図 5 中の灰色のボックスはハードウェアの動作を示します。したがって、ソフトウェアによる処理は必要ありません。

3.2　　低消費電力モード中の WDT の設定について

TRAVEO™ T2G のウォッチドッグタイマ (WDT) は、ソフトウェアが意図しない処理を実行した場合、自動でデバイスをリセットします。加えて、WDT は割込み要因もしくは低消費電力モードからの復帰要因として使用可能です。ソフトウェアにて、リセットまたは割込みのどちらかを選択可能です。

ここでは、低消費電力モード中の WDT 設定と動作を説明します。WDT の詳細については、Architecture reference manual を参照してください。

3.2.1 特長

TRAVEO™ T2G は、Basic WDT と Multi-counter WDT (MCWDT)の 2 種類の WDT に対応します。表 4 に、各消費電力モードで設定可能な WDT の機能を示します。

表 4 PCLK (TCPWM タイマの例) 設定パラメーター覧

消費電力モード	Basic WDT	MCWDT			備考
		Subcounter0	Subcounter1	Subcounter2	
Active	リセット [1] および割込み [2]	リセット [1], 割込み [2], および FAULT[3]		割込み [4]	Active モードでは、WDT は CPU に対して割込みを通知可能。
Sleep	リセット [1] および割込み [2]	リセット [1], 割込み [2], および FAULT[3]		割込み [4]	Sleep モードでは、CPU サブシステムの電源はオフのため、WDT からの割込み要求はウェイクアップ割込みコントローラ (WIC) に直接通知される。それにより CPU が Sleep モードから復帰する。
Deep Sleep	リセット [1] および割込み [2]	リセット [1], 割込み [2], および FAULT[3]		割込み [4]	DeepSleep モードでは、CPU サブシステムの電源はオフのため、WDT からの割込み要求は WIC に直接通知される。 DeepSleep モード中での WDT のカウンタの停止/動作は、選択可能。
Hibernate	リセット [1] および割込み [2]	未対応			• Hibernate モード中、WDT のカウンタの停止または動作は選択可能。 • Hibernate モードでは、いかなる割込みでもリセットが発生。

1) カウンタ値が UPPER_LIMIT に達したとき、または LOWER_LIMIT より前にカウンタがクリアされたときにリセットが発生します。
2) カウンタ値が WARN_LIMIT に達したときに割込みが発生します。
3) fault マネージャがこれを Non-Maskable Interrupt (NMI)といった高い優先度の割込みに変換します。これによりプロセッサはメモリ書込みの停止や周辺機能の解除など安全な状態に戻ることが可能です。
4) MCWDT2_CTR2_CONFIG の BITS[20:16]で定義されているトグルが発生した場合、割込みが発生します。

3.2.2 WDT を使ったウェイクアップ動作の使用例

図 6 に MCWDT のサブカウンタ 0/1 を使った使用例を示します。この例では、MCWDT のサブカウンタ 0 をソフトウェアの暴走監視として使用し、MCWDT のサブカウンタ 1 を定期的な低消費電力モードからの復帰割込みとして使用しています。WDT の詳細については、Architecture reference manual を参照してください。

3 消費電力モードの遷移

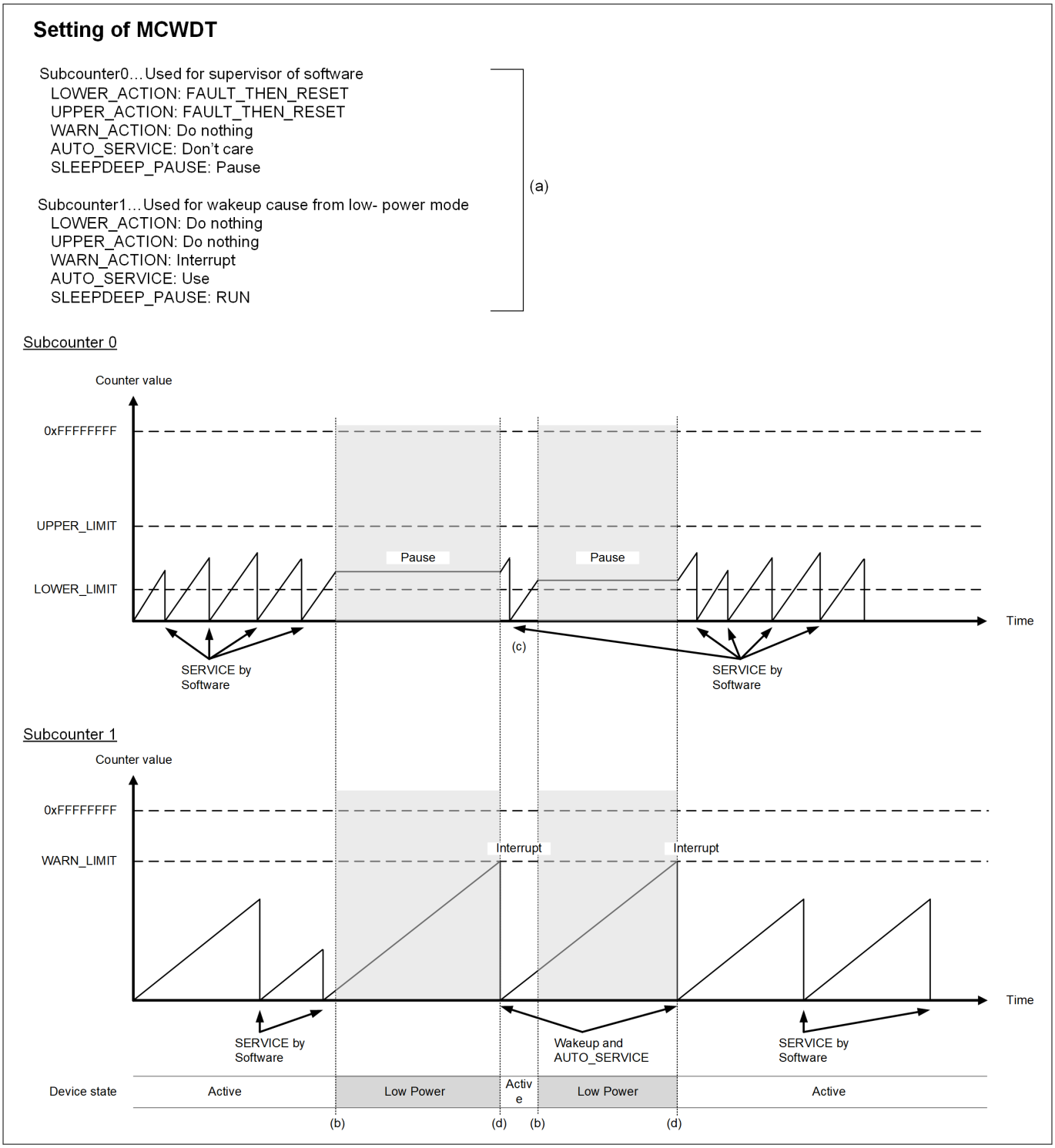

図 6　　WDT 動作例 (ウェイクアップ要因はサブカウンタ 1 の WARN_LIMIT 割込み)

低消費電力モード中、サブカウンタ 0 は動作を停止します。MCU が低消費電力モードから復帰した場合、サブカウンタ 0 はカウントアップを再開します。

低消費電力モード中、サブカウンタ 1 はカウントアップを継続します。カウンタ値が“WARN_LIMIT”に設定した値に到達した場合、MCU は低消費電力モードから復帰します。AUTO_SERVICE 設定を使用した場合、カウンタ値はリセットされます。

3.2.2.1 設定とサンプルコード

低消費電力モードでの WDT 設定における SDL の設定部のパラメーターを表 5 に、関数を表 6 に示します。

表 5 低消費電力モードでの WDT 設定パラメーター一覧

パラメータ	説明	値
.coreSelect	SleepDeepPause に使用する CPU を選択します	CY_MCWDT_PAUSED_BY_NO_CORE
.c0LowerLimit	SleepDeepPause に使用する CPU を選択します	0ul
.c0UpperLimit	SleepDeepPause に使用する CPU を選択します	0xFFFFul
.c0WarnLimit	Subcounter0 警告制限を設定します (符号なし整数 32 ビット)	MCWDT_TICKS_PER_SECOND
.c0LowerAction	Subcounter0 の下限アクションを“no action”, “fault”, または“fault then reset”に設定します	CY_MCWDT_ACTION_FAULT_THEN_RESET
.c0UpperAction	Subcounter0 の上限アクションを“no action”, “fault”, または“fault then reset”に設定します	CY_MCWDT_ACTION_FAULT_THEN_RESET
.c0WarnAction	Subcounter0 警告アクションを“no action”, または“interrupt”に設定します	CY_MCWDT_WARN_ACTION_NONE
.c0AutoService	Subcounter0 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します	CY_MCWDT_DISABLE
.c0SleepDeepPause	対応する CPU が DeepSleep のときに Subcounter0 を一時停止できるようにします	CY_MCWDT_ENABLE
.c0DebugRun	デバッガ設定を設定します。デバッガを使用する場合に必要	CY_MCWDT_ENABLE
.c1LowerLimit	Subcounter1 の下限を設定します (符号なし整数 32 ビット)	0ul
.c1UpperLimit	Subcounter1 の上限を設定します (符号なし整数 32 ビット)	0xFFFFul
.c1WarnLimit	Subcounter1 の警告制限を設定します (符号なし整数 32 ビット)	MCWDT_TICKS_PER_SECOND
.c1LowerAction	Subcounter1 の下限アクションを“no action”, “fault”, または“fault then reset”に設定します	CY_MCWDT_ACTION_NONE
.c1UpperAction	Subcounter1 の上限アクションを“no action”, “fault”, または“fault then reset”に設定します	CY_MCWDT_ACTION_NONE
.c1WarnAction	Subcounter1 警告アクションを“no action”, または“interrupt”に設定します	CY_MCWDT_WARN_ACTION_INT
.c1AutoService	Subcounter1 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します	CY_MCWDT_ENABLE

(続く)

3 消費電力モードの遷移

表 5 (続き) 低消費電力モードでの WDT 設定パラメーター一覧

パラメータ	説明	値
.c1SleepDeepPause	対応する CPU が DeepSleep のときに Subcounter1 を一時停止できるようにします	CY_MCWDT_DISABLE
.c1DebugRun	デバッガ設定を設定します (デバッガを使用する場合に必要)	CY_MCWDT_ENABLE
.c2ToggleBit	トグルを監視するビットを選択します	CY_MCWDT_CNT2_MONITORED_BIT15
.c2Action	Subcounter2 アクションを“no action”または“interrupt”に設定します	CY_MCWDT_CNT2_ACTION_NONE
.c2SleepDeepPause	対応する CPU が DeepSleep のときに Subcounter2 を一時停止できるようにします	CY_MCWDT_ENABLE
.c2DebugRun	デバッガ設定を設定します (デバッガを使用する場合に必要)	CY_MCWDT_ENABLE

表 6 低消費電力モードでの WDT 設定の関数一覧

関数	説明	補足
Cy_MCWDT_DeInit()	MCWDT ブロックを初期化解除し、レジスタ値をデフォルト状態に戻します。	Code Listing 3 を参照してください。
Cy_MCWDT_Init()	MCWDT ブロックを初期化します。	Code Listing 4 を参照してください。
Cy_MCWDT_Unlock()	MCWDT 設定レジスタのロックを解除します。	Code Listing 5 を参照してください。
Cy_MCWDT_SetInterruptMask()	MCWDT 割込みマスクレジスタを書き込みます。	Code Listing 6 を参照してください。
Cy_MCWDT_Enable()	指定されたすべてのカウンタを有効にします	Code Listing 7 を参照してください。
Cy_MCWDT_Lock()	すべての MCWDT レジスタへの設定変更をロックアウトします。	Code Listing 8 を参照してください。
Cy_MCWDT_ClearWatchdog()	XRES デバイスの reset または fault を防ぐため、MC ウォッチドッグカウンタをクリアします。	Code Listing 9 を参照してください。
Cy_SysPm_DeepSleep()	CPU コアを DeepSleep モードへ設定します	Code Listing 10 を参照してください。

Code Listing 1 は、電力モード遷移での WDT ウェイクアップ動作のサンプルプログラムを示します。GPIO および WDT の Architecture reference manual およびアプリケーションノートを参照してください。

3 消費電力モードの遷移

次の説明は、SDL ドライバ部のレジスタ表記を理解するのに役立ちます。

- base は MCWDT レジスタのベースアドレスへのポインタを意味します。カウンタは、MCWDT 内のサブカウンタを指定します。
- レジスタ設定のパフォーマンスを向上させるため、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が生成され、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。

```
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。

3 消費電力モードの遷移

Code Listing 1 消費電力モード遷移での WDT ウェイクアップ動作の例

```
int main(void)
{
    Cy_SysInt_SetSystemIrqVector(srss_interrupt_mcwdt_1_IRQn, irqMCWDT1Handler); /*Assign MCWDT
interrupt*/
    :
    /*MCWDT configuration See (a) of 図 6.
            See Code Listing 3, Code Listing 4, Code Listing 5, Code Listing 6, Code Listing 7,
Code Listing 8
*/
    Cy_MCWDT_DeInit(MCWDT1);
    Cy_MCWDT_Init(MCWDT1, &mcwdtConfig);

    Cy_MCWDT_Unlock(MCWDT1);
    Cy_MCWDT_SetInterruptMask
(MCWDT1, CY_MCWDT_CTR_Msk);
    Cy_MCWDT_Enable(MCWDT1,
                    CY_MCWDT_CTR_Msk,
                    0ul);
    Cy_MCWDT_Lock(MCWDT1);

    /* Put the system to DeeSleep */
    /*See (b) of 図 6. Set to the DeepSleep mode. See Code Listing 10*/
    Cy_SysPm_DeepSleep(CY_SYSPM_WAIT_FOR_INTERRUPT);

    for(;;)
    {
        /* Clear Watchdog counter 0 */
        /*Clears the MCWD counter See (d) of 図 6.  See Code Listing 9*/
        Cy_MCWDT_ClearWatchdog(MCWDT1, CY_MCWDT_COUNTER0);

        while( tFlag == 0ul );
        tFlag = 0ul;
        /*See (b) of  図 6.  See Code Listing 10*/
        Cy_SysPm_DeepSleep(CY_SYSPM_WAIT_FOR_INTERRUPT);
    }
}
```

3 消費電力モードの遷移

Code Listing 2 MCWDT の設定

```
/**
 * \var cy_stc_mcwdt_config_t mcwdtConfig
 * \brief MCWDT configuration
 */
cy_stc_mcwdt_config_t mcwdtConfig = /*Configure MCWDT parameter*/
{
    .coreSelect       = CY_MCWDT_PAUSED_BY_NO_CORE,
    .c0LowerLimit     = 0ul,
    .c0UpperLimit     = 0xFFFFul,
    .c0WarnLimit      = MCWDT_TICKS_PER_SECOND,  /* 1 sec, ignored */
    .c0LowerAction    = CY_MCWDT_ACTION_FAULT_THEN_RESET,
    .c0UpperAction    = CY_MCWDT_ACTION_FAULT_THEN_RESET,
    .c0WarnAction     = CY_MCWDT_WARN_ACTION_NONE,
    .c0AutoService    = CY_MCWDT_DISABLE,
    .c0SleepDeepPause = CY_MCWDT_ENABLE,
    .c0DebugRun       = CY_MCWDT_ENABLE,
    .c1LowerLimit     = 0ul,
    .c1UpperLimit     = 0xFFFFul,
    .c1WarnLimit      = MCWDT_TICKS_PER_SECOND, /* 1 sec */
    .c1LowerAction    = CY_MCWDT_ACTION_NONE,
    .c1UpperAction    = CY_MCWDT_ACTION_NONE,
    .c1WarnAction     = CY_MCWDT_WARN_ACTION_INT,
    .c1AutoService    = CY_MCWDT_ENABLE,
    .c1SleepDeepPause = CY_MCWDT_DISABLE,
    .c1DebugRun       = CY_MCWDT_ENABLE,
    .c2ToggleBit      = CY_MCWDT_CNT2_MONITORED_BIT15,
    .c2Action         = CY_MCWDT_CNT2_ACTION_NONE,
    .c2SleepDeepPause = CY_MCWDT_ENABLE,
    .c2DebugRun       = CY_MCWDT_ENABLE,
};
```

3 消費電力モードの遷移

Code Listing 3 Cy_MCWDT_DeInit() 関数

```
/* 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);

    // 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);
}
```

3 消費電力モードの遷移

Code Listing 4 Cy_MCWDT_Init() 関数

```
/* Initializes the MCWDT block.*/

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))
    {
        Cy_MCWDT_Unlock(base);
        un_MCWDT_CTR_CONFIG_t  tempConfigParams     = { 0ul };
        un_MCWDT_CTR2_CONFIG_t tempCNT2ConfigParams = { 0ul };

        base->unCPU_SELECT.u32Register                    = config->coreSelect;
        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.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;

        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;

        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);

        ret = CY_MCWDT_SUCCESS;
    }

    return (ret);
}
```

3 消費電力モードの遷移

Code Listing 5 Cy_MCWDT_Unlock() 関数

```
/* Unlocks the MCWDT configuration registers.*/

STATIC_INLINE void Cy_MCWDT_Unlock(volatile stc_MCWDT_t *base)
{
    uint32_t interruptState;

    interruptState = Cy_SysLib_EnterCriticalSection();

    base->unLOCK.stcField.u2MCWDT_LOCK = CY_MCWDT_LOCK_CLR0;
    base->unLOCK.stcField.u2MCWDT_LOCK = CY_MCWDT_LOCK_CLR1;

    Cy_SysLib_ExitCriticalSection(interruptState);
}
```

Code Listing 6 Cy_MCWDT_SetInterruptMask() 関数

```
/* Writes MCWDT interrupt mask register.*/

__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;
    }
    if (counters & CY_MCWDT_CTR1)
    {
        base->unINTR_MASK.stcField.u1CTR1_INT = 1ul;
    }
    if (counters & CY_MCWDT_CTR2)
    {
        base->unINTR_MASK.stcField.u1CTR2_INT = 1ul;
    }
}
```

3 消費電力モードの遷移

Code Listing 7 Cy_MCWDT_Enable() 関数

```
/* Enables all specified counters. */

__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;
    }
    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 8 Cy_MCWDT_Lock() 関数

```
/* Locks out configuration changes to all MCWDT registers. */
__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;

    Cy_SysLib_ExitCriticalSection(interruptState);
}
```

Code Listing 9 Cy_MCWDT_ClearWatchdog() 関数

```
/* Clears the MC watchdog counter, to prevent a XRES device reset or fault. */
void Cy_MCWDT_ClearWatchdog(volatile stc_MCWDT_t *base, cy_en_mcwdtctr_t counter)
{
    Cy_MCWDT_Unlock(base);
    Cy_MCWDT_ResetCounters(base, (1u << (uint8_t)counter), 0u);
    Cy_MCWDT_Lock(base);
}
```

3 消費電力モードの遷移

Code Listing 10 Cy_SysPm_DeepSleep() 関数

```
/* Sets a CPU core to the DeepSleep mode */
cy_en_syspm_status_t Cy_SysPm_DeepSleep(cy_en_syspm_waitfor_t waitFor)
{
    uint32_t interruptState;
    cy_en_syspm_status_t retVal = CY_SYSPM_SUCCESS;

    /* Call the registered callback functions with
    * the CY_SYSPM_CHECK_READY parameter.
    */
    if(0u != currentRegisteredCallbacksNumber)
    {
        retVal = Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_CHECK_READY);
    }

    /* The device (core) can switch into the deep sleep power mode only when
    *  all executed registered callback functions with the CY_SYSPM_CHECK_READY
    *  parameter returned CY_SYSPM_SUCCESS.
    */
    if(retVal == CY_SYSPM_SUCCESS)
    {
        /* Call the registered callback functions with the CY_SYSPM_BEFORE_TRANSITION
        * parameter. The return value is ignored.
        */
        interruptState = Cy_SysLib_EnterCriticalSection();
        if(0u != currentRegisteredCallbacksNumber)
        {
            (void) Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_BEFORE_ENTER);
        }

    #if(0u != CY_CPU_CORTEX_M0P)

        /* The CPU enters the deep sleep mode upon execution of WFI/WFE */
        SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;

        if(waitFor != CY_SYSPM_WAIT_FOR_EVENT)
        {
            __WFI();
        }
        else
        {
            __WFE();
        }
    #else

        /* Repeat WFI/WFE instructions if wake up was not intended.
        *  Cypress Ticket #272909
        */
        do
        {
            SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
```

```
            if(waitFor != CY_SYSPM_WAIT_FOR_EVENT)
            {
                __WFI();
            }
            else
            {
                __WFE();
            }
        } while (0); //rmkn _FLD2VAL(CPUSS_CM4_PWR_CTL_PWR_MODE, CPUSS-
>unCM4_PWR_CTL.u32Register) == CY_SYSPM_CM4_PWR_CTL_PWR_MODE_RETAINED);

    #endif /* (0u != CY_CPU_CORTEX_M0P) */

        Cy_SysLib_ExitCriticalSection(interruptState);

        /* Call the registered callback functions with the CY_SYSPM_AFTER_TRANSITION
        *  parameter. The return value is ignored.
        */
        if(0u != currentRegisteredCallbacksNumber)
        {
            (void) Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_AFTER_EXIT);
        }
    }
    else
    {
        /* Execute callback functions with the CY_SYSPM_CHECK_FAIL parameter to
        * undo everything done in the callback with the CY_SYSPM_CHECK_READY
        * parameter. The return value is ignored.
        */
        (void) Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_CHECK_FAIL);
        retVal = CY_SYSPM_FAIL;
    }
    return retVal;
}
```

3.3 Cyclic ウェイクアップ動作

Cyclic ウェイクアップ動作は、MCU の間欠動作です。例えば、Electronic Control Unit (ECU) がスリープモード中、MCU は定期的に DeepSleep モードへの遷移および復帰を繰り返します。この動作により、アプリケーションの平均消費電力を最小限にすることを目的とします。ここでは、TRAVEO™ T2G ファミリ MCU を使った Cyclic ウェイクアップ動作の実装例を示します。

3.3.1 Cyclic ウェイクアップの使用例

図 7 にユーザシステムの例を示します。外部デバイスやセンサなどを監視するため、MCU は GPIO と ADC を制御します。

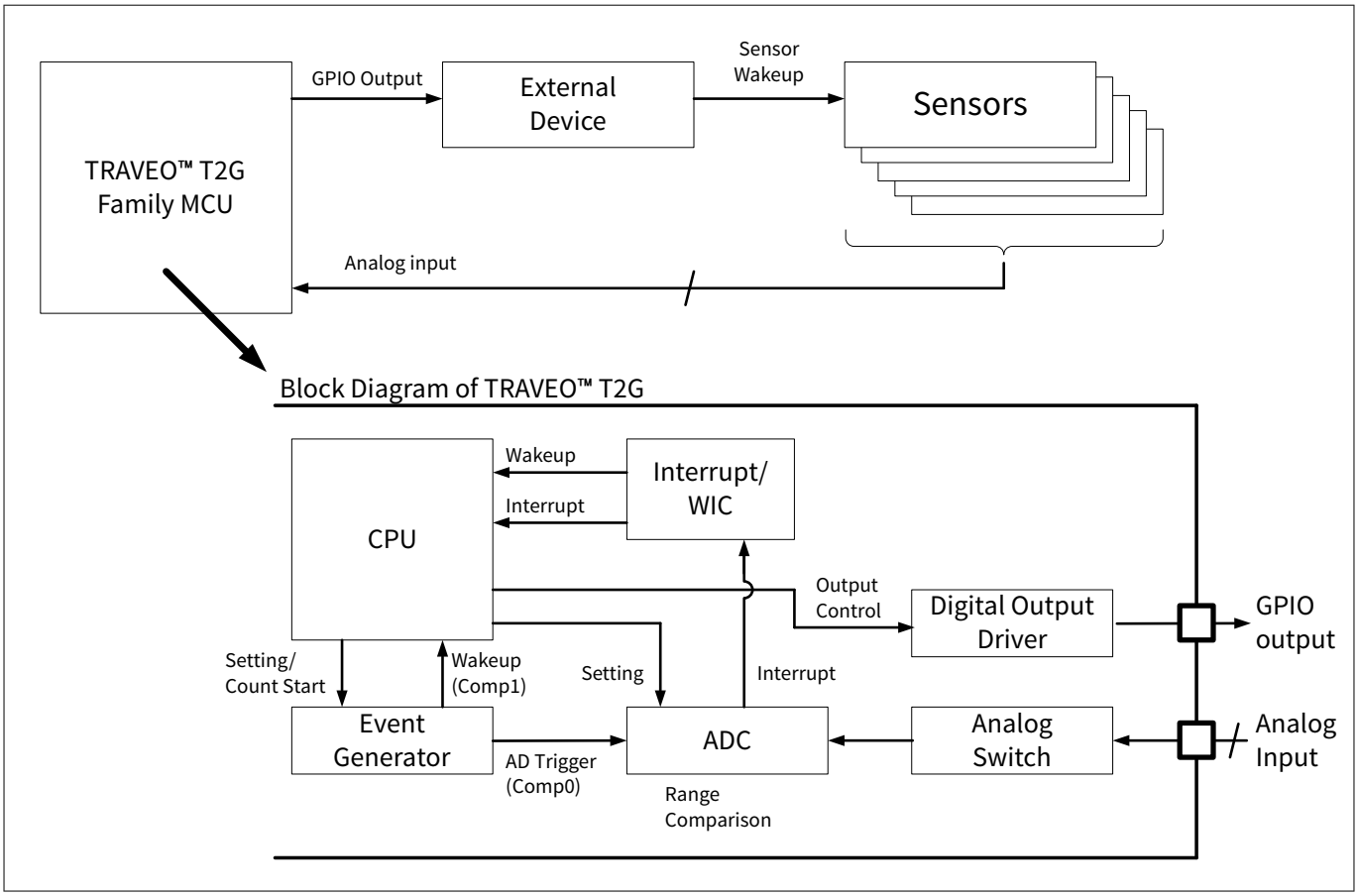

図 7　　ブロックダイヤグラム (ユーザシステムの例)

外部デバイスは、GPIO を介して MCU と接続されます。MCU からの制御信号により、外部デバイスはセンサを起動させ、センサの出力は MCU の ADC に取り込まれます。

しかし、一般的にセンサは起動直後、正確な出力をするために安定待ち時間が必要です。

そのため、MCU は外部デバイスに対してセンサ起動用の制御信号を出力し、ある一定の時間を待った後、ADC にてセンサからの信号を AD 変換します。2 つの異なるタイミングを作り出すため、イベントジェネレータの Comp0 と Comp1 の 2 つのタイマ割込みを使用します。このケースでは、Comp0 を ADC 起動のトリガとし、Comp1 を CPU のウェイクアップに使用します。

3.3.2　　Cyclic ウェイクアップ動作

図 8 に Cyclic ウェイクアップ動作の概念を示します。ECU がスリープモードのとき、外部センサの情報をチェックするために、TRAVEO™ T2G デバイスは定期的にウェイクアップします。

3 消費電力モードの遷移

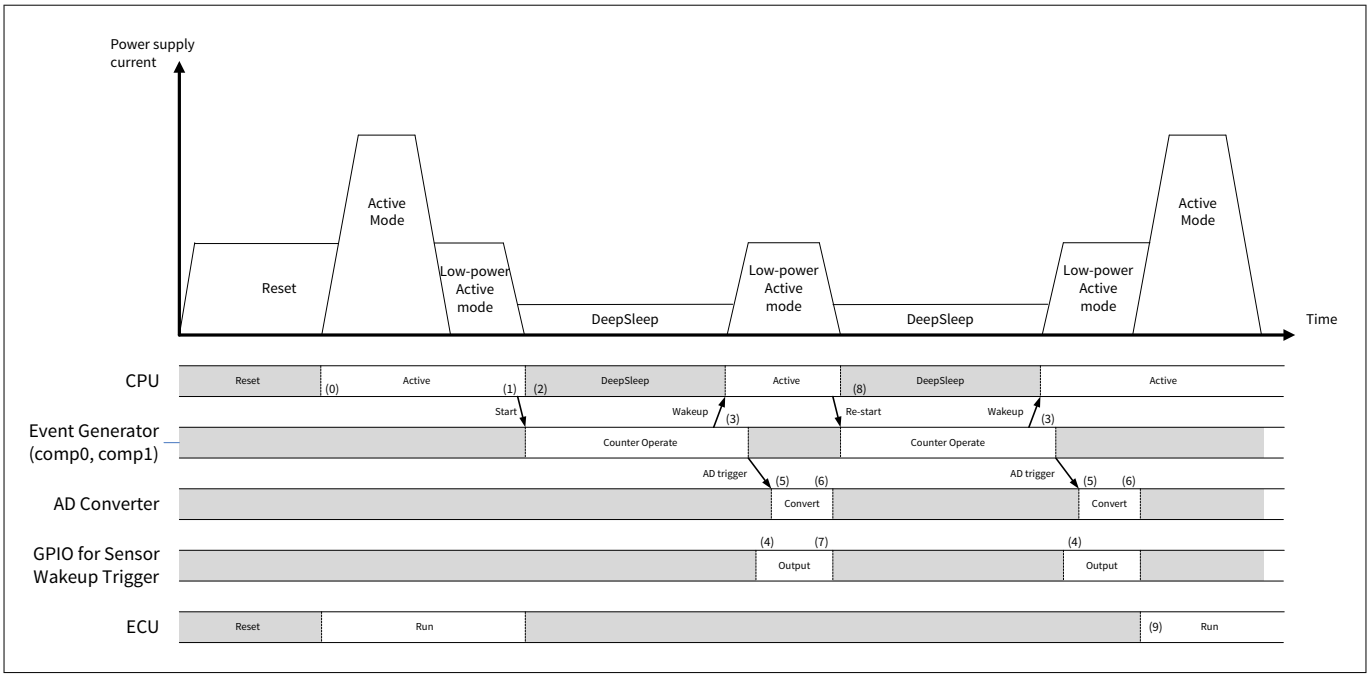

図 8　　Cyclic ウェイクアップ動作

(0) リセット後、TRAVEO™ T2G は Active モードになりユーザソフトウェアを実行します。

(1) CPU は、32.768 kHz の低周波発振器を使ってイベントジェネレータを起動させます。

(イベントジェネレータのソースクロックは、ILO0, ILO1, および WCO から選択可能です。)

(2) TRAVEO™ T2G は DeepSleep モードに遷移し、CPU コアは DeepSleep 状態になります。

(3) もしイベントジェネレータのカウンタ値が Comp1 と一致した場合、Comp1 トリガによって CPU コアがウェイクアップします。

(4) CPU (ソフトウェア) は外部デバイスを起動させるための信号を出力するため GPIO を制御します。

(5) Comp0 のトリガにより ADC のレンジ比較動作が開始します。

(6) CPU (ソフトウェア) は ADC の変換結果をチェックします。

(7) CPU (ソフトウェア) は GPIO を制御します。

[もし ADC の変換結果が範囲内であれば、ECU は Cyclic ウェイクアップ動作を継続します。]

(8) CPU (ソフトウェア) はイベントジェネレータを再始動します。CPU は DeepSleep モードに遷移します。[(2)へ]

[もし ADC の変換結果が範囲外だった場合、ECU は Cyclic ウェイクアップ動作を終了します。]

(9) CPU (ソフトウェア) は ECU システムの動作を再始動させます。

イベントジェネレータ, ADC, GPIO, およびクロックについての詳細は Architecture reference manual を参照してください。

3.3.3 Cyclic ウェイクアップ動作のフローチャート

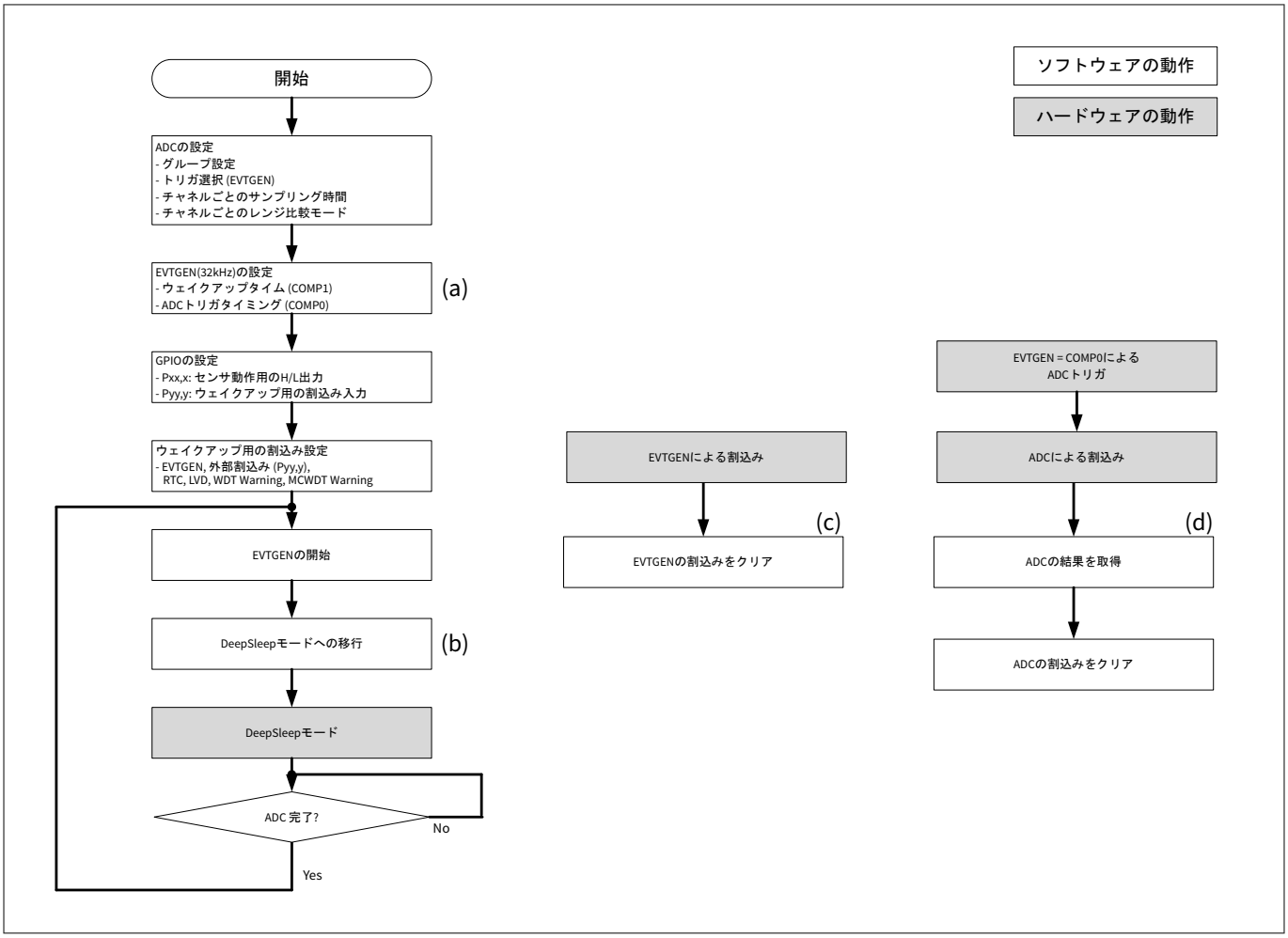

図 9 Cyclic ウェイクアップ動作のフローチャート

注: 図 9 中の灰色のボックスはハードウェアの動作を示します。したがって、ソフトウェアによる処理は必要ありません。

3.3.3.1 設定とサンプルコード

Cyclic ウェイクアップ動作の SDL の設定部のパラメーターを表 7 に、関数を表 8 を示します。

表 7 Cyclic ウェイクアップ操作の設定パラメーターの一覧

パラメーター	説明	値
.frequencyRef	clk_ref	8000000
.frequencyLf	clk_lf	32000
.frequencyTick	イベントジェネレータクロック (clk_ref_div)	1000000 (1,000,000 Hz 設定)
.ratioControlMode	イベントジェネレータ比制御モード	CY_EVTGEN_RATIO_CONTROL_HW

(続く)

表 7 (続き) Cyclic ウェイクアップ操作の設定パラメーターの一覧

パラメーター	説明	値
.ratioValueDynamicMode	イベントジェネレータダイナミックモード	CY_EVTGEN_RATIO_DYNAMIC_MODE0
.functionalitySelection	イベントジェネレータ選択機能	CY_EVTGEN_DEEPSLEEP_FUNCTIONALITY
.triggerOutEdge	イベントジェネレータトリガ	CY_EVTGEN_EDGE_SENSITIVE
.valueDeepSleepComparator	時間の経過後に CPU をウェイクアップ	1000000 (1 秒)
.valueActiveComparator	時間の経過後に ADC をトリガ	1500000 (1.5 秒)
.userIsr	イベントジェネレータの構造体ハンドラ	NULL

表 8 Cyclic ウェイクアップ動作設定命令の一覧

関数	説明	備考
AdcIntHandler()	ADC の割込みハンドラ	Code Listing 11 を参照してください。
irqEVTGEN_sleep()	イベントジェネレータの割込み設定	Code Listing 11 を参照してください。
Cy_Evtgen_ClearStructInterruptDeepSleep()	対応する構造の DeepSleep 割込み要因をクリア	Code Listing 12 を参照してください。
Cy_Evtgen_DeinitializeCompStruct()	イベントジェネレータ構造の初期化を解除	Code Listing 13 を参照してください。
Cy_Evtgen_Deinitialize()	イベントジェネレータの初期化を解除	Code Listing 14 を参照してください。
Cy_Evtgen_Initialize()	イベントジェネレータを初期化	Code Listing 15 を参照してください。
Cy_Evtgen_InitializeCompStruct()	コンパレータ構造を初期化	Code Listing 16 を参照してください。
Cy_SysInt_SetSystemIrqVector()	システム割込みのユーザー ISR ベクタを変更	Code Listing 18 を参照してください。
Cy_SysPm_DeepSleep()	CPU コアを DeepSleep モードに設定	Code Listing 17 を参照してください。

Code Listing 11 に、Cyclic ウェイクアップ動作のサンプルプログラムを示します。GPIO および ADC の Architecture reference manual およびアプリケーションノートを参照してください。

3 消費電力モードの遷移

Code Listing 11 Cyclic ウェイクアップ動作の例

```
/* Eventgenerator Configration */

const cy_stc_evtgen_config_t evtgenTestConfig =
{
  #else
    .frequencyRef          = 8000000,  //  clk_ref = clk_hf1
    .frequencyLf           = 32000,    // clk_lf = 32,000 for silicon
  #endif
    .frequencyTick         = 1000000,  // Setting 1,000,000 Hz for event generator clock
    .ratioControlMode      = CY_EVTGEN_RATIO_CONTROL_HW,
    .ratioValueDynamicMode = CY_EVTGEN_RATIO_DYNAMIC_MODE0,
};


const cy_stc_evtgen_struct_config_t evtgenTestStructureConfig =
{
    .functionalitySelection   = CY_EVTGEN_DEEPSLEEP_FUNCTIONALITY,
    .triggerOutEdge           = CY_EVTGEN_EDGE_SENSITIVE,
    .valueDeepSleepComparator = 1000000, // It wake CPU up after 1s.
    .valueActiveComparator    = 1500000, // It triggers ADC after 1.5s.
    .userIsr = NULL,
};

/* See (d) of 図 9.  Interrupt handler for ADC */
void AdcIntHandler(void)
{
   cy_stc_adc_interrupt_source_t intrSource = { false };
    /* Get the result(s) */
    Cy_Adc_Channel_GetResult(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&resultBuff[resultIdx], &statusBuff[resultIdx]);
    resultIdx = (resultIdx + 1) % (sizeof(resultBuff) / sizeof(resultBuff[0]));
    /* Clear inerrupt source */
    Cy_Adc_Channel_GetInterruptMaskedStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&intrSource);
    Cy_Adc_Channel_ClearInterruptStatus(&BB_POTI_ANALOG_MACRO->CH[ADC_LOGICAL_CHANNEL],
&intrSource);

    adcCompletedFlag = 1;
}

void irqEVTGEN_sleep(void)
{
/* See (c) of 図 9. See Code Listing 12 */
    Cy_Evtgen_ClearStructInterruptDeepSleep(EVTGEN0,0);
}

int main(void)
{
:
/* Event generator configuration.See (a) of 図 9.  */
    /*****************************************/
```

```
    /*        Deinitialize peripherals         */
    /*******************************************/
    Cy_Evtgen_DeinitializeCompStruct(EVTGEN0, 0);  /* See Code Listing 13 */
            Cy_Evtgen_Deinitialize(EVTGEN0);   /* See Code Listing 14 */

    /*******************************************/
    /*  Initialize and start Event generator   */
    /*******************************************/
    Cy_Evtgen_Initialize(EVTGEN0,&evtgenTestConfig);  /* See Code Listing 15 */

    /*******************************************/
    /*  Initialize comparator structure 0      */
    /*******************************************/
    Cy_Evtgen_InitializeCompStruct(EVTGEN0, 0,   /* See Code Listing 16 */
&evtgenTestStructureConfig, &evtgenStruct0Context);

    /* Register ADC interrupt handler and enable interrupt */
    Cy_SysInt_SetSystemIrqVector(irq_cfg_adc.sysIntSrc, AdcIntHandler);

    /* Put the system to DeeSleep */
    Cy_SysPm_DeepSleep((cy_en_syspm_waitfor_t)CY_SYSPM_WAIT_FOR_INTERRUPT);

/* See (b) of 図 9. Set to the DeepSleep mode. See Code Listing 17 */
    for(;;)
    {
        while(adcCompletedFlag == 0);
        adcCompletedFlag = 0;

        Cy_Evtgen_DeinitializeCompStruct(EVTGEN0, 0);
        Cy_Evtgen_InitializeCompStruct(EVTGEN0, 0, &evtgenTestStructureConfig,
&evtgenStruct0Context);
        /* Put the system to DeeSleep */
        Cy_SysPm_DeepSleep((cy_en_syspm_waitfor_t)CY_SYSPM_WAIT_FOR_INTERRUPT);
    }

}
```

Code Listing 12 Cy_Evtgen_ClearStructInterruptDeepSleep() 関数

```
/* Clear DeepSleep interrupt factor of corresponding structure */
__STATIC_INLINE void Cy_Evtgen_ClearStructInterruptDeepSleep(volatile stc_EVTGEN_t *base,
uint8_t structNumber)
{
    base->unINTR_DPSLP.u32Register = ((uint32_t)1 << structNumber);

    // Dummy read. This is to wait for reflection above write operation.
    base->unINTR_DPSLP;
}
```

3 消費電力モードの遷移

Code Listing 13 Cy_Evtgen_DeinitializeCompStruct() 関数

```
void Cy_Evtgen_DeinitializeCompStruct(volatile stc_EVTGEN_t *base, uint8_t structNum)
{
    /* Deinitialize event generator structure */
    base->COMP_STRUCT[structNum].unCOMP_CTL.u32Register = 0;
    base->COMP_STRUCT[structNum].unCOMP0.stcField.u32INT32 = 0;
    base->COMP_STRUCT[structNum].unCOMP1.stcField.u32INT32 = 0;
    evtgenContext[structNum] = NULL;
}
```

Code Listing 14 Cy_Evtgen_Deinitialize() 関数

```
void Cy_Evtgen_Deinitialize(volatile stc_EVTGEN_t *base)
{
    /* Deinitialize event generator */
    base->unCTL.u32Register = 0;
    base->unREF_CLOCK_CTL.u32Register = 0;
    base->unRATIO.u32Register = 0;
    base->unRATIO_CTL.u32Register = 0;
    base->unINTR_MASK.u32Register = 0;
    base->unINTR_DPSLP_MASK.u32Register = 0;
}
```

Code Listing 15 Cy_Evtgen_Initialize() 関数

```
cy_en_evtgendrv_status_t Cy_Evtgen_Initialize(volatile stc_EVTGEN_t *base, const
cy_stc_evtgen_config_t* config)
{

/* Initialize the event generator */
uint16_t refDiv;

    un_EVTGEN_RATIO_CTL_t ratioCtl;

    /* 1. Checking input parameter valid */
    if(config == NULL)
    {
        return CY_EVTGEN_ERR;
    }

    /* 2. Initialize internal variable */
    for(uint32_t i = 0; i < EVTGEN_COMP_STRUCT_NR; i++)
    {
        evtgenContext[i] = NULL;
    }
    mapUsed = 0;

    Cy_Evtgen_Enable(base);

    /* 2. Setting divider value of clk_ref */
    refDiv = config->frequencyRef / config->frequencyTick;
    if(config->frequencyRef % config->frequencyTick != 0)
    {
        return CY_EVTGEN_ERR;
    }
    else if(refDiv > 256 || refDiv  < 1)
    {
        return CY_EVTGEN_ERR;
    }
    else
    {
        base->unREF_CLOCK_CTL.stcField.u8INT_DIV = refDiv - 1u;
    }

    /* 3. Setting ratio operation */
    if(config->ratioControlMode == CY_EVTGEN_RATIO_CONTROL_SW)
    {
        /* SW controll: setting value for ratio value should be ratio between tick_ref_div and
clk_lf. */
        uint64_t temp = (uint64_t)(config->frequencyRef / refDiv) << EVTGEN_RATIO_INT16_Pos;
        temp = temp / (uint64_t)(config->frequencyLf);
        base->unRATIO.u32Register = ((uint32_t)temp) & (EVTGEN_RATIO_INT16_Msk |
EVTGEN_RATIO_FRAC8_Msk);

        base->unRATIO_CTL.stcField.u1DYNAMIC = 0u;
```

3 消費電力モードの遷移

```
            /* SW controll: valid bit should be set manually. */
            base->unRATIO_CTL.stcField.u1VALID = 1u; /* Set VALID bit */
        }
        else
        {
            /* HW controll: */
            ratioCtl.u32Register = base->unRATIO_CTL.u32Register;
            ratioCtl.stcField.u1DYNAMIC = 1u; /* Set Dynamic bit */
            ratioCtl.stcField.u3DYNAMIC_MODE = config->ratioValueDynamicMode; /* Set Dynamic bit */
            base->unRATIO_CTL.u32Register = ratioCtl.u32Register;

            /* Waiting until valid bit is set. */
            while(base->unRATIO_CTL.stcField.u1VALID == 0u);
        }

        /* Waiting until counter become ready. */
        while(base->unCOUNTER_STATUS.stcField.u1VALID == 0u);

        return CY_EVTGEN_OK;
    }
```

3 消費電力モードの遷移

Code Listing 16 Cy_Evtgen_InitializeCompStruct() 機能

```
cy_en_evtgendrv_status_t Cy_Evtgen_InitializeCompStruct(volatile stc_EVTGEN_t *base,
                                                        uint8_t structNum,
                                                        const cy_stc_evtgen_struct_config_t*
configStruct,
                                                        cy_stc_evtgen_struct_context_t* context)
{

/* Initialize a comparator structure */

un_EVTGEN_COMP_STRUCT_COMP_CTL_t compCtr;
    uint64_t tempCounterValue;
    uint32_t savedIntrStatus;

    /* Checking input parameter valid */
    if(configStruct == NULL)
    {
        return CY_EVTGEN_ERR;
    }

    if(structNum >= EVTGEN_COMP_STRUCT_NR)
    {
        return CY_EVTGEN_ERR;
    }

    if(configStruct->functionalitySelection != CY_EVTGEN_DEEPSLEEP_FUNCTIONALITY)
    {
        if(context == NULL)
        {
            return CY_EVTGEN_ERR;
        }
        evtgenContext[structNum] = context;
        evtgenContext[structNum]->addValueForCOMP0 = configStruct->valueActiveComparator;
        evtgenContext[structNum]->userIsr = configStruct->userIsr;
        mapUsed |= 1 << structNum;
    }

    compCtr.u32Register = base->COMP_STRUCT[structNum].unCOMP_CTL.u32Register;

    if(configStruct->functionalitySelection == CY_EVTGEN_DEEPSLEEP_FUNCTIONALITY)
    {
        compCtr.stcField.u1COMP1_EN = 1u;
    }

    compCtr.stcField.u1COMP0_EN = 1u;

    compCtr.stcField.u1TR_OUT_EDGE = configStruct->triggerOutEdge;

    compCtr.stcField.u1ENABLED = 1u;

    savedIntrStatus = Cy_SysLib_EnterCriticalSection();
```

3 消費電力モードの遷移

```
    tempCounterValue = (uint64_t)Cy_Evtgen_GetCounterValue(base);

    /* Setting active comparator value */
    base->COMP_STRUCT[structNum].unCOMP0.stcField.u32INT32 = (uint32_t)(tempCounterValue +
(uint64_t)configStruct->valueActiveComparator);

    /* Setting deep sleep comparator value */
    if(configStruct->functionalitySelection == CY_EVTGEN_DEEPSLEEP_FUNCTIONALITY)
    {
        base->COMP_STRUCT[structNum].unCOMP1.stcField.u32INT32 = (uint32_t)(tempCounterValue +
(uint64_t)configStruct->valueDeepSleepComparator);
    }

    Cy_SysLib_ExitCriticalSection(savedIntrStatus);

    /* Setting comparator struct controll parameter */
    base->COMP_STRUCT[structNum].unCOMP_CTL.u32Register = compCtr.u32Register;

    Cy_Evtgen_SetInterruptMask(base, structNum);
    if(configStruct->functionalitySelection == CY_EVTGEN_DEEPSLEEP_FUNCTIONALITY)
    {
        Cy_Evtgen_SetInterruptDeepSleepMask(base, structNum);
    }

    return CY_EVTGEN_OK;
}
```

3 消費電力モードの遷移

Code Listing 17 Cy_SysPm_DeepSleep() 関数

```
cy_en_syspm_status_t Cy_SysPm_DeepSleep(cy_en_syspm_waitfor_t waitFor)
{

    /* Sets a CPU core to the DeepSleep  mode */
    uint32_t interruptState;
    cy_en_syspm_status_t retVal = CY_SYSPM_SUCCESS;

    /* Call the registered callback functions with
    * the CY_SYSPM_CHECK_READY parameter.
    */
    if(0u != currentRegisteredCallbacksNumber)
    {
        retVal = Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_CHECK_READY);
    }

    /* The device (core) can switch into the deep sleep power mode only when
    *  all executed registered callback functions with the CY_SYSPM_CHECK_READY
    *  parameter returned CY_SYSPM_SUCCESS.
    */
    if(retVal == CY_SYSPM_SUCCESS)
    {
        /* Call the registered callback functions with the CY_SYSPM_BEFORE_TRANSITION
        * parameter. The return value is ignored.
        */
        interruptState = Cy_SysLib_EnterCriticalSection();
        if(0u != currentRegisteredCallbacksNumber)
        {
            (void) Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_BEFORE_ENTER);
        }

    #if(0u != CY_CPU_CORTEX_M0P)

        /* The CPU enters the deep sleep mode upon execution of WFI/WFE */
        SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;

        if(waitFor != CY_SYSPM_WAIT_FOR_EVENT)
        {
            __WFI();
        }
        else
        {
            __WFE();
        }
    #else

        /* Repeat WFI/WFE instructions if wake up was not intended.
        *  Cypress Ticket #272909
        */
        do
        {
            SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
```

```
            if(waitFor != CY_SYSPM_WAIT_FOR_EVENT)
            {
                __WFI();
            }
            else
            {
                __WFE();
            }
        } while (0); //rmkn _FLD2VAL(CPUSS_CM4_PWR_CTL_PWR_MODE, CPUSS-
>unCM4_PWR_CTL.u32Register) == CY_SYSPM_CM4_PWR_CTL_PWR_MODE_RETAINED);

    #endif /* (0u != CY_CPU_CORTEX_M0P) */

        Cy_SysLib_ExitCriticalSection(interruptState);

        /* Call the registered callback functions with the CY_SYSPM_AFTER_TRANSITION
        *  parameter. The return value is ignored.
        */
        if(0u != currentRegisteredCallbacksNumber)
        {
            (void) Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_AFTER_EXIT);
        }
    }
    else
    {
        /* Execute callback functions with the CY_SYSPM_CHECK_FAIL parameter to
        * undo everything done in the callback with the CY_SYSPM_CHECK_READY
        * parameter. The return value is ignored.
        */
        (void) Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_CHECK_FAIL);
        retVal = CY_SYSPM_FAIL;
    }
    return retVal;
}
```

Code Listing 18 Cy_SysInt_SetSystemIrqVector() 機能

```
void  Cy_SysInt_SetSystemIrqVector(cy_en_intr_t sysIntSrc, cy_systemIntr_Handler userIsr)
{

    /* Changes the User ISR vector for the System Interrupt */
    if (Cy_SysInt_SystemIrqUserTableRamPointer != NULL)
    {
        Cy_SysInt_SystemIrqUserTableRamPointer[sysIntSrc] = userIsr;
    }
}
```

3.3.4 Cyclic ウェイクアップでのスマート I/O の使用

ここでは、Cyclic ウェイクアップ動作で LPACTIVE 期間を短縮する際のスマート I/O の役割について説明します。

Cyclic ウェイクアップ動作のセクションの冒頭で説明したように、Cyclic ウェイクアップは、アプリケーションの平均消費電力を最小限に抑えることを目的としています。この平均消費電流は、次の影響を受けます。

- DeepSleep 電流
- LPACTIVE 電流
- 1 つの期間の LPACTIVE 時間のパーセンテージ

DeepSleep 電流と LPACTIVE 電流は、主に HW の電気的仕様に依存しますが、1 期間の LPACTIVE 時間の割合は、SW の最適化に依存します。LPACTIVE (数 mA) 中の消費電流は DeepSleep (数 10µA) よりも比較的高いため、SW の観点から、LPACTIVE 時間を短縮することは、低い平均消費電流を達成するための重要な要素です。LPACTIVE 期間を短縮するには、スマート I/O の使用を推奨します。

図 8 で提案されている Cyclic ウェイクアップ操作のフローでは、MCU はウェイクアップしてセンサをオンにするように I/O ポートを設定し、センサの安定化を待ってから ADC 変換をトリガする必要があります。長いセンサ安定化時間が必要な場合は、オプションで MCU を再度 Sleep/DeepSleep に入れて消費電流を減らすことができますが、このアプローチではプログラムがより複雑になる可能性があります。さらに、異なる電力モード間の遷移時間を考慮する必要があります。

図 10 に、このユースケースのシステム設定を示します。GPIO は、図 7 に示すように、CPU ではなくスマート I/O によってアクティブ化されます。

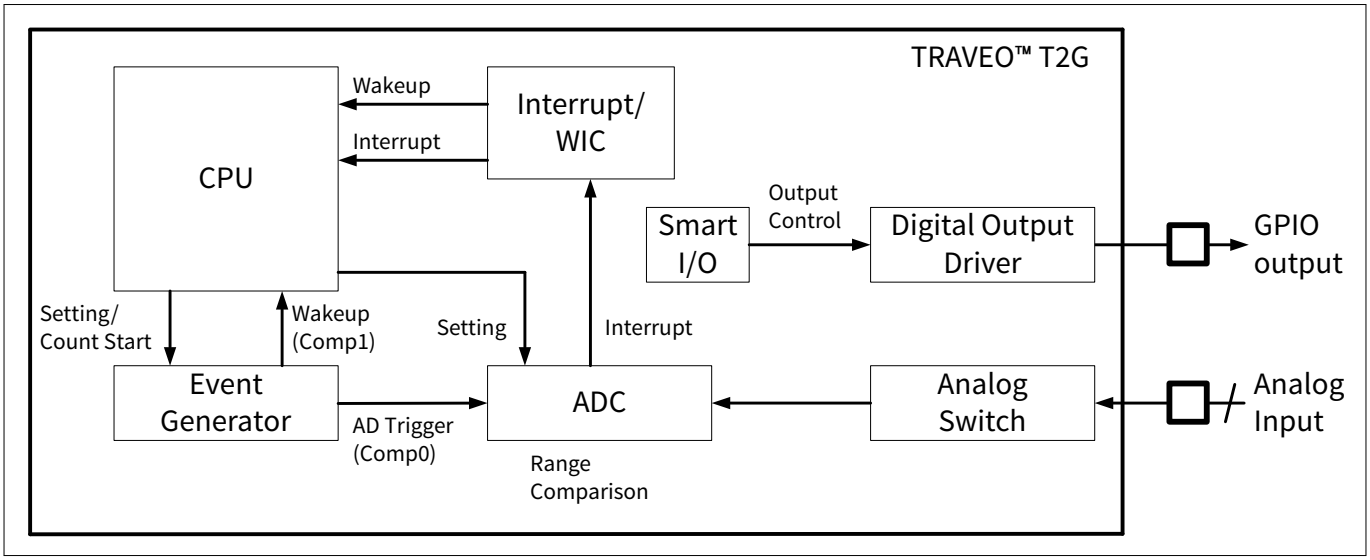

図 10　　スマート I/O を使用した Cyclic ウェイクアップのシステム設定

3.3.4.1　Cyclic ウェイクアップにおけるスマート I/O 実装の利点

スマート I/O を使用して、DeepSleep モード中に I/O を操作できます。これにより、特にセンサが安定するまで CPU が長時間待機する必要がある場合に、CPU が I/O をアクティブ化するために費やす不要な LPACTIVE 時間をなくすことができます。

スマート I/O の内部ロジックには、他のコンポーネントの中でも特に 3 入力ルックアップテーブル (LUT) とデータユニット (DU) が含まれます。DU はカウントアップ/カウントダウンとリロード機能を備えたカウンタとして機能します。データユニットと LUT を適切に設定することにより、GPIO が必要なレイテンシで High 出力するのを遅らせる回路を作成できます。

図 11 に示すように、スマート I/O は DeepSleep で動作できるため、CPU が DeepSleep モードのときに GPIO をアクティブ化するために使用できます。

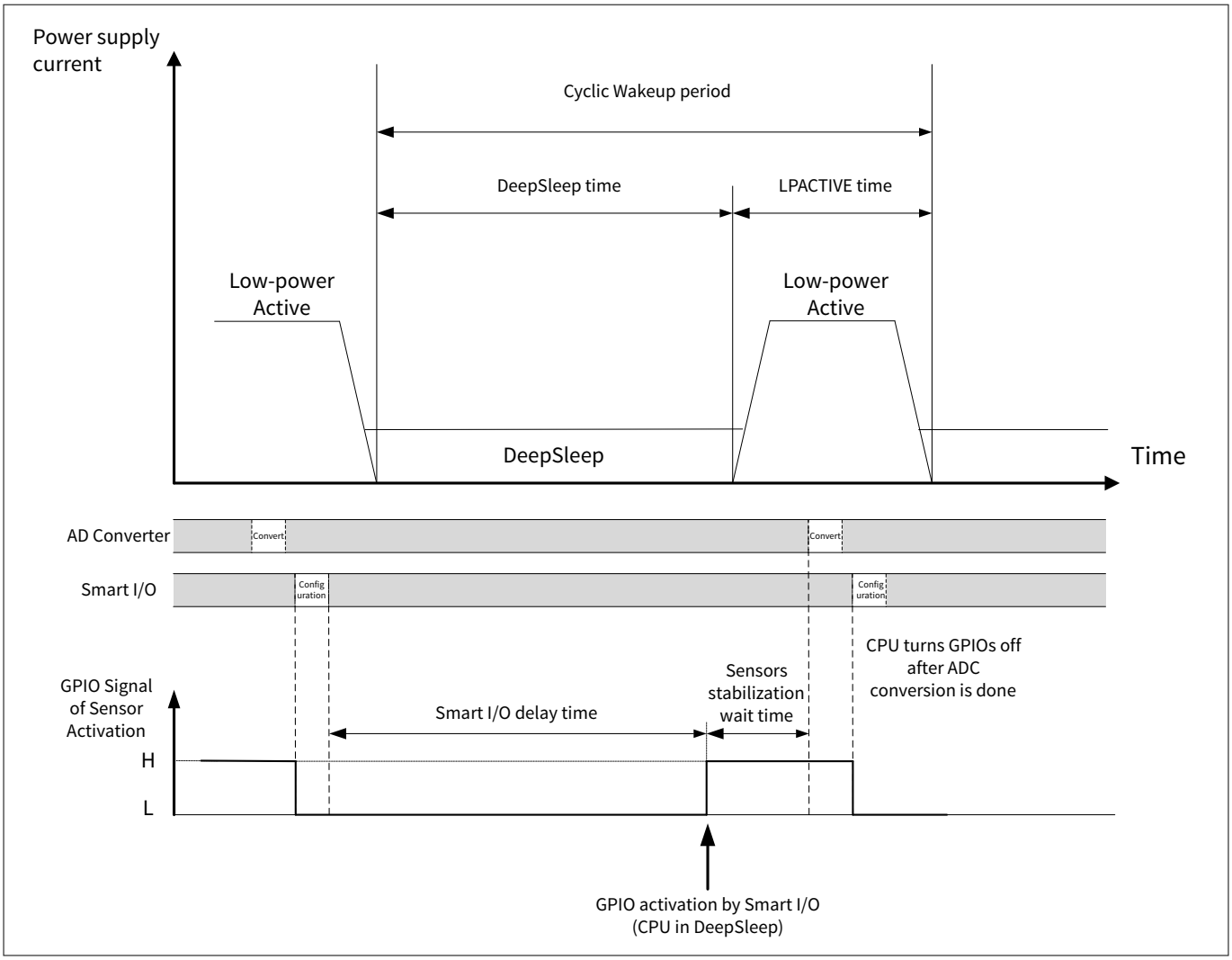

図 11　　スマート I/O ベースの cyclic ウェイクアップ操作

3.3.4.2　Cyclic ウェイクアップでのスマート I/O 設定

データユニット (DU) は、“H”出力を遅延させるためのカウンタとして使用できます。ただし、DU は 8 ビットカウンタのみです。DeepSleep 中、スマート I/O のソースクロックは周波数 32kHz の ILO であるため、DU は次のように最大間隔をカウントできます。

$$\frac{2^8}{32 \times 10^3} \times 10^3 \; = \; 8 \left[ms \right]$$

したがって、8 ms を超える周期的なウェイクアップ期間を必要とするアプリケーションの場合、カウンタ用に追加のビットが必要です。

このように、この例では、「センサ起動回路」と呼ばれるスマート I/O による 11 ビットタイマ相当の回路が実装されています。11 ビットのカウンタと同等の回路は、最大 32 ms の遅延を生成できます。図 12 に回路の動作を示します。ここで、「DAT」は DU の上限です。DAT は、SMARTIO_PRTx_DATA レジスタによって設定されます。カウント値が DAT に等しい場合、データユニット tr_out に単一のクロックパルスが出力されます。

3 消費電力モードの遷移

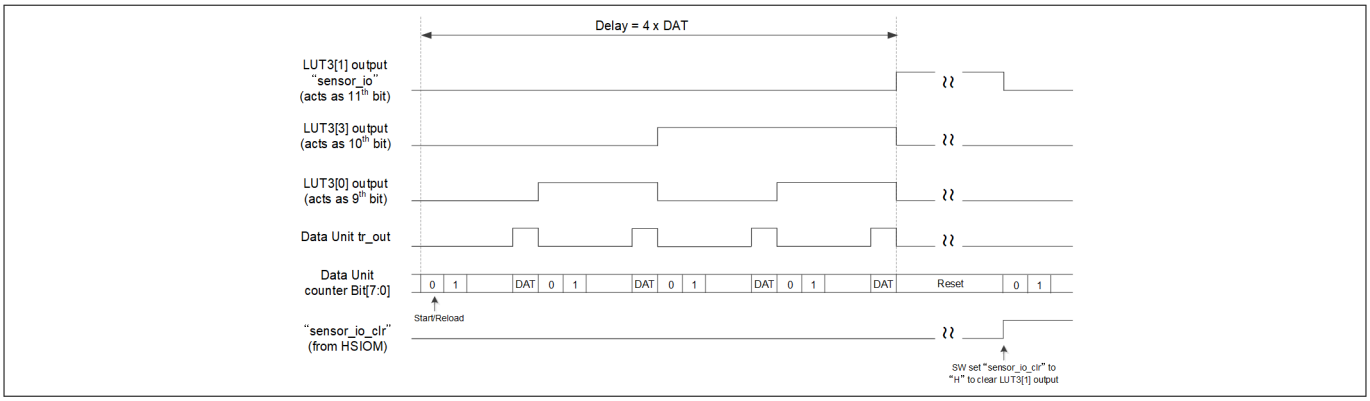

図 12　　センサ起動回路の動作

この使用例では、回路は HSIOM から“sensor_io_clr”信号という名前の 1 つの信号を受信し、アクティブな HIGH を使用して、LUT3 [1]の出力、つまりセンサのアクティブ化出力をクリアします。次の I/O ポートと HSIOM 信号が使用されます。

- smartio_data[1]= Sensor_io (I/O ポート、つまり外部センサーアクティベーションポートへ)
- chip_data[1]= Sensor_io_clr (HSIOM から、“sensor_io”をクリアするために CPU によって操作されます)

図 13 に、この回路の各 LUT3 と DU の接続と機能ロジックを示します。

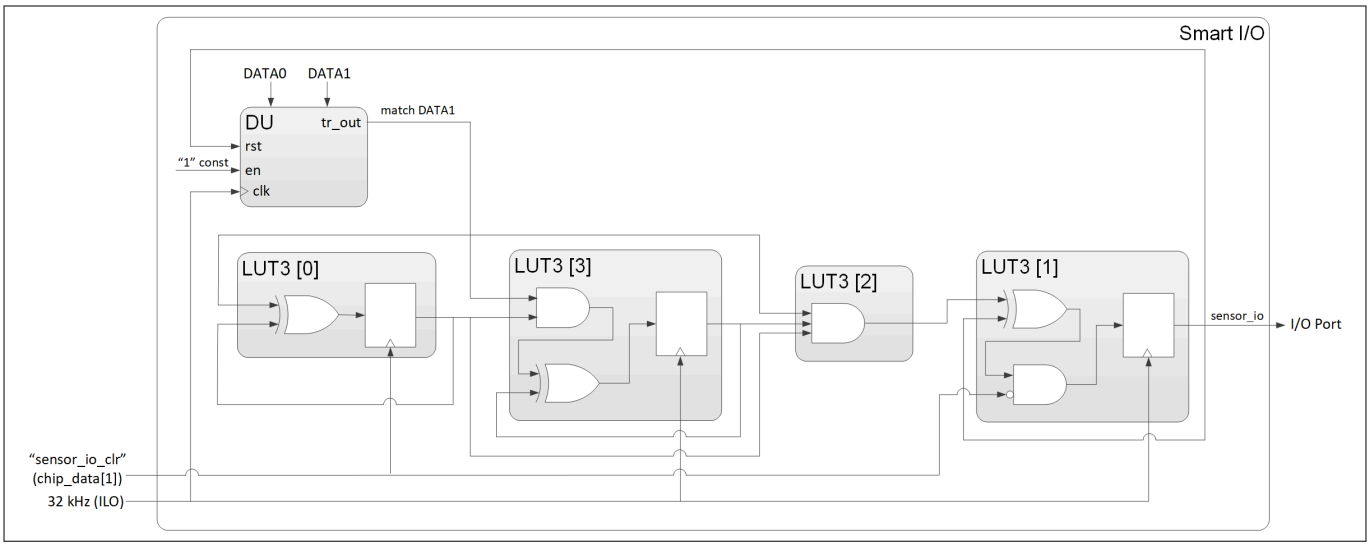

図 13　　センサ起動回路の論理例

データユニットは INCR_WRAP モードで動作します。このモードでは、データが初期値 (DATA 0) から最終値 (DATA 1) に達するまで 1 ずつ増加します。カウント値が最終値と一致すると、DATA 0 にラップアラウンドします。

この回路では、データユニットは 11 ビットカウンタの下位 8 ビットを伝送します。LUT3[0], LUT3 [1], LUT3 [3]は、11 ビットカウンタの 9, 10, および 11 ビットを表します。LUT3 [1]の出力、つまりカウンタの 11 番目のビットは、外部センサをアクティブにするために GPIO ポートに接続されます。図 14 に、この使用例の信号パスを示します。

3 消費電力モードの遷移

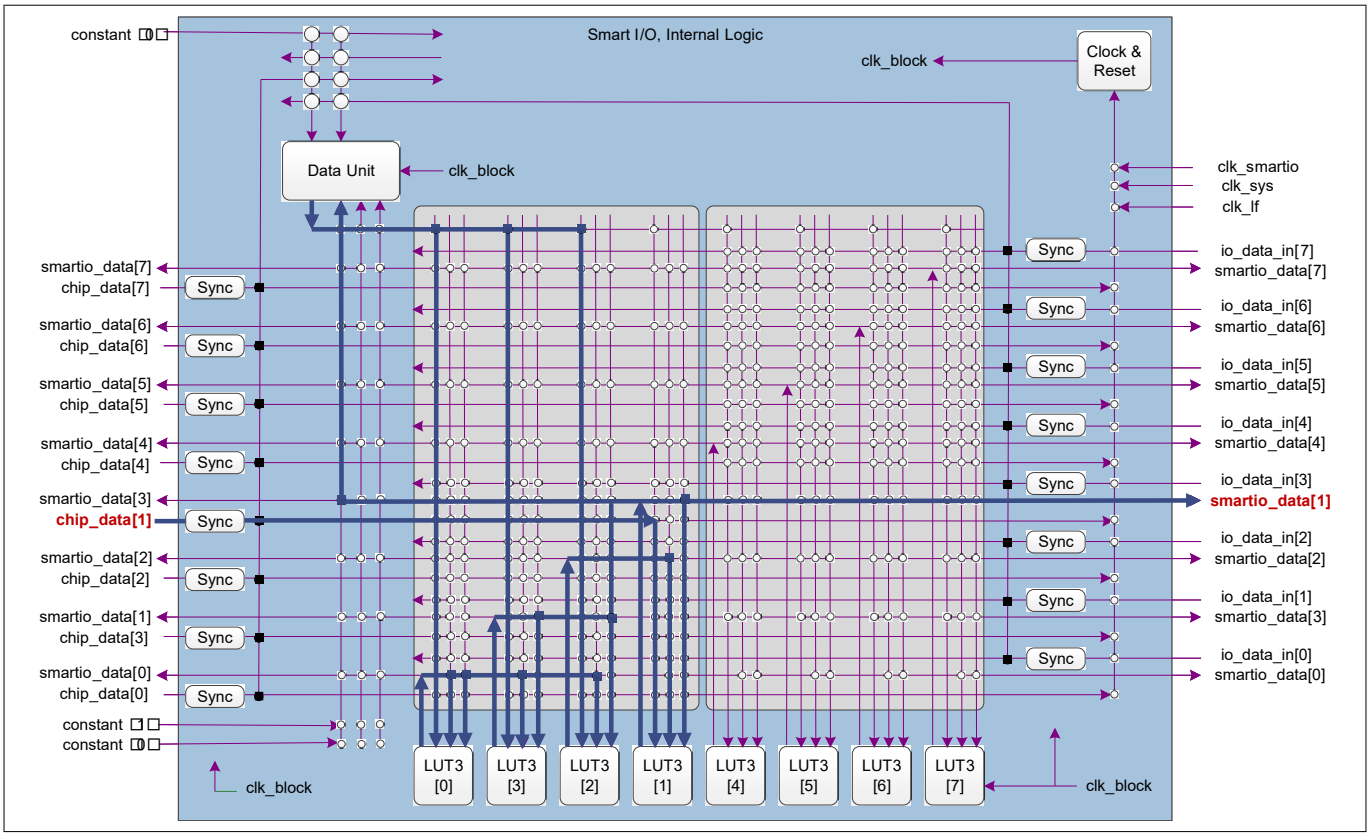

図 14　　センサ起動回路の信号経路

表 9, 表 10, 表 11, および表 12 に、各 LUT3 の真理値表を示します。表の太字は、無効なパターンを示しています。

表 9　　Lookup table LUT3 [0]

Tr2_in	Tr1_in	Tr0_in	Tr_out
LUT3[0] out	**LUT3[0] out**	**DU tr_out**	
0	0	0	0
0	0	1	1
0	**1**	**0**	**0**
0	**1**	**1**	**0**
1	**0**	**0**	**0**
1	**0**	**1**	**0**
1	1	0	1
1	1	1	0

表 10　　Lookup table LUT3 [1]

Tr2_in	Tr1_in	Tr0_in	Tr_out
LUT3[1] out	**LUT3[0] out**	**DU tr_out**	
0	0	0	0

(続く)

表 10 (続き) Lookup table LUT3 [1]

Tr2_in	Tr1_in	Tr0_in	Tr_out
LUT3[1] out	LUT3[0] out	DU tr_out	
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

表 11 Lookup table LUT3 [2]

Tr2_in	Tr1_in	Tr0_in	Tr_out
LUT3[1] out	LUT3[0] out	DU tr_out	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

表 12 Lookup table LUT3 [3]

Tr0_in	Tr1_in	Tr2_in	Tr_out
LUT3[3] out	LUT3[2] out	CHIP_ DATA[0]	
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

センサ起動回路によって遅延される H 出力は、次のように計算できます。

$$delay = 4 \times \frac{DAT1 - DAT0}{32} = \frac{DAT1 - DAT0}{8}[ms]$$

したがって、次のような大まかな見積もりのように、センサ安定化待機時間を満足するように DAT1 と DAT0 (通常は「0」に設定) を設定できます。

$$delay = T_{Cyclic\ Wakeup\ period} - T_{sensor\ stabilization\ wait}$$

$$\therefore DAT1 = 8 \times \left(T_{Cyclic\ Wakeup\ period} - T_{sensor\ stabilization\ wait}\right)$$

例えば、$T_{Cyclic\ Wakeup\ period}$ =32 [ms] の場合、DAT0 = 0, DAT1 = 0xFD に設定して以下のようにできます。

$$T_{sensor\ stabilization\ wait} \geq 300[\mu s]$$

3.3.4.3 Cyclic ウェイクアップ動作でのセンサ起動回路

前のセクションで構築したセンサ起動回路を使用すると、図 15 に示すように、Cyclic ウェイクアップの動作を強化できます。

3 消費電力モードの遷移

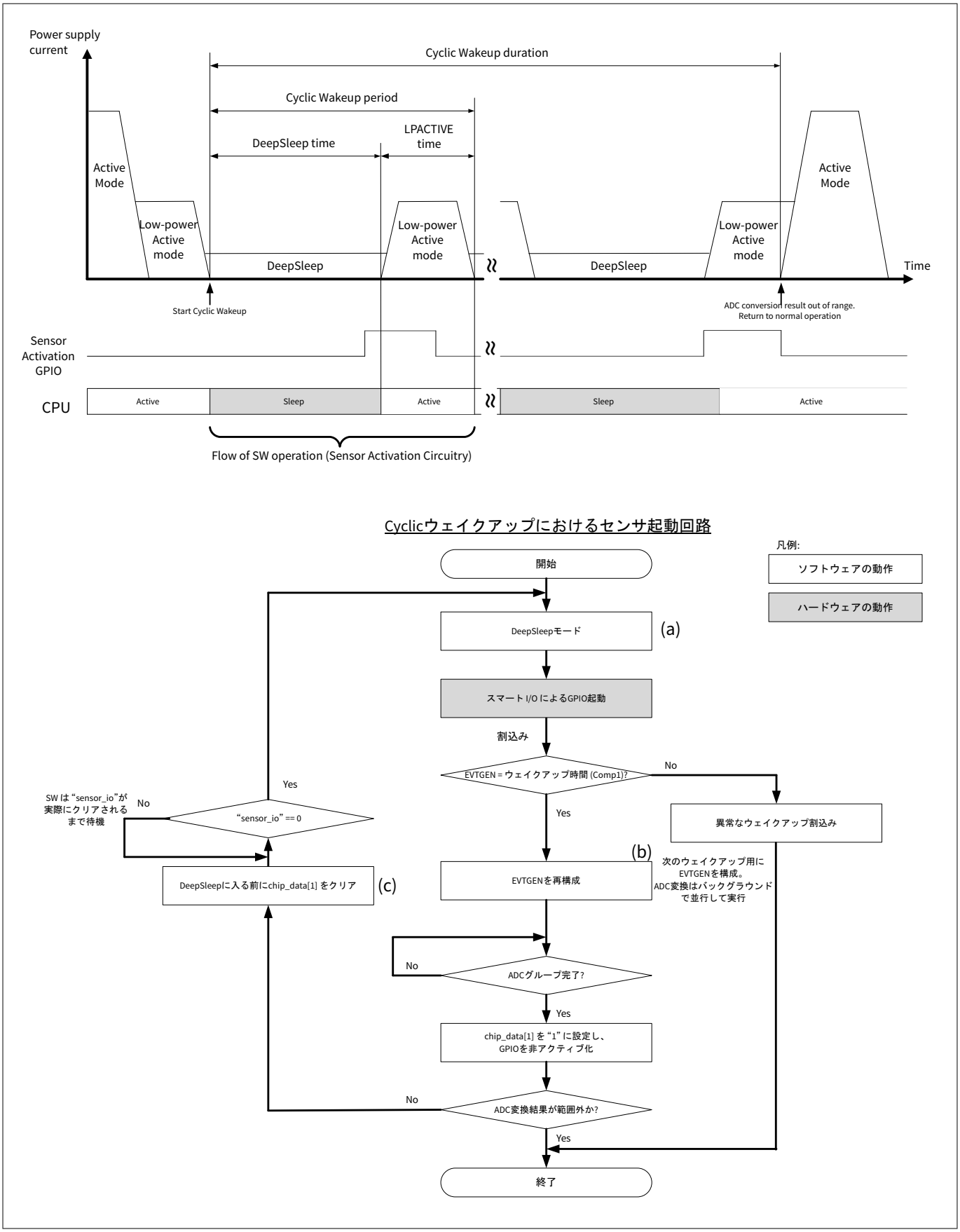

図 15　　スマート I/O を使用した Cyclic ウェイクアップ動作

注: フローチャートの灰色のボックスは、ハードウェア操作を示します。したがって、ソフトウェアによる処理は必要ありません。

GPIO は、CPU が DeepSleep モードのままで、スマート I/O によってアクティブ化され、DAT0 と DAT1 を適切に調整するだけで、センサの安定化時間を満足させることができます。GPIO の動作は、CPU の動作から分離されました。これにより、ソフトウェアフローが簡単になります。

例えば、長いセンサ待機時間が必要な場合、GPIO をアクティブ化した後、CPU がスリープモードに移行して消費電流を削減する必要はありません。

3.3.4.4 設定とサンプルコード

Cyclic ウェイクアップでのスマート I/O 用 SDL の設定部のパラメーターを表 13 に、関数を表 14 を示します。

表 13 Cyclic ウェイクアップでのスマート I/O 用設定パラメーター一覧

パラメーター	説明	値
.sysIntSrc	イベントジェネレータの割込み設定	evtgen_0_interrupt_dpslp_IRQn
.intIdx		CPUIntIdx4_IRQn
.isEnabled		true
.frequencyRef	clk_ref	8000000ul
.frequencyLf	clk_lf	32000ul
.frequencyTick	イベントジェネレータクロック (clk_ref_div)	32000ul
.ratioControlMode	イベントジェネレータ比制御モード	CY_EVTGEN_RATIO_CONTROL_SW
.ratioValueDynamicMode	イベントジェネレータダイナミックモード	CY_EVTGEN_RATIO_DYNAMIC_MODE0
.functionalitySelection	イベントジェネレータ選択機能	CY_EVTGEN_DEEPSLEEP_FUNCTIONALITY
.triggerOutEdge	イベントジェネレータトリガ	CY_EVTGEN_EDGE_SENSITIVE
.valueDeepSleepComparator	コンパレータ構造を初期化	DPSLP_COMP_VAL
.valueActiveComparator	コンパレータ構造を初期化	ACTIVE_COMP_VAL
lutCfgLut0.opcode	LUT3[0] 動作モード設定を設定	CY_SMARTIO_LUTOPC_GATED_OUT
lutCfgLut0.lutMap	LUT3[0] 出力パターン設定を設定	0x42ul
lutCfgLut0.tr0	LUT3[0] tr0 入力を設定	CY_SMARTIO_LUTTR_DU_OUT
lutCfgLut0.tr1	LUT3[0] tr1 入力を設定	CY_SMARTIO_LUTTR_LUT0_OUT
lutCfgLut0.tr2	LUT3[0] tr2 入力を設定	CY_SMARTIO_LUTTR_LUT0_OUT
lutCfgLut3.opcode	LUT3[3] 動作モード設定を設定	CY_SMARTIO_LUTOPC_GATED_OUT
lutCfgLut3.lutMap	LUT3[3] 出力パターン設定を設定	0x78ul
lutCfgLut3.tr0	LUT3[3] tr0 入力を設定	CY_SMARTIO_LUTTR_DU_OUT
lutCfgLut3.tr1	LUT3[3] tr1 入力を設定	CY_SMARTIO_LUTTR_LUT0_OUT

(続く)

表 13 (続き) Cyclic ウェイクアップでのスマート I/O 用設定パラメーター一覧

パラメーター	説明	値
lutCfgLut3.tr2	LUT3[3] tr2 入力を設定	CY_SMARTIO_LUTTR_LUT3_OUT
lutCfgLut2.opcode	LUT3[2] 動作モード設定を設定	CY_SMARTIO_LUTOPC_COMB
lutCfgLut2.lutMap	LUT3[2] 出力パターン設定を設定	0x80ul
lutCfgLut2.tr0	LUT3[2] tr0 入力を設定	CY_SMARTIO_LUTTR_DU_OUT
lutCfgLut2.tr1	LUT3[2] tr1 入力を設定	CY_SMARTIO_LUTTR_LUT0_OUT
lutCfgLut2.tr2	LUT3[2] tr2 入力を設定	CY_SMARTIO_LUTTR_LUT3_OUT
lutCfgLut1.opcode	LUT3[1] 動作モード設定を設定	CY_SMARTIO_LUTOPC_GATED_OUT
lutCfgLut1.lutMap	LUT3[1] 出力パターン設定を設定	0x54ul
lutCfgLut1.tr0	LUT3[1] tr0 入力を設定	CY_SMARTIO_LUTTR_CHIP1
lutCfgLut1.tr1	LUT3[1] tr1 入力を設定	CY_SMARTIO_LUTTR_LUT1_OUT
lutCfgLut1.tr2	LUT3[1] tr2 入力を設定	CY_SMARTIO_LUTTR_LUT2_OUT
lutCfgDu.tr0	DU 入力トリガ 0 ソース選択を設定	CY_SMARTIO_DUTR_LUT1_OUT
lutCfgDu.tr1	DU 入力トリガ 1 ソース選択を設定	CY_SMARTIO_DUTR_ONE
lutCfgDu.data0	DU 入力 DATA0 ソース選択	CY_SMARTIO_DUDATA_ZERO
lutCfgDu.data1	DU 入力 DATA1 ソース選択	CY_SMARTIO_DUDATA_DATAREG
lutCfgDu.opcode	DU オペコード	CY_SMARTIO_DUOPC_INCR_WRAP
lutCfgDu.size	DU 幅サイズは 8	CY_SMARTIO_DUSIZE_8
lutCfgDu.dataReg	DU DATA レジスタ値	0xFCul

表 14 Cyclic ウェイクアップでのスマート I/O 用設定関数一覧

関数	説明	備考
CyclicWakeUp_SystemUpdate()	Cyclic ウェイクアップのシステム更新	Code Listing 21 を参照してください。
CyclicWakeUp_Operation()	サイクリックウェイクアップ機能	Code Listing 22 を参照してください。
Init_SmartIO()	スマート I/O モジュールを初期化	Code Listing 23 を参照してください。
Cy_SmartIO_Enable()	スマート I/O を有効	Code Listing 24 を参照してください。
Init_SmartIO_Cfg()	スマート I/O を設定	Code Listing 25 を参照してください。

(続く)

表 14 (続き) Cyclic ウェイクアップでのスマート I/O 用設定関数一覧

関数	説明	備考
Cy_SmartIO_Deinit()	スマート I/O をデフォルト値にリセット	Code Listing 26 を参照してください。
Cy_GPIO_Inv()	ピン出力ロジック状態を現在の出力ロジック状態の逆に設定	Code Listing 27 を参照してください。

Code Listing 19 に、Cyclic ウェイクアップ動作でスマート I/O を実行するためのサンプルプログラムを示します。GPIO, ADC, およびスマート I/O の Architecture reference manual とアプリケーションノートを参照してください。

Code Listing 19 Cyclic ウェイクアップ動作でのスマート I/O の使用例

```
int main(void)
{
:
  CyclicWakeUp_SystemUpdate();  /* SystemUpdate for Cyclic wakeup See Code Listing 21 */

  Init_SmartIO();   /* Smart IO module initialization See Code Listing 23 */

  Cy_SmartIO_Enable(SMART_IO_PORT);   /* Configures Smart I/O See Code Listing 24 */

  while(1 /*g_flagContinueCWK*/){
    Cy_GPIO_Inv(DPSLP_IDC_PRT, DPSLP_IDC_PIN);   /* See Code Listing 27 */
    CyclicWakeUp_Operation();   /* Cyclic WakeUp function. See (a) of 図 15. See Code Listing
22 */
  }


  for(;;)
  {
    Cy_GPIO_Inv(DPSLP_IDC_PRT, DPSLP_IDC_PIN);
    for(uint32_t idx = 0ul; idx < 1000000ul; idx++ ){}
  }

}
```

3 消費電力モードの遷移

Code Listing 20 イベントジェネレータの設定

```
/**
 * \var cy_stc_evtgen_config_t evtgenConfig
 * \brief Evtgen configuration
 */
/* Eventgenerator Configuration */
const cy_stc_evtgen_config_t evtgenConfig =
{

    .frequencyRef          = 8000000ul,   /**< clk_ref = clk_hf1 = CLK_PATH2 (IMO) -> 8,000,000
for silicon */
    .frequencyLf           = 32000ul,     /**< clk_lf = 32,000 for silicon */

    .frequencyTick         = 32000ul,     /**< Setting 1,000,000 Hz for event generator clock
(clk_ref_div) */
    .ratioControlMode      = CY_EVTGEN_RATIO_CONTROL_SW,
    .ratioValueDynamicMode = CY_EVTGEN_RATIO_DYNAMIC_MODE0,
};

/**
 * \var cy_stc_evtgen_struct_config_t evtgenStructureConfig
 * \brief Evtgen structure configuration
 */
const cy_stc_evtgen_struct_config_t evtgenStructureConfig =
{
    .functionalitySelection   = CY_EVTGEN_DEEPSLEEP_FUNCTIONALITY,
    .triggerOutEdge           = CY_EVTGEN_EDGE_SENSITIVE,
    .valueDeepSleepComparator = DPSLP_COMP_VAL, /**< In active functionality, this value is
used for making period of interrupts/triggers */
                                        /**< 32,000 / 1,000,000 (clk_ref_div) = 32[ms] */
    .valueActiveComparator    = ACTIVE_COMP_VAL, /**< In active functionality, this value is
used for making period of interrupts/triggers */
                                        /**< 40,000 / 1,000,000 (clk_ref_div) = 4[ms] */
};
```

3 消費電力モードの遷移

Code Listing 21 CyclicWakeUp_SystemUpdate() 関数

```
/* SystemUpdate for Cyclic wakeup */
void CyclicWakeUp_SystemUpdate(void)
{

    SRSS->unPWR_CTL2.stcField.u1LINREG_DIS = 0ul;
    SRSS->unPWR_CTL2.stcField.u1BGREF_LPMODE = 1ul;

    /*********************/
    /*  Clock Setthings  */
    /*********************/
    /***     FLL  disabling        ***/
    /* Disable Fll */
    SRSS->unCLK_FLL_CONFIG.stcField.u1FLL_ENABLE = 0ul; /* 0 = disable */
    SRSS->unCLK_FLL_CONFIG4.stcField.u1CCO_ENABLE = 0ul; /* 0 = disable */

    /*********** Setting wait state for ROM **********/
    CPUSS->unROM_CTL.stcField.u2SLOW_WS = 0ul;
    CPUSS->unROM_CTL.stcField.u2FAST_WS = 0ul;

    /*********** Setting wait state for RAM **********/
    CPUSS->unRAM0_CTL0.stcField.u2SLOW_WS = 0ul;
    CPUSS->unRAM0_CTL0.stcField.u2FAST_WS = 0ul;

    #if defined (CPUSS_RAMC1_PRESENT) && (CPUSS_RAMC1_PRESENT == 1UL)
    CPUSS->unRAM1_CTL0.stcField.u2SLOW_WS = 0ul;
    CPUSS->unRAM1_CTL0.stcField.u2FAST_WS = 0ul;
    #endif /* defined (CPUSS_RAMC1_PRESENT) && (CPUSS_RAMC1_PRESENT == 1UL) */

    #if defined (CPUSS_RAMC2_PRESENT) && (CPUSS_RAMC2_PRESENT == 1UL)
    CPUSS->unRAM2_CTL0.stcField.u2SLOW_WS = 0ul;
    CPUSS->unRAM2_CTL0.stcField.u2FAST_WS = 0ul;
    #endif /* defined (CPUSS_RAMC2_PRESENT) && (CPUSS_RAMC2_PRESENT == 1UL) */

    /*********** Setting wait state for FLASH **********/
    FLASHC->unFLASH_CTL.stcField.u4MAIN_WS = 0ul;

    /***    Set clock LF source        ***/
    SRSS->unCLK_SELECT.stcField.u3LFCLK_SEL = CY_SYSCLK_LFCLK_IN_ILO0;

    /*****************************************/
    /*        Deinitialize peripherals       */
    /*****************************************/
    Cy_Evtgen_DeinitializeCompStruct(EVTGEN0, EVTGEN_COMP_STRUCT_NO);
    Cy_Evtgen_Deinitialize(EVTGEN0);

    /*****************************************/
    /*  Initialize and start Event generator  */
    /*****************************************/
    Cy_Evtgen_Initialize(EVTGEN0,&evtgenConfig);

    /*****************************************/
```

3 消費電力モードの遷移

```
    /*  Initialize comparator structure       */
    /*****************************************/
    Cy_Evtgen_InitializeCompStruct(EVTGEN0, EVTGEN_COMP_STRUCT_NO, &evtgenStructureConfig,
&evtgenStruct0Context);
}
```

3 消費電力モードの遷移

Code Listing 22 CyclicWakeUp_Operation() 関数

```
/* Cyclic WakeUp function */
void CyclicWakeUp_Operation(void)
{
    /* confirm that output of LUT1 has been cleared */
    while((LUT1_OUT_LED_PORT->unIN.u32Register >> (LUT1_OUT_LED_PIN)) & CY_GPIO_IN_MASK){};

    /* clear chip_data_out[1] before entering deepsleep*/
    LUT1_OUT_LED_PORT->unOUT_CLR.u32Register = CY_GPIO_OUT_MASK << LUT1_OUT_LED_PIN;    /* See
(c) of 図 15. */

    /* Put the system to DeepSleep */
    {
        /* put mcu in deepsleep */
        SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
//        SCB->SCR;  /* dummy read +/
        WaitCoreCycle(1ul);
        __WFI();   /* See (a) of 図 15. Set to the DeepSleep mode */

        /* Start ADC by software trigger*/
        Cy_Adc_Channel_SoftwareTrigger(&CYCLIC_ADC_POT_MACRO-
>CH[ADC_GROUP_FIRST_LOGICAL_CHANNEL]);

        /* Clear evtgen deepsleep interrput */
        Cy_Evtgen_ClearStructInterruptDeepSleep(EVTGEN0,EVTGEN_COMP_STRUCT_NO);
        NVIC_ClearPendingIRQ(irq_cfg_evtgen_dpslp.intIdx);
    }

    /* Reconfigure the event generator for the next wake-up */
    /* See (b) of 図 15. */
    {
        g_EvtgenCompareValue = (uint32_t)(g_EvtgenCompareValue + DPSLP_COMP_VAL);

        /* Disable deep sleep comparator */
        EVTGEN0->COMP_STRUCT[EVTGEN_COMP_STRUCT_NO].unCOMP_CTL.stcField.u1COMP1_EN = 0ul;

        /* Setting deep sleep comparator value */
        EVTGEN0->COMP_STRUCT[EVTGEN_COMP_STRUCT_NO].unCOMP1.stcField.u32INT32 =
g_EvtgenCompareValue;

        /* Enable deep sleep comparator */
        EVTGEN0->COMP_STRUCT[EVTGEN_COMP_STRUCT_NO].unCOMP_CTL.stcField.u1COMP1_EN = 1ul;
    }

    /* wait for ADC completion */
    while(!CYCLIC_ADC_POT_MACRO->CH[ADC_GROUP_LAST_LOGICAL_CHANNEL].unINTR.stcField.u1GRP_DONE)
{};

    /* set chip_data_out[1] to clear lut1_trout */
    LUT1_OUT_LED_PORT->unOUT_SET.u32Register = CY_GPIO_OUT_MASK << LUT1_OUT_LED_PIN;

    /* In this sample software, only check range comparison result for first ADC channel  */
```

```
    if(CYCLIC_ADC_POT_MACRO->CH[ADC_GROUP_FIRST_LOGICAL_CHANNEL].unINTR.stcField.u1CH_RANGE ==
1ul) {
        g_flagContinueCWK = false;
    }

    for (uint8_t ch = ADC_GROUP_FIRST_LOGICAL_CHANNEL; ch < (ADC_GROUP_FIRST_LOGICAL_CHANNEL +
ADC_GROUP_NUMBER_OF_CHANNELS); ch++)
    {
        /* Clear interrupt source */
        CYCLIC_ADC_POT_MACRO->CH[ch].unINTR.u32Register = 0xFFFFFFFFul;
    }
}
```

Code Listing 23 Init_SmartIO() 関数

```
/* Smart IO module initialization */
void Init_SmartIO(void)
{
Cy_SmartIO_Deinit(SMART_IO_PORT);    /* See Code Listing 26 */
            Init_SmartIO_Cfg();     /* See Code Listing 25 */
}
```

Code Listing 24 Cy_SmartIO_Enable() 関数

```
/* Enable Smart I/O */
void Cy_SmartIO_Enable(volatile stc_SMARTIO_PRT_t* base)
{
    un_SMARTIO_PRT_CTL_t workCTL = base->unCTL;
    workCTL.stcField.u1ENABLED     = CY_SMARTIO_ENABLE;
    workCTL.stcField.u1PIPELINE_EN = CY_SMARTIO_DISABLE;
    base->unCTL.u32Register        = workCTL.u32Register;
}
```

3 消費電力モードの遷移

Code Listing 25 Init_SmartIO_Cfg() 関数

```
cy_en_smartio_status_t Init_SmartIO_Cfg(void)
{
    /* Configures Smart I/O */
    /* Configure smart io to output H in deepsleep
     * Using data unit and LUT0,1,2,3 to create a 11bit counter
     * Data uint acts as lower 8 bit, count up from 0 to value written in DATA, reset to 0 at
overflow
     * LUT0 acts as 9th bit, LUT3 acts as 10th bit and LUT 1 act as 11th bit
     * output of LUT1 is smart io output, i.e. P15.1 or TP202 on the CPU board
    */

    cy_stc_smartio_ducfg_t lutCfgDu;
    cy_stc_smartio_lutcfg_t lutCfgLut0;
    cy_stc_smartio_lutcfg_t lutCfgLut1;
    cy_stc_smartio_lutcfg_t lutCfgLut2;
    cy_stc_smartio_lutcfg_t lutCfgLut3;

    cy_stc_smartio_config_t smart_io_cfg;
    cy_en_smartio_status_t retStatus = (cy_en_smartio_status_t)0xFF;

    /* initialize the Smart IO structure */
    memset(&lutCfgDu, 0, sizeof(cy_stc_smartio_ducfg_t));
    memset(&lutCfgLut0, 0, sizeof(cy_stc_smartio_lutcfg_t));
    memset(&lutCfgLut1, 0, sizeof(cy_stc_smartio_lutcfg_t));
    memset(&lutCfgLut2, 0, sizeof(cy_stc_smartio_lutcfg_t));
    memset(&lutCfgLut3, 0, sizeof(cy_stc_smartio_lutcfg_t));
    memset(&smart_io_cfg, 0, sizeof(cy_stc_smartio_config_t));

    /* Active clock source is selected */
    smart_io_cfg.clkSrc = (cy_en_smartio_clksrc_t)CY_SMARTIO_CLK_LFCLK;

    /* Bypass channel mask is 11111100 for Pin0 and Pin1 */
    smart_io_cfg.bypassMask = SMARTIO_BYPASS_CH_MASK;

    smart_io_cfg.hldOvr = true;

    /************************** LUT0 config **************************/
    /* Configure LUT3 [0] */
    lutCfgLut0.opcode = CY_SMARTIO_LUTOPC_GATED_OUT;

    lutCfgLut0.lutMap = 0x42ul;

    lutCfgLut0.tr0 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_DU_OUT;
    lutCfgLut0.tr1 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT0_OUT;
    lutCfgLut0.tr2 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT0_OUT;
    smart_io_cfg.lutCfg[0] = &lutCfgLut0;

    /************************** LUT3 config **************************/
    /* Configure LUT3 [3] */
    lutCfgLut3.opcode = CY_SMARTIO_LUTOPC_GATED_OUT;
```

3 消費電力モードの遷移

```
    lutCfgLut3.lutMap = 0x78ul;

    lutCfgLut3.tr0 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_DU_OUT;
    lutCfgLut3.tr1 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT0_OUT;
    lutCfgLut3.tr2 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT3_OUT;
    smart_io_cfg.lutCfg[3] = &lutCfgLut3;

    /************************** Lut2 config **************************/
    /* Configure LUT3 [2] */
    lutCfgLut2.opcode =  CY_SMARTIO_LUTOPC_COMB;

    lutCfgLut2.lutMap = 0x80ul;

    lutCfgLut2.tr0 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_DU_OUT;
    lutCfgLut2.tr1 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT0_OUT;
    lutCfgLut2.tr2 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT3_OUT;
    smart_io_cfg.lutCfg[2] = &lutCfgLut2;

    /************************** LUT1 config **************************/
    /* Configure LUT3 [1] */
    lutCfgLut1.opcode = CY_SMARTIO_LUTOPC_GATED_OUT;

    lutCfgLut1.lutMap = 0x54ul;

    lutCfgLut1.tr0 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_CHIP1;
    lutCfgLut1.tr1 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT1_OUT;
    lutCfgLut1.tr2 = (cy_en_smartio_luttr_t)CY_SMARTIO_LUTTR_LUT2_OUT;
    smart_io_cfg.lutCfg[1] = &lutCfgLut1;

    /* Data Unit (DU) configuration structure */
    /* Configure DU */
    lutCfgDu.tr0 = CY_SMARTIO_DUTR_LUT1_OUT, /*CY_SMARTIO_DUTR_DU_OUT;*/       /**< DU input
trigger 0 source selection */
    lutCfgDu.tr1 = CY_SMARTIO_DUTR_ONE;       /**< DU input trigger 1 source selection */
    lutCfgDu.data0 = CY_SMARTIO_DUDATA_ZERO;   /**< DU input DATA0 source selection */
    lutCfgDu.data1 = CY_SMARTIO_DUDATA_DATAREG;   /**< DU input DATA1 source selection */
    lutCfgDu.opcode = CY_SMARTIO_DUOPC_INCR_WRAP;   /**< DU op-code */
    lutCfgDu.size = CY_SMARTIO_DUSIZE_8;    /**< DU operation bit size */
    lutCfgDu.dataReg = 0xFCul;               /**< DU DATA register value */
    smart_io_cfg.duCfg = &lutCfgDu;

}
```

Code Listing 26 Cy_SmartIO_Deinit() 関数

```
void Cy_SmartIO_Deinit(volatile stc_SMARTIO_PRT_t* base)
{
    un_SMARTIO_PRT_CTL_t workCTL= {.u32Register = 0ul};
    workCTL.stcField.u1ENABLED     = CY_SMARTIO_DISABLE;   /* Resets the Smart I/O to default
values */
    workCTL.stcField.u1PIPELINE_EN = CY_SMARTIO_ENABLE;
    workCTL.stcField.u5CLOCK_SRC   = CY_SMARTIO_CLK_GATED;
    workCTL.stcField.u8BYPASS      = CY_SMARTIO_CHANNEL_ALL;
    base->unCTL.u32Register        = workCTL.u32Register;

    base->unSYNC_CTL.u32Register = CY_SMARTIO_DEINIT;
    for(uint8_t idx = CY_SMARTIO_LUTMIN; idx < CY_SMARTIO_LUTMAX; idx++)
    {
        base->unLUT_SEL[idx].u32Register = CY_SMARTIO_DEINIT;
        base->unLUT_CTL[idx].u32Register = CY_SMARTIO_DEINIT;
    }
    base->unDU_SEL.u32Register = CY_SMARTIO_DEINIT;
    base->unDU_CTL.u32Register = CY_SMARTIO_DEINIT;
    base->unDATA.u32Register = CY_SMARTIO_DEINIT;
}
```

Code Listing 27 Cy_GPIO_Inv() 機能

```
__STATIC_INLINE void Cy_GPIO_Inv(volatile stc_GPIO_PRT_t* base, uint32_t pinNum)
{
base->unOUT_INV.u32Register = CY_GPIO_OUT_MASK << pinNum;    /*Set a pin output logic state to
the inverse of the current output logic state */
}
```

3.4 CAN ウェイクアップ動作

CAN バス上で通信がある限り、ECU は起動しています。ECU が低消費電力モードのときに CAN 通信が発生した場合、ECU は低消費電力モードから起動する必要があります。ここでは、TRAVEO™ T2G ファミリ MCU を使用した CAN ウェイクアップ動作の実装例を説明します。

TRAVEO™ T2G ファミリ MCU には、以下のウェイクアップ用ピンがあります。

- Hibernate モードからウェイクアップするための最大 10 ピン

Hibernate モードからデバイスをウェイクアップできるピンのサポート数については、データシートを参照してください。

- DeepSleep モードからウェイクアップするための GPIO ピン

DeepSleep モードからデバイスをウェイクアップできる GPIO のサポート数については、データシートを参照してください。

MCU が DeepSleep または Hibernate モードの場合、CAN ブロックはウェイクアップ状態を検出できません。CAN ウェイクアップをサポートするには、MCU は GPIO 割込みまたは WAKEUP ピンの機能を使用する必要があります。DeepSleep に入る前に CAN_RX の HSIOM コンフィギュレーションを GPIO に変更し、ウェイクアップ後に同じピンを CAN_RX に再設定することは、必須ではありません。ただし、GPIO の割り込みコンフィギュレーションは必須です。

図 16 に DeepSleep モードからの CAN ウェイクアップ動作を示します。

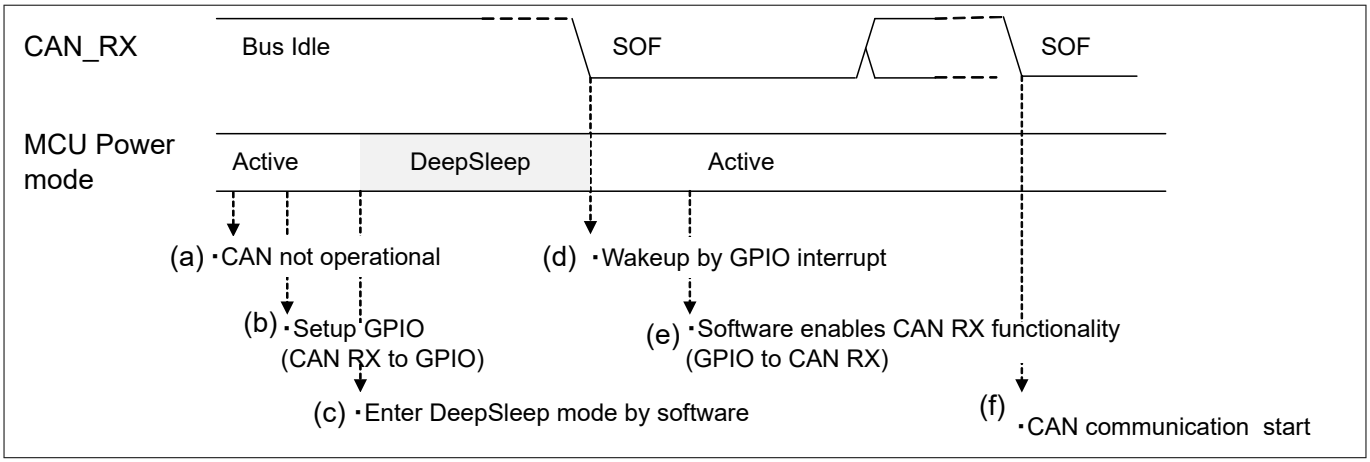

図 16　　DeepSleep モードからの CAN ウェイクアップ動作

MCU が低消費電力モードに入る前に、ソフトウェアは I/O ポートと GPIO 割込みを設定します。CAN の受信機能は無効です。その後、低消費電力モードで MCU がウェイクアップ信号を検出すると、MCU は低消費電力モードからウェイクアップします。その後、ソフトウェアが CAN の受信機能を有効にし、CAN 通信が開始されます。CAN および GPIO の詳細については Architecture reference manual を参照してください。

3.4.1　　設定とサンプルコード

表 15 に、CAN ウェイクアップ動作の SDL の設定部の関数を示します。この例では、GPIO ピンと CAN RX ピンが共通です。

表 15　　CAN ウェイクアップ動作の SDL の設定部の関数一覧

関数	説明	備考
GpioIntHandler()	GPIO 割込みのハンドラ	Code Listing 30 を参照してください。
Cy_GPIO_Pin_Init()	ピンのすべてのピン設定を初期化	Code Listing 30 を参照してください。
Cy_SysPm_DeepSleep()	システムを DeepSleep にします	Code Listing 31 を参照してください。

Code Listing 28 に、CAN ウェイクアップ動作のサンプルプログラムを示します。CAN の Architecture reference manual およびアプリケーションノートを参照してください。

3 消費電力モードの遷移

Code Listing 28 CAN ウェイクアップ動作の例

```
int main(void)
{
/* Wakeup by GPIO interrupt. See (d) of 図 16. See Code Listing 29 */

    Cy_SysInt_SetSystemIrqVector(gpio_irq_cfg.sysIntSrc, GpioIntHandler);

    for(;;)
    {

            /* Stop CAN */
            /* CAN Clock stop request */
            /* Stop CANFD. See (a) of 図 16. */
            CY_CANFD_UNIT->unCTL.stcField.u8STOP_REQ = CY_CANFD_STOP_REQ_BIT;
            while(CY_CANFD_UNIT->unSTATUS.stcField.u8STOP_ACK != CY_CANFD_STOP_REQ_BIT);


            /* Change CAN Rx Port to GPIO */
            /* Change CAN RX to GPIO. See (b) of 図 16.  See Code Listing 30 */
            Cy_GPIO_Pin_Init(CY_CANFD0_RX_PORT, CY_CANFD0_RX_PIN, &user_button_port_pin_cfg);

            /* Put the system to DeepSleep */
            /* Enter DeepSleep mode. See (c) of 図 16. See Code Listing 31 */
            Cy_SysPm_DeepSleep(CY_SYSPM_WAIT_FOR_INTERRUPT);

            /* Change GPIO to CAN Rx Port */
            /* Change GPIO to CAN RX. See (e) of 図 16. See Code Listing 30 */
            Cy_GPIO_Pin_Init(can_pin_cfg[0].portReg, can_pin_cfg[0].pinNum,
&can_pin_cfg[0].cfg);

            /* CAN Clock start request */
            CY_CANFD_UNIT->unCTL.stcField.u8STOP_REQ = 0x00ul;
            while(CY_CANFD_UNIT->unSTATUS.stcField.u8STOP_ACK != 0x00ul);

            /* Start CAN */
            CY_CANFD_TYPE->M_TTCAN.unCCCR.stcField.u1INIT = 0ul;

            /* Start CANFD. See (f) of 図 16 */
            while(CY_CANFD_TYPE->M_TTCAN.unCCCR.stcField.u1INIT != 0ul);
        }
    }
}
```

3 消費電力モードの遷移

Code Listing 29 GpioIntHandler() 関数

```
/* Handler for GPIO interrupts */
void GpioIntHandler(void)
{
    uint32_t intStatus;

    /* If falling edge detected */
    intStatus = Cy_GPIO_GetInterruptStatusMasked(CY_CANFD0_RX_PORT, CY_CANFD0_RX_PIN);
    if (intStatus != 0ul)
    {
        Cy_GPIO_ClearInterrupt(CY_CANFD0_RX_PORT, CY_CANFD0_RX_PIN);
    }
}
```

Code Listing 30 Cy_GPIO_Init() 関数

```
cy_en_gpio_status_t Cy_GPIO_Pin_Init(volatile stc_GPIO_PRT_t *base, uint32_t pinNum, const
cy_stc_gpio_pin_config_t *config)
{
    /* Initialize all pin configuration setting for the pin */
    cy_en_gpio_status_t status = CY_GPIO_SUCCESS;

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

    return(status);
}
```

3 消費電力モードの遷移

Code Listing 31 Cy_SysPm_DeepSleep() 関数

```
/* Sets a CPU core to the DeepSleep mode */
cy_en_syspm_status_t Cy_SysPm_DeepSleep(cy_en_syspm_waitfor_t waitFor)
{
    uint32_t interruptState;
    cy_en_syspm_status_t retVal = CY_SYSPM_SUCCESS;

    /* Call the registered callback functions with
    * the CY_SYSPM_CHECK_READY parameter.
    */
    if(0u != currentRegisteredCallbacksNumber)
    {
        retVal = Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_CHECK_READY);
    }

    /* The device (core) can switch into the deep sleep power mode only when
    *  all executed registered callback functions with the CY_SYSPM_CHECK_READY
    *  parameter returned CY_SYSPM_SUCCESS.
    */
    if(retVal == CY_SYSPM_SUCCESS)
    {
        /* Call the registered callback functions with the CY_SYSPM_BEFORE_TRANSITION
        * parameter. The return value is ignored.
        */
        interruptState = Cy_SysLib_EnterCriticalSection();
        if(0u != currentRegisteredCallbacksNumber)
        {
            (void) Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_BEFORE_ENTER);
        }

    #if(0u != CY_CPU_CORTEX_M0P)

        /* The CPU enters the deep sleep mode upon execution of WFI/WFE */
        SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;

        if(waitFor != CY_SYSPM_WAIT_FOR_EVENT)
        {
            __WFI();
        }
        else
        {
            __WFE();
        }
    #else

        /* Repeat WFI/WFE instructions if wake up was not intended.
        *  Cypress Ticket #272909
        */
        do
        {
            SCB->SCR |= SCB_SCR_SLEEPDEEP_Msk;
```

```
            if(waitFor != CY_SYSPM_WAIT_FOR_EVENT)
            {
                __WFI();
            }
            else
            {
                __WFE();
            }
        } while (0); //rmkn _FLD2VAL(CPUSS_CM4_PWR_CTL_PWR_MODE, CPUSS-
>unCM4_PWR_CTL.u32Register) == CY_SYSPM_CM4_PWR_CTL_PWR_MODE_RETAINED);

    #endif /* (0u != CY_CPU_CORTEX_M0P) */

        Cy_SysLib_ExitCriticalSection(interruptState);

        /* Call the registered callback functions with the CY_SYSPM_AFTER_TRANSITION
        *  parameter. The return value is ignored.
        */
        if(0u != currentRegisteredCallbacksNumber)
        {
            (void) Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_AFTER_EXIT);
        }
    }
    else
    {
        /* Execute callback functions with the CY_SYSPM_CHECK_FAIL parameter to
        * undo everything done in the callback with the CY_SYSPM_CHECK_READY
        * parameter. The return value is ignored.
        */
        (void) Cy_SysPm_ExecuteCallback(CY_SYSPM_DEEPSLEEP, CY_SYSPM_CHECK_FAIL);
        retVal = CY_SYSPM_FAIL;
    }
    return retVal;
}
```

用語集

表 16　　用語集

用語	説明
ADC	アナログデジタル変換器 (Analog-to-digital converter の略)。詳細については、Architecture reference manual の SAR ADC 章を参照してください。
Basic WDT	Basic ウォッチドッグタイマ (Basic watchdog timer の略)。詳細については、Architecture reference manual の Watchdog Timer 章を参照してください。
BOD	ブラウンアウト検出回路 (Brown-out detection の略)。詳細については、Architecture reference manual の Power Supply and Monitoring 章を参照してください。
CPUSS	CPU サブシステム (CPU subsystem の略)。詳細については、Architecture reference manual の CPU Subsystem セクションを参照してください。
ECO	外部水晶発振回路 (External crystal oscillator の略)。詳細については、Architecture reference manual の Clocking System 章を参照してください。
EVTGEN	イベントジェネレータ (Event generator の略)。詳細については、Architecture reference manual の Event Generator 章を参照してください。
FLL	周波数同期回路 (Frequency-locked loop の略)。詳細については、Architecture reference manual の Clocking System 章を参照してください。
GPIO	汎用ポート (General-purpose input/output の略)。詳細は Architecture reference manual の I/O System 章を参照してください。
HF	高速クロック (High frequency clock の略)。詳細については、Architecture reference manual の Clocking System 章を参照してください。
ILO	内部低速発振器 (Internal low-speed oscillators の略)。詳細については、Architecture reference manual の Clocking System 章を参照してください。
IMO	内部メイン発振器。詳細については、Architecture reference manual の Clocking System 章を参照してください。
LF	低速クロック (Low frequency clock の略)。詳細については、Architecture reference manual の Clocking System 章を参照してください。
LPACTIVE	低消費電力アクティブ (Low-power active の略)。詳細については、Architecture reference manual の Device power modes 章を参照してください。
LV supplies	低電圧 (Low-voltage) 電源。
LVD	低電圧検出回路 (Low-voltage detection の略)。詳細については、Architecture reference manual の Power Supply and Monitoring 章を参照してください。
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)。詳細については、Architecture reference manual の Watchdog Timer 章を参照してください。
Pending interrupt (保留中の割込み)	割込み要因が発生し、処理が待たされている状態の割込み。詳細については、Architecture reference manual の Interrupts 章を参照してください。
PLL	位相同期回路 (Phase-locked loop の略)。詳細については、Architecture reference manual の Clocking System 章を参照してください。
POR	パワーオンリセット (Power-on reset の略)。詳細については、Architecture reference manual の Reset system 章を参照してください。

(続く)

用語集

表 16　(続き) 用語集

用語	説明
RTC	Real-time clock (リアルタイムクロックの略)。詳細については、Architecture reference manual の Real-Time Clock 章を参照してください。
SCB	シリアルコミュニケーションブロック (Serial communications block の略)。詳細については、Architecture reference manual の Serial Communications Block (SCB) 章を参照してください。
SRSS	システムリソースサブシステム (System resources subsystem の略)。詳細は Architecture reference manual の System Resources Subsystem セクションを参照してください。
VDDD	デジタル電源。
WCO	時計用水晶発振回路 (Watch crystal oscillator の略)。詳細については、Architecture reference manual の Clocking System 章を参照してください。
WDT	ウォッチドッグタイマ (Watchdog timer の略)。詳細については、Architecture reference manual の Watchdog Timer 章を参照してください。
WFE	Wait for event 命令。
WFI	Wait for interrupt 命令。
WIC	Wakeup interrupt controller (ウェイクアップ割込みコントローラの略)。詳細については、Architecture reference manual の Interrupts 章を参照してください。
XRES	外部リセット端子。詳細については、Architecture reference manual の Reset system 章を参照してください。

関連ドキュメント

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するに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. テクニカルリファレンスマニュアル
   - Body controller entry ファミリ
     - TRAVEO™ T2G automotive body controller entry family architecture reference manual
     - TRAVEO™ T2G automotive body controller entry registers reference manual for CYT2B7
     - TRAVEO™ T2G automotive body controller entry registers reference manual for CYT2B9
     - TRAVEO™ T2G automotive body controller entry registers reference manual for CYT2BL (Doc No. 002-29852)
   - Body controller high ファミリ
     - TRAVEO™ T2G automotive body controller high family architecture reference manual
     - TRAVEO™ T2G automotive body controller high registers reference manual for CYT3BB/4BB
     - TRAVEO™ T2G automotive body controller high registers reference manual for CYT4BF
     - TRAVEO™ T2G automotive body controller high registers reference manual for CYT6BJ (Doc No. 002-36068)
   - Cluster 2D ファミリ
     - TRAVEO™ T2G automotive cluster 2D architecture reference manual
     - TRAVEO™ T2G automotive cluster 2D registers reference manual for CYT3DL
     - TRAVEO™ T2G automotive cluster 2D registers reference manual for CYT4DN
     - TRAVEO™ T2G automotive cluster 2D registers reference manual for CYT4EN (Doc No. 002-35181)
   - クラスタ Entry ファミリ
     - TRAVEO™ T2G automotive cluster entry family architecture reference manual
     - TRAVEO™ T2G automotive cluster entry registers reference manual for CYT2CL
3. アプリケーションノート
   - AN220193 - TRAVEO™ T2G ファミリ GPIO の使用方法
   - AN219944 - TRAVEO™ T2G ファミリ MCU でのウォッチドッグタイマの使用
   - AN219755 - TRAVEO™ T2G 車載マイクロコントローラーでの SAR ADC の使用
   - AN220203 - TRAVEO™ T2G ファミリ スマート I/O の使用方法
   - AN220278 – TRAVEO™ T2G ファミリの CAN FD 使用方法

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) が提供されています。SDL は、公式の AUTOSAR 製品でカバーされていないドライバのお客様 向けリファレンスとしても機能します。SDL は自動車用規格に適合していないため、量産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 を入手するには、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
**	2019-07-10	これは英語版 002-20222 Rev. **を翻訳した日本語版 002-27578 Rev. **です。
*A	2019-12-10	これは英語版 002-20222 Rev. *A を翻訳した日本語版 002-27578 Rev. *A です。 Added target part numbers “CYT4D Series” related information in all instances across the document. Updated Power Modes Transition: Added “CAN wakeup operation”.
*B	2020-04-21	これは英語版 002-20222 Rev. *B を翻訳した日本語版 002-27578 Rev. *B 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 numbers “CYT3 Series” related information in all instances across the document.
英語版*C	2020-12-03	本版は英語版のみの発行です。英語版の改訂内容: Updated power modes transition: Updated cyclic wakeup Operation: Added “Usage of smart I/O in cyclic wakeup”.
英語版*D	2021-04-19	本版は英語版のみの発行です。英語版の改訂内容: Updated to Infineon template.
*C	2022-06-13	これは英語版 002-20222 Rev. *E を翻訳した日本語版 002-27578 Rev. *C です。英語版の改訂内容: Added example of SDL code and description in all instances.
*D	2024-05-29	これは英語版 002-20222 Rev. *F を翻訳した日本語版 002-27578 Rev. *D です。英語版の改訂内容: Template update; no content update.
*E	2025-07-16	これは英語版 002-20222 Rev. *G を翻訳した日本語版 002-27578 Rev. *E です。英語版の改訂内容: Added support to CYT6BJ Updated 関連ドキュメント section

Trademarks

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

Edition 2025-07-16

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-dgq1681442014347

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。