参考資料

Errata

MSP432E4 SimpleLink™ マイクロコントローラ

TEXAS INSTRUMENTS

目次

1 MSP432E4 SimpleLink™ マイクロコントローラ

1.1 はじめに

この文書では、SimpleLink™ MSP432E4 マイクロコントローラの機能仕様に対する既知の例外 (アドバイザリ) について説明します。シリーズ内のすべてのデバイスで一部の機能が利用可能とは限らないため、お使いのデバイスにすべてのエラッタが適用されるとは限りません。詳細については、デバイス固有のデータシートを参照してください。

Arm® Cortex®-M4F CPU のアドバイザリに関する詳細については、*Arm® Cortex®-M4F エラッタ*を参照してください。

1.2 デバイスの命名規則

製品開発サイクルの段階を示すために、TI では MSP432™ MCU デバイスとサポート ツールのすべての型番に接頭辞が割り当てられています。MSP432 MCU 商用ファミリの各メンバーには、MSP または X のいずれかの接頭辞があります。MSP または XMS (たとえば、MSP432E411Y)。各サポートツールには 2 つのいずれかの接頭辞があります:MSP および MSPX。これらの接頭辞は、製品開発の進化段階を表しており、エンジニアリング プロトタイプ (デバイスでは XMS、ツールでは MSPX) から完全に認定済みの量産デバイスおよびツール (デバイスとツールの両方で MSP)まで含まれます。

デバイスの開発進展フロー:

XMS — 実験段階のデバイスで、必ずしも最終的な電気的仕様を反映しない場合があります

MSP — 完全に認定済みの量産版デバイスです。

サポート ツールの開発進展フロー:

MSPX — TI の社内認定テストがまだ完了していない開発サポート製品です。

MSP — 完全に認定済みの開発サポート製品

XMS デバイスと MSPX 開発サポート ツールは、次の免責事項付きで出荷されます。

「開発中の製品は、社内での評価用です。」

MSP デバイスと MSP 開発サポート ツールの特性は完全に明確化されており、デバイスの品質と信頼性が十分に示されてきました。テキサス・インスツルメンツの標準保証が適用されます。

プロトタイプ デバイス (XMS) は、標準の量産デバイスよりも故障率が高いことが予想されます。これらのデバイスは、予測される最終使用時の故障率が未定義であるため、テキサス・インスツルメンツはそれらのデバイスを量産システムで使用しないよう推奨しています。認定済みの量産デバイスのみを使用する必要があります。

1.3 デバイスのマーキング

以下にマイクロコントローラのパッケージシンボルの例を示します。

MSP432E411Y、212 ピンの ZAD (NFBGA) パッケージの例

```
+----------------+
|     \TI/       |      \TI/ = TI Logo
| MSP432(TM)     |      (TM) = Trademark Symbol
| E411YT         |      #    = Die Revision
| xxxxxxxxxx     |      O    = Pin 1
| O  REV #       |
+----------------+
```

MSP432E401Y、128 ピンの PDT (TQFP) パッケージの例

```
+----------------+
| \TI/ xxxxxxxxx |      \TI/ = TI Logo
| MSP432         |      #    = Die Revision
| E401YT         |      O    = Pin 1
| REV #          |
| O              |
+----------------+
```

1.4 エラッタの概要

表 1-1 はデバイスのエラッタを一覧表示します。

表 1-1. デバイス エラッタ 表

名称	エラッタのタイトル
	ADC
ADC#13	ADC アナログ入力チャネルを使用してサンプリングする際、ピン PE3 にグリッチが発生する可能性があります
ADC#14	最初の 2 つの ADC サンプルが不正確である可能性があります
	EPI
EPI#01	EPI モジュールのコードアドレス空間を使用する場合、データ読み取りが破損する可能性があります
	GPIO
GPIO#09	一部の場合、GPIO ピン PB0 および PB1 に注入されるノイズが大電流の消費を引き起こす可能性があります
	汎用タイマー
GPTM#09	汎用タイマは RTC モードに設定されている場合、同期しません
GPTM#15	エッジカウントアップモードで一致に達したとき、カウンタが即座に 0 にクリアされません
	ハイバネーション
HIB#10	MEMCLR が 0 以外の値に設定されている場合、改ざんイベントにより HIBDATA レジスタのすべてのビットがクリアされない可能性があります
HIB#16	クリア中にアプリケーションコードが新しいタンパ イベントを見逃す可能性があります
HIB#18	カレンダーモードで 1 日に 2 つのマッチを取得する可能性があります
HIB#19	ハイバネーション モジュールのリセット後、HIBCTL レジスタへの最初の書き込みが正常に完了しない可能性があります
	メモリ
MEM#07	EEPROM 動作中はソフト リセットをアサートしないでください
MEM#15	いずれかのセクタの特定のフラッシュ位置が消去されません
MEM#16	BOOTCFG が NW = 0 でコミットされている場合の JTAG ロック解除の問題
	PWM
PWM#04	PWM ジェネレータ割り込みは、割り込み発生から 1 PWM クロックサイクル後にのみクリアできます
PWM#05	グローバル同期を持つジェネレータ負荷が誤ったパルス幅を引き起こす可能性があります
PWM#06	PWM 出力が Low の代わりに連続的な High を生成したり、high の代わりに連続的な Low を生成する可能性があります
	QEI
QEI#01	インデックス パルスを使用してカウンタをリセットする際、QEI モジュールの特定の初期条件により、最初のカウントの方向が誤って読み取られる可能性があります
	SSI
SSI#03	SSI1 はレガシー モードでのみ使用できます
SSI#05	SSI のバイモードおよびクワッドモードでのバス競合
SSI#06	SSI 受信 FIFO タイムアウト割り込みがスレーブモードで想定より早くアサートされる可能性があります
SSI#07	SSI 送信割り込みステータスビットがラッチされていません
SSI#08	バイおよびクワッド モードの SSI スレーブが XDAT0 と XDAT1 を交換します
	システム コントロール
SYSCTL#03	MOSC 検証回路はクロックが正常に動作した後、クロック喪失を検出しません
SYSCTL#18	DIVSCLK は DIV = 0x0 の場合、想定と異なるクロック周波数を出力します
SYSCTL#24	電源ドメインが最初にオフされ、その後オンになると、モジュールがアクセス準備ができていない可能性があります
	USB
USB#04	デバイスが USB バスリセットに応答して SE0 を送信します
	ウォッチドッグ タイマ
WDT#08	ウォッチドッグ タイマ 1 を使用している場合、WDTVALUE レジスタの読み出しで誤った値が返される可能性があります

1.5 エラッタの説明

ADC#13 ***ADC アナログ入力チャネルを使用してサンプリングする場合、ピン PE3 にグリッチが発生する可能性があります***

説明

ADC アナログ入力チャネル (AINx) のいずれかを使用してサンプリングする場合、PE3 にグリッチが発生する可能性があります。このグリッチは、PE3 が入力チャネルとして構成されている場合に発生する可能性があり、ADC 変換の終了時に発生します。PE3 が AIN0 として構成されている場合、指定されたソース抵抗が満たされていれば、これらのグリッチはアナログ測定に影響を与えません。

回避方法

PE3 に 1-kΩ の外部プルアップまたはプルダウンを接続すると、グリッチの振幅を 200 mV 以下に最小限に抑えられます。

ADC#14 ***最初の 2 つの ADC サンプルが不正確な可能性があります***

説明

ADC クロックがイネーブルになった後、xCGCADC レジスタで取得された最初の 2 つの ADC サンプルが正しくない可能性があります。

回避方法

1. ADC ペリフェラルクロックがイネーブルになった後、ADC を初期化しサンプルシーケンサをイネーブルする前に、SRADC レジスタを使用して ADC ペリフェラルをリセットしてください。
2. アプリケーションで再構成ができない場合、データを処理する前に最初の 2 つのサンプルを破棄してください。

EPI#01 ***EPI モジュールのコードアドレス空間を使用する場合、データ読み取りが破損する可能性があります***

説明

アドレス 0x1000.0000 の外部コードアドレス空間は、EPI アドレスマップ (EPIADDRMAP) の ECSZ および ECADR フィールドを使用して EPI モジュールに指定されます。ただし、このアドレス空間を使用する場合、データの読み取りが破損する可能性があります。

回避方法

0x1000.0000 アドレス空間からはコードを実行できません。代わりに 0x6000.0000 および 0x8000.0000 の EPI アドレス空間を使用できます。

さらに、0x1000.0000 のコードアドレス空間にマップされた EPI メモリから読み取る場合、読み取り操作のデータサイズに応じて、ポインタを介した直接的な EPI メモリ読み取りを EPIWorkaroundWordRead()、EPIWorkaroundHWordRead()、または EPIWorkaroundByteRead() 関数の呼び出しに置き換えてください。同様に、EPI コードアドレス空間に書き込む場合、直接的な書き込みを EPIWorkaroundWordWrite()、EPIWorkaroundHWordWrite()、または EPIWorkaroundByteWrite() 関数の呼び出しに置き換えてください。これらの API は新しく、セクション 1.7 で参照できます。Keil、IAR、GCC、および Code Bench の場合、これらの関数は *driverlib* フォルダ内の epi.h ファイルにインライン関数として定義されています。CCS がこの構造をサポートしていない場合、これらを *\driverlib* ディレクトリに配置された新しいファイル epi_workaround_ccs.s に追加し、このファイルをプロジェクトに含めてください。セクション 1.7 で言及された新しい DriverLib API および CCS ファイルが SimpleLink MSP432 SDK に含まれていることに注意してください。

GPIO#09 ***一部の場合、GPIO ピン PB0 および PB1 に注入されるノイズが大電流消費を引き起こす可能性があります***

説明

PB0 または PB1 のいずれかが高速に遷移すると、1 つまたは両方のピンとグランド間の低抵抗パスがオンになり、大電流消費を引き起こす可能性があります。

デバイス ピンの信号が 2 ns より速い立ち上がり時間または立ち下がり時間 (VDD の 10% から 90% で測定) を持つ場合、この条件が観測されています。この状態は高温またはノイズの多い環境で発生する可能性が高まります。ピンが入力または出力モード、またはピン多重化オプションのいずれかの場合に発生する可能性があります。

ピンが出力 GPIO として構成されている間にこの条件が発生した場合、ピンの状態を Low に変更し、より低い温度で High 状態に戻すことで条件が解消されます。

回避方法

1. PB0 および PB1 を使用しないでください。1-kΩ 抵抗を介して両方を GND に接続し、GPIO 入力として構成してください。
2. PB0 および PB1 を USB0ID および USB0VBUS として使用する場合、*MSP432E4 マイクロコントローラのシステム設計ガイドライン*の USB セクションを参照し、ドキュメントに記載されたすべてのガイドラインに従ってください。

GPTM#09 ***RTC モードに構成されている場合、汎用タイマが同期しません***

説明

GPTM 同期 (GPTMSYNC) レジスタを使用して汎用タイマを同期しようとすると、いずれかのタイマが RTC モードに設定されている場合、同期されません。

回避方法

なし。

GPTM#15 ***エッジカウントアップモードで一致に達した際、カウンタが即座に 0 にリセットされません***

説明

入力エッジカウントモードおよびカウントアップモードに構成されている場合、一致値に達した後、カウンタが 1 つの追加エッジを使用してタイマを 0 にリセットします。その結果、最初の一致イベントの後、それ以降のすべての一致イベントは、プログラムされたエッジイベント数に 1 を加えた後に発生します。

回避方法

ソフトウェアで、最初の一致割り込みを受信した後、プログラムされたエッジカウントに 1 つの追加エッジを考慮してください。

HIB#10 ***MEMCLR が 0 以外の値に設定されている場合、タンパイベントにより HIBDATA レジスタのすべてのビットがクリアされない可能性があります***

説明

HIB タンパ制御 (HIBTPCTL) レジスタの MEMCLR ビットフィールドが 0 以外の値に設定されている場合、ハイバネーション データ (HIBDATA) レジスタが指定されたビットをクリアしない可能性があります。MEMCLR ビットフィールドは、タンパイベント時にハイバネーション メモリのすべて、上半分、下半分、または何もクリアするオプションを提供します。

回避方法

TPCLR ビットを設定してタンパイベントをクリアした後、アプリケーションは HIBDATA レジスタのハイバネーション メモリデータをクリアしてください (上半分、下半分、またはすべてのビットを 0 に書き込みます)。

HIB#16 *アプリケーションコードがクリア中に新しいタンパイベントを見逃す可能性があります*

説明

タンパイベントのクリア中に、新しいタンパイベントを見逃すか、タンパログが破損する可能性があります。 タンパイベントのクリアは、HIB タンパ制御レジスタ (HIBTPCTL) のタンパクリア (TPCLR) ビットの書き込みから始まります。 書き込みには、32.768 kHz クロックの 3 つの立ち上がりエッジが必要で、クリアが完了します。

回避方法

これら 3 つのハイバネーションクロックサイクル中にタンパイベントを見逃さず、タンパログをリセット状態に復元するには、セクション 1.6 に示すように NMI ハンドラに回避策コードを実装してください。セクション 1.6 で言及された新しい DriverLib API が SimpleLink MSP432E4 SDK に含まれています。

HIB#18 *カレンダーモードで 1 日に 2 回の一致を取得する可能性があります*

説明

ハイバネーション カレンダー制御 (HIBCALCTL) レジスタの CAL24 ビットがクリアされている場合、RTC は 12 時間 AM/PM モードでカウントされます。ハイバネーション カレンダー マッチ 0 (HIBCALM0) の AM/PM ビットは、AM または PM のどちらで一致するかを指定します。ただし、一致が発生しているかどうかを判定する際、このビットは無視されます。その結果、RTC マッチが 1 日に 2 回発生する可能性があります。

回避方法

HIBCALM0 レジスタを構成する前に一致時間を 24 時間モードに調整し、CAL24 ビットを設定してください。または、一致が発生した際に、ハイバネーション カレンダー (HICAL0) レジスタの AM/PM ビットを確認して、一致が正しいか判定してください。

HIB#19 *ハイバネーション モジュールのリセット後、HIBCTL レジスタへの最初の書き込みが正常に完了しない可能性があります*

説明

ハイバネーション モジュールがリセットされた場合、HIBCTL レジスタへの初期書き込みが発生しない可能性があります。HIBCTL レジスタの WRC ビットが設定されない可能性があります。

回避方法

ハイバネーション モジュールのリセット後、WRC ビットが設定されているか確認し、次の手順を実行してください:

- WRC ビットが最大発振器起動時間内に設定されない場合、ハイバネーション モジュールのソフトウェアリセットを実行し、HIBCTL 書き込みを再試行してください。最大発振器起動時間は、データシートの「電気的特性」章にある「ハイバネーション外部発振器 (XOSC) 入力特性」表のハイバネーション XOSC スタートアップ時間パラメータ TSTART で指定されます。
- WRC ビットが設定されているにもかかわらず HIBCTL 書き込みが成功しなかった場合、HIBCTL 書き込みを再試行してください。

MEM#07 *EEPROM 動作中はソフト リセットをアサートしないでください*

説明

EEPROM プログラムまたは消去動作中に次のソフトリセットのいずれかがアサートされると、EEPROM データが破損する可能性があります:

- ソフトウェアリセット (SYSRESREQ)
- EEPROM モジュールのソフトウェアペリフェラルリセット
- ウォッチドッグリセット (RESBEHAVCTL レジスタでシステムリセットとして構成されている場合)
- MOSC 障害リセット

MEM#07 (続き) ***EEPROM 動作中はソフト リセットをアサートしないでください***

- BOR リセット (RESBEHAVCTL レジスタでシステムリセットとして構成されている場合)
- 外部リセット (RESBEHAVCTL レジスタでシステムリセットとして構成されている場合)
- HSSR レジスタへの書き込み

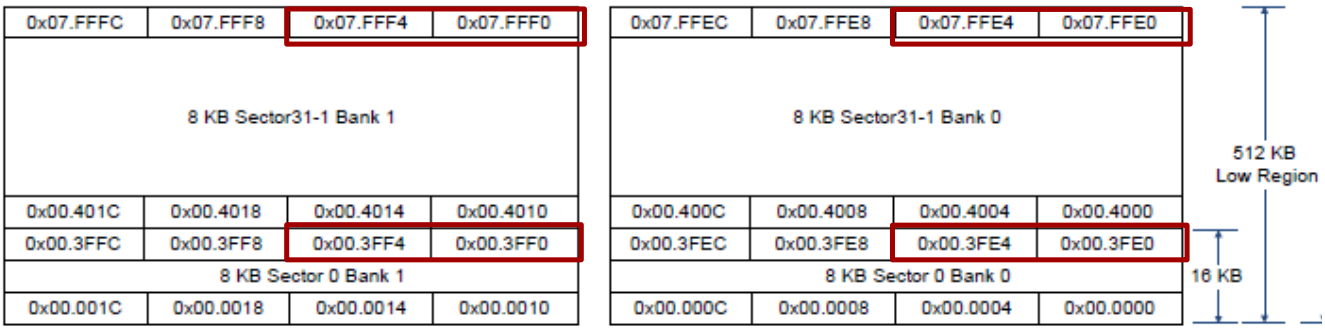
回避方法

EEPROM プログラムまたは消去動作中に上記のソフトリセットがアサートされないようにしてください。リセットがアサートされる前に EEDONE レジスタの WORKING ビットを確認し、EEPROM プログラムまたは消去動作が発生しているかどうかをチェックできます。デバッガを使用する際、ソフトリセットが発生する可能性があり、EEPROM 動作中は避けてください。時間に制約がない場合、GPIO またはハイバネーションを使用してウォッチドッグリセットなどのリセットを外部リセットにマッピングできます。

MEM#15 ***どのセクタの特定のフラッシュ位置も消去されません***

説明

バンク内のセクタの最後のラインの最初の 2 ワードのうち 1 つまたは両方がプログラムされている場合、セクタ全体の消去、デバイスの一括消去、または一括消去の切り替えでも、フラッシュがすべて 1 に消去されません。下の図は、下位 512KB のフラッシュのセクタ 0 およびセクタ 31 で影響を受けるワードを示しています。



回避方法

セクター消去が機能するように、セクターバンク内の他のワードをプログラムしてください。

注

一括消去またはその切り替えでは依然として機能しません。

MEM#16 ***BOOTCFG が NW = 0 でコミットされている場合の JTAG ロック解除問題、***

説明

BOOTCFG レジスタを NW = 0、DBG1 = 0、DBG = 1 で構成した後、JTAG デバッガアクセスがロックされ、1 回のロック解除シーケンス実行では解除できません。

回避方法

デバイスのロックを解除するには、ロック解除シーケンスを 2 回実行する必要があります。

PWM#04 ***PWM ジェネレータの割り込みは、割り込み発生後 1 PWM クロックサイクルでのみクリア可能です***

説明

PWMxISC レジスタに 1 を書き込むと、次のシステムクロックで対応するジェネレータ割り込みステータスがクリアされることが期待されます。ただし、書き込みにより次の PWM クロックでジェネレ

PWM#04 (続き) ***PWM ジェネレータの割り込みは、割り込み発生後 1 PWM クロックサイクルでのみクリア可能です***

ータ割り込みステータスがクリアされます。次の PWM クロック前に PWMxISC への書き込みで割り込みをクリアしようとすると無視され、割り込みが再度アサートされます。

回避方法

割り込みがアサートされた後、CPU は PWMxISC に 1 を書き込んで対応するジェネレータ割り込みステータスをクリアする前に、1 PWM クロックサイクル待機してください。PWM クロック分周値が大きいほど、割り込みをクリアするシステムの遅延時間が長くなります。

PWM#05 ***グローバル同期を使用したジェネレータ負荷が誤ったパルス幅を引き起こす可能性があります***

説明

ジェネレータタイマが古い値でカウントを続けている場合でも、新しいコンパレータ値がロードされ、誤った High パルス幅が発生する状況があります。

回避方法

アプリケーションは割り込みステータスを使用して、新しい負荷値とコンパレータ値を書き込む必要があります。ダウンカウントモードを使用する場合、対応するコンパレータダウンカウント一致割り込みステータスビットの未処理割り込みステータスビットをクリアし、再度設定されるのを待ってから、負荷とコンパレーター一致の新しい値を更新してください。

PWM#06 ***PWM 出力が low の代わりに連続的な high を生成するか、high の代わりに連続的な low を生成する可能性があります***

説明

アップダウンカウントモードで PWM を使用する場合、PWM ジェネレータが次の条件で Low の代わりに連続的な High を生成するか、High の代わりに連続的な Low を生成する可能性があります:

- PWM ジェネレータは、コンパレータ A または B の上下動作を使用して PWM サイクルを制御します。
- PWM コンパレータ A または B が 2 以上のカウントから PWM ロードカウント値に変更されます。
- 反転オプションが有効な場合、連続的な High が生成されます。
- 反転オプションが無効な場合、連続的な Low が生成されます。

回避方法

- PWM のダウンカウントモードを使用してください
- コンパレータ負荷レジスタの値 (1 以外) から PWM 負荷カウント値に変更する場合、PWM 負荷カウント値に切り替える前に必ず 1 の値にしてください。

QEI#01 ***インデックスパルスを使用してカウンタをリセットする場合、QEI モジュールの特定の初期条件により、最初のカウントの方向が誤って読み取られる可能性があります***

説明

QEI 制御 (QEICTL) レジスタの次の構成でインデックスパルスを使用してカウンタをリセットする場合:

- SIGMODE が 0 の場合、直交モードを示します
- CAPMODE が 1 の場合、PhA および PhB の両方のエッジがカウントされることを示します

および次の初期条件:

- PhA および PhB がともに 0 です
- 次の直交状態が反時計回り方向になります

QEI#01 (続き) ***インデックスパルスを使用してカウンタをリセットする場合、QEI モジュールの特定の初期条件により、最初のカウントの方向が誤って読み取られる可能性があります***

QEI が状態変化を時計回りの更新と解釈し、位置の不一致が 2 になります。

回避方法

なし。

SSI#03 ***SSI1 はレガシーモードでのみ使用可能です***

説明

バイ、クワッド、およびアドバンスモードの動作が、指定された SSI モジュールで正しく機能しません。その結果、影響を受けるモジュールはレガシーモードでの操作にのみ使用できます。

回避方法

バイ、クワッド、およびアドバンスモードの動作には SSI0、SSI2、または SSI3 を使用してください。レガシーモードの動作には SSI1 を使用してください。

SSI#05 ***SSI のバイおよびクワッドモードでのバス競合***

説明

SSI がバイまたはクワッドモードに構成され、SSI が受信モードに設定された後、外部メモリからの読み取りを実行すると、最初のデータ読み取り時に SSI データピンでバス競合が発生する可能性があります。

回避方法

SSI を受信モードに設定した後 (QSSI 制御 (SSICR1) レジスタの DIR ビットを設定)、最初の有効な読み出し動作の前にメモリからダミー読み出しを実行してください。次に例を示します。

```
    SSIConfigSetExpClk(SSI0_BASE,SysCtlClockFreqSet (),
    SSI_FRF_MOTO_MODE_0, SSI_MODE_MASTER, 1000000, 8);
    SSIAdvModeSet(SSI0_BASE, SSI_ADV_BI_READ); //Receive Mode set
    SSIDataPut(SSI0_BASE, &pui32DataRx[ui32Index]); //intentional dummy write
    SSIDataGetNonBlocking(SSI-_BASE, ui32Dummy); //dummy read
    SSIDataPut(SSI0_BASE, &pui32DataRx[ui32Index]); //intentional dummy write
    SSIDataGetNonBlocking(SSI0_BASE, &pui32DataRx[0])); //first intentional
read
    SSIDataPut(SSI0_BASE, &pui32DataRx[ui32Index]); //intentional dummy write
    SSIDataGetNonBlocking(SSI0_BASE, &pui32DataRx[1])); //second intentional
read
```

転送で通常上記のような意図的なダミー書き込みなどのダミー操作が必要な場合、通常のダミー操作の前にダミー読み出しを実行してください。

アプリケーションが SSIClk に敏感な場合、ダミー読み出しがクロックサイクルを出力することに注意してください。ダミー読み出しを実行する間、SSIClk ピンを GPIO 入力に再構成して、クロックに敏感なアプリケーションに影響を与えないでください。

SSI#06 ***SSI 受信 FIFO タイムアウト割り込みがスレーブモードで想定より早くアサートされる可能性があります***

説明

SSI クロックプリスケール (SSICPSR) レジスタの CPSDVSR フィールドが 0x2 より大きい値に設定されている場合、SSI 受信 FIFO タイムアウト割り込みがスレーブモードで 32 システムクロック周期より早くアサートされる可能性があります。マスターモードはこの動作の影響を受けません。

SSI#06 (続き) ***SSI 受信 FIFO タイムアウト割り込みがスレーブモードで想定より早くアサートされる可能性があります***

回避方法

場合によっては、ソフトウェアが SSI 制御 0 (SSICR0) レジスタの SCR フィールドを CPSDVSR フィールド値 0x2 と組み合わせて使用し、同じ SSI クロック周波数を実現できます。たとえば、目的のシリアルクロックレートが SysClk/48 の場合、式 SSInCLK = SysClk / (CPSDVSR * (1 + SCR)) を使用して同じクロックレートを実現するには、CPSDVSR = 0x18 および SCR = 0x1 の代わりに CPSDVSR = 0x2 および SCR = 0x17 を使用できます。必要なシリアルクロックレートを達成するために CPSDVSR = 0x2 と組み合わせられる SCR 値がない場合、受信 FIFO タイムアウト機能は使用できません。

SSI#07 ***SSI 送信割り込みステータスビットがラッチされません***

説明

SSI 送信割り込みステータスビットが正しく動作しません。

- マスターモードで、送信 FIFO が半分以下の場合に割り込みが有効です。SSIMIS は TXFIFO が FIFO スレッショールドの半分を下回るたびにアサートされ、SSIICR レジスタを使用して割り込み状態をクリアしても、割り込みが常にアサートされます

回避方法

バッファから SSIDR への転送が完了した際の割り込みハンドラで、SSIIM.TXIM をクリアしてそれ以上の割り込みを停止してください。送信に次の割り込みセットが必要な場合、SSIIM.TXIM ビットを設定してください。

SSI#08 ***バイおよびクワッドモードの SSI スレーブが XDAT0 と XDAT1 を交換します***

説明

BI またはクワッドモードのとき、SSI スレーブは XDAT0 から XDAT1 ラインへ、XDAT1 から XDAT0 ラインへデータを送信します。

回避方法

BI およびクワッドモードで SSI スレーブからデータを送信する際、CPU はデータレジスタ SSIDR に書き込むときに正しいビットを交換して送信する必要があります。uDMA を使用する場合、CPU は uDMA のソースバッファにスワップされたビットを書き込む必要があります。

SYSCTL#03 ***クロックが正常に動作した後、MOSC 検証回路がクロック喪失を検出しません***

説明

MOSC クロックソースが電源投入され正常に動作した後、取り外されたりフラットラインになった場合、MOSC 検証回路はエラー状態を示しません。

回避方法

MOSC に障害が発生した場合、システムをリセットするために PIOSC で動作するウォッチドッグモジュール 1 を使用してください。

SYSCTL#18 ***DIVSCLK は DIV = 0x0 の場合、期待されるクロック周波数と異なる出力を行います***

説明

Divisor およびソースクロック構成 (DIVSCLK) レジスタで、DIV ビットフィールドが 0x0 (1 分周) の場合、GPIO へのクロック出力が期待される値ではありません。

- DIV ビットフィールドがコードでまだ調整されていない場合、クロック周波数は SRC ビットフィールドで定義されたクロックソースを 32 で割った値になります。たとえば、SRC = 0x1

SYSCTL#18 (続き) ***DIVSCLK は DIV = 0x0 の場合、期待されるクロック周波数と異なる出力を行います***

(PIOSC) および DIV = 0x0 の場合、GPIO へのクロック出力は PIOSC から導出される 500 kHz の周波数になります。

- DIV ビットフィールドがコード内で既に調整されている場合、クロック周波数は SRC ビットフィールドで定義されたクロックソースを、以前の DIV ビットフィールド値で割った値になります。たとえば、SRC = 0x1 (PIOSC) かつ DIV が以前 0x1 (2 分周) であった後、0x0 に書き込まれた場合、GPIO へのクロック出力は PIOSC から導出される 8MHz の周波数になります。

回避方法

ソースのクロック精度が問題でない場合、ゼロでない DIV 値と異なる SRC 値を使用して特定の周波数を実現できます。たとえば、16MHz クロックを実現するには、SRC = 0x1 (PIOSC) および DIV = 0x0 の代わりに、SRC = 0x0 (システムクロック) および対応する DIV 値 (システムクロックが 80MHz の場合、DIV = 0x4 (5 分周)) を使用してください。

SYSCTL#24 ***電源ドメインが最初にオフされ、その後オンになると、モジュールがアクセス準備ができていない可能性があります***

説明

このデバイスには、別個の電力ドメインに存在する 5 つのモジュール (CAN、イーサネット MAC、イーサネット PHY、USB、CCM) が含まれています。これらのモジュールは、電力ドメインをオフにすることで追加のリーク電流を節約できます。電源ドメインをオフにしてからオンにした場合、モジュールは PRUSB レジスタなどの各ペリフェラル準備完了レジスタでポーリングされた際に、アクセス準備ができていない状態を無期限に示す可能性があります。

回避方法

なし。電源ドメインをオフにしないでください。

USB#04 ***デバイスが USB バスリセットに応答して SE0 を送信します***

説明

USB デバイスは、ホストからの USB バスリセットに応答して、シングルエンドゼロ (SE0) バス状態 (USB0DP および USB0DM を low に駆動) を送信します。USB 仕様によれば、USB バスがリセットされた場合、デバイスはこれらのピンを駆動しないでください。これは USB 認証に影響を与えません。

回避方法

なし。

WDT#08 ***ウォッチドッグタイマ 1 を使用する場合、WDTVALUE レジスタの読み取りで誤った値が返される可能性があります***

説明

ウォッチドッグタイマ 1 を使用する場合、ウォッチドッグタイマ 1 のベースアドレスにあるウォッチドッグ値 (WDTVALUE) レジスタから誤った値が読み取られる可能性があります。

回避方法

なし。

1.6 付録 1

エラッタに対処するため、HIB#16 アプリケーションコードがクリア中に新しいタンパイベントを見逃す可能性があります HibernateTamperEventsClear() API を次の API に置き換える必要があります:

```
HibernateTamperEventsClearNoLock();
HibernateTamperUnLock();
HibernateTamperLock();
```

API の定義は次のとおりです:

```
//*****************************************************************************
//
//! Clears the tamper feature events without Unlock and Lock.
//!
//! This function is used to clear all tamper events without unlock/locking
//! the tamper control registers, so API HibernateTamperUnLock() should be
//! called before this function, and API HibernateTamperLock() should be
//! called after to ensure that tamper control registers are locked.
//!
//! This function doesn't block until the write is complete.
//! Therefore, care must be taken to ensure the next immediate write will
//! occure only after the write complete bit is set.
//!
//! This function is used to implement a software workaround in NMI interrupt
//! handler to fix an issue when a new tamper event could be missed during
//! the clear of current tamper event.
//!
//! \note The hibernate tamper feature is not available on all
//! devices.  Please consult the data sheet for the device that you
//! are using to determine if this feature is available.
//!
//! \return None.
//
//*****************************************************************************
void
HibernateTamperEventsClearNoLock(void)
{
    //
    // Wait for write completion.
    //
    _HibernateWriteComplete();
    //
    //
    // Set the tamper event clear bit.
    //
    HWREG(HIB_TPCTL) |= HIB_TPCTL_TPCLR;
}
//*****************************************************************************
//
//! Unlock temper registers.
//!
//! This function is used to unlock the temper control registers.  This
//! function should be only used before calling API
//! HibernateTamperEventsClearNoLock().
//!
//! \note The hibernate tamper feature is not available on all
//! devices.  Please consult the data sheet for the device that you
//! are using to determine if this feature is available.
//!
//! \return None.
//
//*****************************************************************************
void
HibernateTamperUnLock(void)
{
    //
    // Unlock the tamper registers.
    //
    HWREG(HIB_LOCK) = HIB_LOCK_HIBLOCK_KEY;
    _HibernateWriteComplete();
}
//*****************************************************************************
//
//! Lock temper registers.
```

```
//!
//! This function is used to lock the temper control registers.  This
//! function should be used after calling API
//! HibernateTamperEventsClearNoLock().
//!
//! \note The hibernate tamper feature is not available on all
//! devices.  Please consult the data sheet for the device that you
//! are using to determine if this feature is available.
//!
//! \return None.
//
//*****************************************************************************
void
HibernateTamperLock(void)
{
    //
    // Wait for write completion.
    //
    _HibernateWriteComplete();
    //
    // Lock the tamper registers.
    //
    HWREG(HIB_LOCK) = 0;
    _HibernateWriteComplete();
}
```

ソフトウェアの回避策は NMI ハンドラに追加する必要があります。このコードは、主にタンパクリア同期中にタンパログエントリをポーリングします。

この回避策を使用した NMI ハンドラの例を以下に示します。

```
static uint32_t g_ui32RTCLog[4];
static uint32_t g_ui32EventLog[4];
//*****************************************************************************
//
// Handles an NMI interrupt generated by a Tamper event.
//
//*****************************************************************************
void
NMITamperEventHandler(void)
{
    uint32_t ui32NMIStatus, ui32TamperStatus;
    uint32_t pui32Buf[3];
    uint8_t ui8Idx, ui8StartIdx;
    bool     bDetectedEventsDuringClear;
    //
    // Get the cause of the NMI event.
    //
    ui32NMIStatus = SysCtlNMIStatus();
    //
    // We should have got the cause of the NMI event from the above function.
    // But in Snowflake RA0 the NMIC register is not set correctly when an
    // event occurs.  So as a work around check if the NMI event is caused by a
    // tamper event and append this to the return value from SysCtlNMIStatus().
    // This way only this section can be removed once the bug is fixed in next
    // silicon rev.
    //
    ui32TamperStatus = HibernateTamperStatusGet();
    if(ui32TamperStatus & (HIBERNATE_TAMPER_STATUS_EVENT |
                           HIBERNATE_TAMPER_STATUS_EXT_OSC_FAILED))
    {
        ui32NMIStatus |= SYSCTL_NMI_TAMPER;
    }
    //
    // Check if SysCtlNMIStatus() returned a valid value.
    //
    if(ui32NMIStatus)
    {
        //
        // Check if the NMI Interrupt is due to a Tamper event.
        //
        if(ui32NMIStatus & SYSCTL_NMI_TAMPER)
        {
            //
            // If the previous NMI event has not been processed by main
            // thread, we need to OR the new event along with the old ones.
```

```
        //
        if(g_ui32NMIEvent == 0)
        {
            //
            // Reset variables that used for tamper event.
            //
            g_ui32TamperEventFlag = 0;
            g_ui32TamperRTCLog = 0;
            //
            // Clean the log data for debugging purpose.
            //
            memset(g_ui32RTCLog, 0, (sizeof(g_ui32RTCLog))<<2);
            memset(g_ui32EventLog, 0, (sizeof(g_ui32EventLog))<<2);
        }
        //
        // Log the tamper event data before clearing tamper events.
        //
        for(ui8Idx = 0; ui8Idx< 4; ui8Idx++)
        {
            if(HibernateTamperEventsGet(ui8Idx,
                                        &g_ui32RTCLog[ui8Idx],
                                        &g_ui32EventLog[ui8Idx]))
            {
                //
                // Event in this log entry, store it.
                //
                g_ui32TamperEventFlag |= g_ui32EventLog[ui8Idx];
                g_ui32TamperRTCLog = g_ui32RTCLog[ui8Idx];
            }
            else
            {
                //
                // No event in this log entry. Done checking the logs.
                //
                break;
            }
        }
        //
        // Process external oscillator failed event.
        //
        if(ui32TamperStatus & HIBERNATE_TAMPER_STATUS_EXT_OSC_FAILED)
        {
            g_ui32TamperXOSCFailEvent++;
            g_ui32TamperEventFlag |= HIBERNATE_TAMPER_EVENT_EXT_OSC;
            g_ui32TamperRTCLog = HWREG(HIB_TPLOG0);
        }
```

注

これは回避策コードのブロックの開始です。

```
        // The following block of code is to workaround hardware defect
        // which results in missing new tamper events during tamper clear
        // synchronization.
        //
        // There is a window after the application code writes the tamper
        // clear where a new tamper event can be missed if the application
        // requires more than one tamper event pins detection.
        //
        // The tamper Clear is synchronized to the hibernate 32kHz clock
        // domain.  The clear takes 3 rising edges of the 32KHz clock.
        // During this window, new tamper events could be missed.
        // A software workaround is to poll the tamper log during the
        // tamper event clear synchronization.
        //
        //
        // Clear the flag for the case there are events triggered
        // during clear execution.
        //
        bDetectedEventsDuringClear = false;
        //
        // Unlock the Tamper Control register. This is required before
        // calling HibernateTamperEventsClearNoLock().
        //
```

```
            HibernateTamperUnLock();
            do
            {
                //
                // We will start to poll the log registers at index 1 for
                // any new events.
                //
                ui8StartIdx = 1;
                //
                // Clear the Tamper event.
                // Note this API doesn't wait for synchronization, which
                // allows us to check the tamper log during
                // synchronization.
                //
                HibernateTamperEventsClearNoLock();
                //
                // Check new tamper event during tamper event clear
                // synchronization.
                // This will take about 92us(three clock cycles) at most.
                //
                while(HibernateTamperStatusGet() & HIBERNATE_TAMPER_STATUS_EVENT)
                {
                    //
                    // Clear execution isn't done yet , poll for new events.
                    // If there were any new event, it will be logged in log 1
                    // registers and so on.
                    //
                    for(ui8Idx = ui8StartIdx; ui8Idx< 4; ui8Idx++)
                    {
                        if(HibernateTamperEventsGet(ui8Idx,
                                                    &g_ui32RTCLog[ui8Idx],
                                                    &g_ui32EventLog[ui8Idx]))
                        {
                            //
                            // detected new event, store it.
                            //
                            g_ui32TamperEventFlag |= g_ui32EventLog[ui8Idx];
                            //
                            // check for more event.
                            //
                            continue;
                        }
                        else
                        {
                            //
                            // no new event in this log, update the log index
                            // to be checked next, and break out of loop.
                            //
                            ui8StartIdx = ui8Idx;
                            break;
                        }
                    }
                    //
                    // all last three logs have info. Check if all 4 logs
                    // have the same info. This is to detect the case that
                    // events happen during clear execution.
                    //
                    if(ui8Idx == 4)
                    {
                        //
                        // If events happens during clear
                        // execution, all four log registers will be
                        // logged with the same event, to detect this
                        // condition, we will compare with all four log data.
                        //
                        if(HibernateTamperEventsGet(0, &g_ui32RTCLog[0], &g_ui32EventLog[0]))
                        {
                            if((g_ui32RTCLog[0] == g_ui32RTCLog[1])     &&
                               (g_ui32EventLog[0] == g_ui32EventLog[1]) &&
                              (g_ui32RTCLog[0] == g_ui32RTCLog[2])     &&
                               (g_ui32EventLog[0] == g_ui32EventLog[2]) &&
                               (g_ui32RTCLog[0] == g_ui32RTCLog[3])     &&
                               (g_ui32EventLog[0] == g_ui32EventLog[3]))
                            {
                                //
                                // Detected events during clear execution.
                                // Event logging takes priority, the clear
                                // will not be done in this case. We will need
```

```
                                // to go back to the beginning of the loop and
                                // clear the events.
                                //
                                if(bDetectedEventsDuringClear)
                                {
                                    //
                                    // This condition has already detected,
                                    // we have cleared the event,
                                    // clear the flag.
                                    //
                                    bDetectedEventsDuringClear = false;
                                }
                                else
                                {
                                    // This is the first time it has been
                                    // detected, set the flag.
                                    //
                                    bDetectedEventsDuringClear = true;
                                }
                                //
                                // Break out of while loop so that we can
                                // clear the events, and start the
                                // workaround all over again.
                                //
                                break;
                            }
                        }
                        else
                        {
                            //
                            // Log 0 didn't detect any events. So this is not
                            // the case of missing events during clear
                            // execution.
                            // Update the log index at which we will poll next.
                            // It should be the last log entry that OR all the
                            // new events.
                            //
                            ui8StartIdx = 3;
                        }
                    }
                }
            }
            while(bDetectedEventsDuringClear);
            //
            // Lock the Tamper Control register.
            //
            HibernateTamperLock();
```

注

これで回避策コードのブロックは終了します。

```
            //
            // Save the tamper event and RTC log info in the Hibernate Memory
            //
            HibernateDataGet(pui32Buf, 3);
            pui32Buf[1] = g_ui32TamperEventFlag;
            pui32Buf[2] = g_ui32TamperRTCLog;
            HibernateDataSet(pui32Buf, 3);
            //
            // Signal the main loop that an NMI event occurred.
            //
            g_ui32NMIEvent++;
        }
        //
        // Clear NMI events
        //
        SysCtlNMIClear(ui32NMIStatus);
    }
}
```

1.7 付録 2

エラッタに対処するため、EPI#01 EPI モジュールのコードアドレス空間を使用する場合、データ読み取りが破損する可能性があります epi.h ファイルに次のコードを追加する必要があります。

```
#ifdef rvmdk
//*****************************************************************************
//
// Keil case.
//
//*****************************************************************************
inline void
EPIWorkaroundWordWrite(uint32_t *pui32Addr, uint32_t ui32Value)
{
  uint32_t ui32Scratch;
    __asm
    {
        //
        // Add a NOP to ensure we don’t have a flash read immediately before
        // the EPI read.
        //
        NOP
        //
        // Perform the write we're actually interested in.
        //
        STR ui32Value, [pui32Addr]
        //
        // Read from SRAM to ensure that we don't have an EPI write followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }
}
inline uint32_t
EPIWorkaroundWordRead(uint32_t *pui32Addr)
{
  uint32_t ui32Value, ui32Scratch;
    __asm
    {
        //
        // Add a NOP to ensure we don’t have a flash read immediately before
        // the EPI read.
        //
        NOP
        //
        // Perform the read we're actually interested in.
        //
        LDR ui32Value, [pui32Addr]
        //
        // Read from SRAM to ensure that we don't have an EPI read followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }
    return(ui32Value);
}
inline void
EPIWorkaroundHWordWrite(uint16_t *pui16Addr, uint16_t ui16Value)
{
    uint32_t ui32Scratch;
    __asm
    {
        //
        // Add a NOP to ensure we don’t have a flash read immediately before
        // the EPI read.
        //
        NOP
        //
        // Perform the write we're actually interested in.
        //
        STRH ui16Value, [pui16Addr]
        //
        // Read from SRAM to ensure that we don't have an EPI write followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }
```

```
}
inline uint16_t
EPIWorkaroundHWordRead(uint16_t *pui16Addr)
{
    uint32_t ui32Scratch;
    uint16_t ui16Value;
    __asm
    {
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        NOP
        //
        // Perform the read we're actually interested in.
        //
        LDRH ui16Value, [pui16Addr]
        //
        // Read from SRAM to ensure that we don't have an EPI read followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }
    return(ui16Value);
}
inline void
EPIWorkaroundByteWrite(uint8_t *pui8Addr, uint8_t ui8Value)
{
  uint32_t ui32Scratch;
    __asm
    {
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        NOP
        //
        // Perform the write we're actually interested in.
        //
        STRB ui8Value, [pui8Addr]
        //
        // Read from SRAM to ensure that we don't have an EPI write followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }
}
inline uint8_t
EPIWorkaroundByteRead(uint8_t *pui8Addr)
{
    uint32_t ui32Scratch;
    uint8_t ui8Value;
    __asm
    {
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        NOP
        //
        // Perform the read we're actually interested in.
        //
        LDRB ui8Value, [pui8Addr]
        //
        // Read from SRAM to ensure that we don't have an EPI read followed by
        // a flash read.
        //
        LDR ui32Scratch, [__current_sp()]
    }
    return(ui8Value);
}
#else
#ifdef ccs
//*****************************************************************************
//
// Code Composer Studio versions of these functions can be found in separate
// source file epi_workaround_ccs.s.
//
```

```
//*****************************************************************************
extern void EPIWorkaroundWordWrite(uint32_t *pui32Addr, uint32_t ui32Value);
extern uint32_t EPIWorkaroundWordRead(uint32_t *pui32Addr);
extern void EPIWorkaroundHWordWrite(uint16_t *pui16Addr, uint16_t ui16Value);
extern uint16_t EPIWorkaroundHWordRead(uint16_t *pui16Addr);
extern void EPIWorkaroundByteWrite(uint8_t *pui8Addr, uint8_t ui8Value);
extern uint8_t EPIWorkaroundByteRead(uint8_t *pui8Addr);
#else
//*****************************************************************************
//
// GCC and IAR case.
//
//*****************************************************************************
inline void
EPIWorkaroundWordWrite(uint32_t *pui32Addr, uint32_t ui32Value)
{
    volatile register uint32_t ui32Scratch;
    __asm volatile (
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    STR %[value],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         :  [scratch] "=r" (ui32Scratch)
         :  [addr] "r" (pui32Addr), [value] "r" (ui32Value)
     );
    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;
}
inline uint32_t
EPIWorkaroundWordRead(uint32_t *pui32Addr)
{
    volatile register uint32_t ui32Data, ui32Scratch;
    //
    // ui32Scratch is not used other than to add a padding read following the
    // "real" read.
    //
    __asm volatile(
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    LDR %[ret],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         : [ret] "=r" (ui32Data),
           [scratch] "=r" (ui32Scratch)
         : [addr] "r" (pui32Addr)
    );
    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;
    return(ui32Data);
}
inline void
EPIWorkaroundHWordWrite(uint16_t *pui16Addr, uint16_t ui16Value)
{
    volatile register uint32_t ui32Scratch;
    __asm volatile (
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    STRH %[value],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         :  [scratch] "=r" (ui32Scratch)
         :  [addr] "r" (pui16Addr), [value] "r" (ui16Value)
     );
    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;
```

```
}
inline uint16_t
EPIWorkaroundHWordRead(uint16_t *pui16Addr)
{
    register uint16_t ui16Data;
    register uint32_t ui32Scratch;
    //
    // ui32Scratch is not used other than to add a padding read following the
    // "real" read.
    //
    __asm volatile(
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    LDRH %[ret],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         : [ret] "=r" (ui16Data),
           [scratch] "=r" (ui32Scratch)
         : [addr] "r" (pui16Addr)
    );
    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;
    return(ui16Data);
}
inline void
EPIWorkaroundByteWrite(uint8_t *pui8Addr, uint8_t ui8Value)
{
    volatile register uint32_t ui32Scratch;
    __asm volatile (
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    STRB %[value],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         :  [scratch] "=r" (ui32Scratch)
         :  [addr] "r" (pui8Addr), [value] "r" (ui8Value)
     );
    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;
}
inline uint8_t
EPIWorkaroundByteRead(uint8_t *pui8Addr)
{
    register uint8_t ui8Data;
    register uint32_t ui32Scratch;
    //
    // ui32Scratch is not used other than to add a padding read following the
    // "real" read.
    //
    __asm volatile(
        //
        // Add a NOP to ensure we don't have a flash read immediately before
        // the EPI read.
        //
        "    NOP\n"
        "    LDRB %[ret],[%[addr]]\n"
        "    LDR %[scratch],[sp]\n"
         : [ret] "=r" (ui8Data),
           [scratch] "=r" (ui32Scratch)
         : [addr] "r" (pui8Addr)
    );
    //
    // Keep the compiler from generating a warning.
    //
    ui32Scratch = ui32Scratch;
    return(ui8Data);
}
#endif
```

さらに、CCS を使用する場合、以下のコードを epi_workaround_ccs.s というファイル名で *driverlib* ディレクトリ に保存し、プロジェクトに含める必要があります:

```
;*******************************************************************************
;
; epi_workaround_ccs.s - EPI memory access functions.
;
; Copyright (c) 2013 Texas Instruments Incorporated.  All rights reserved.
; TI Information - Selective Disclosure
;
;*******************************************************************************
;*******************************************************************************
;
; void EPIWorkaroundWordWrite(uint32_t *pui32Addr, uint32_t ui32Value)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundWordWrite"
    .global EPIWorkaroundWordWrite
EPIWorkaroundWordWrite:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop
    ;
    ; Store the word in EPI memory.
    ;
    str r1, [r0]
    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [sp]
    ;
    ; Return to the caller.
    ;
    bx lr
    .align 4
;*******************************************************************************
;
; uint32_t EPIWorkaroundWordRead(uint32_t *pui32Addr)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundWordRead"
    .global EPIWorkaroundWordRead
EPIWorkaroundWordRead:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop
    ;
    ; Read the word from EPI memory.
    ;
    ldr r0, [r0]
    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [r13]
    ;
    ; Return to the caller.
    ;
    bx lr
    .align 4
;*******************************************************************************
;
; void EPIWorkaroundHWordWrite(uint16_t *pui16Addr, uint16_t ui16Value)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundHWordWrite"
    .global EPIWorkaroundHWordWrite
EPIWorkaroundHWordWrite:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
```

```
    nop
    ;
    ; Store the word in EPI memory.
    ;
    strh r1, [r0]
    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [sp]
    ;
    ; Return to the caller.
    ;
    bx lr
    .align 4
;*******************************************************************************
;
; uint16_t EPIWorkaroundHWordRead(uint16_t *pui16Addr)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundHWordRead"
    .global EPIWorkaroundHWordRead
EPIWorkaroundHWordRead:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop
    ;
    ; Read the half word from EPI memory.
    ;
    ldrh r0, [r0]
    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [r13]
    ;
    ; Return to the caller.
    ;
    bx lr
    .align 4
;*******************************************************************************
;
; void EPIWorkaroundByteWrite(uint8_t *pui8Addr, uint8_t ui8Value)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundByteWrite"
    .global EPIWorkaroundByteWrite
EPIWorkaroundByteWrite:
    ;
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop
    ;
    ; Store the byte in EPI memory.
    ;
    strb r1, [r0]
    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [sp]
    ;
    ; Return to the caller.
    ;
    bx lr
    .align 4
;*******************************************************************************
;
; uint8_t EPIWorkaroundByteRead(uint8_t *pui8Addr)
;
;*******************************************************************************
    .sect ".text:EPIWorkaroundByteRead"
    .global EPIWorkaroundByteRead
EPIWorkaroundByteRead:
    ;
```

```
    ; Include a no-op to ensure that we don't have a flash data access
    ; immediately before the EPI access.
    ;
    nop
    ;
    ; Read the byte from EPI memory.
    ;
    ldrb r0, [r0]
    ;
    ; Make a dummy read from the stack to ensure that we don't have a flash
    ; data access immediately after the EPI access.
    ;
    ldr r1, [r13]
    ;
    ; Return to the caller.
    ;
    bx lr
    .align 4
.end
```

2 商標

SimpleLink™ and MSP432™ are trademarks of Texas Instruments.
Arm® and Cortex® are registered trademarks of Arm Limited.
すべての商標は、それぞれの所有者に帰属します。

3 改訂履歴

Changes from OCTOBER 1, 2017 to JUNE 30, 2025 (from Revision * (October 2017) to Revision A (June 2025)) **Page**

- SYSCTL #24 アドバイザリを追加 12

重要なお知らせと免責事項

テキサス・インスツルメンツは、技術データと信頼性データ (データシートを含みます)、設計リソース (リファレンス デザインを含みます)、アプリケーションや設計に関する各種アドバイス、Web ツール、安全性情報、その他のリソースを、欠陥が存在する可能性のある「現状のまま」提供しており、商品性および特定目的に対する適合性の黙示保証、第三者の知的財産権の非侵害保証を含むいかなる保証も、明示的または黙示的にかかわらず拒否します。

これらのリソースは、 テキサス・インスツルメンツ製品を使用する設計の経験を積んだ開発者への提供を意図したものです。(1) お客様のアプリケーションに適した テキサス・インスツルメンツ製品の選定、(2) お客様のアプリケーションの設計、検証、試験、(3) お客様のアプリケーションに該当する各種規格や、その他のあらゆる安全性、セキュリティ、規制、または他の要件への確実な適合に関する責任を、お客様のみが単独で負うものとします。

上記の各種リソースは、予告なく変更される可能性があります。これらのリソースは、リソースで説明されている テキサス・インスツルメンツ製品を使用するアプリケーションの開発の目的でのみ、 テキサス・インスツルメンツはその使用をお客様に許諾します。これらのリソースに関して、他の目的で複製することや掲載することは禁止されています。 テキサス・インスツルメンツや第三者の知的財産権のライセンスが付与されている訳ではありません。お客様は、これらのリソースを自身で使用した結果発生するあらゆる申し立て、損害、費用、損失、責任について、 テキサス・インスツルメンツおよびその代理人を完全に補償するものとし、 テキサス・インスツルメンツは一切の責任を拒否します。

テキサス・インスツルメンツの製品は、 テキサス・インスツルメンツの販売条件、または ti.com やかかる テキサス・インスツルメンツ製品の関連資料などのいずれかを通じて提供する適用可能な条項の下で提供されています。 テキサス・インスツルメンツがこれらのリソースを提供することは、適用される テキサス・インスツルメンツの保証または他の保証の放棄の拡大や変更を意味するものではありません。

お客様がいかなる追加条項または代替条項を提案した場合でも、 テキサス・インスツルメンツはそれらに異議を唱え、拒否します。

郵送先住所：Texas Instruments, Post Office Box 655303, Dallas, Texas 75265