参考資料

Application Note

TI スマートヒューズ ハイサイド スイッチにおける一般的なソフトウェア使用例

Timothy Logan

概要

テキサス・インスツルメンツ (TI) のスマート ヒューズ ハイサイドスイッチ ポート フォリオ (HCS デバイス ファミリー) は、物理的な溶断ヒューズの代替として、自動車アプリケーションにおいて柔軟に構成可能な半導体設計を提供する、汎用性と高機能を備えたデバイス群です。HCS ハイサイド スイッチ デバイス ファミリは SPI を通信として使用し、容量性充電や I2T などの各種パラメータを構成し、電流センス (内蔵 ADC を使用) や障害検出などの診断を読み出します。ソフトウェア開発を簡素化するために、TI は複数のプロセッサ プラットフォームまたはマイコン プラットフォーム (TI とそれ以外の両方) に対応するソフトウェア ドライバとサンプル コードの包括的なスイートを提供しています。このアプリケーション ノートでは、TI のスマート ヒューズ ソフトウェア エコシステムのすべての要素について説明します。トップ レベルのドライバ、構成または評価ツール、基礎となるスマート ヒューズ デバイスの一般的な使用事例を示す包括的なサンプル コードが該当します。

目次

商標

1 ソフトウェア エコシステム

TI のスマート ヒューズ エコシステムは、以下のソフトウェア コンポーネントから構成されています。

表 1-1. スマート ヒューズの付属品

資料名	概要
スマート ヒューズ コンフィギュレータ	TPSxHCxx-Q1 デバイスの設定およびソフトウェア開発用の C 設定ファイルのエクスポートに使用されるホスト GUI ツールです。このソフトウェアは、HSS-HCSMOTHERBRDEVM と、対応するドーター カードを制御する目的でも使用します。
デバイス固有の C ヘッダ ファイル	HCS デバイスのレジスタ マップを表したヘッダ ファイルです。このファイルには、デバイスのすべてのレジスタ定義および列挙体が含まれています。
HCS プラットフォーム ドライバ	低レベルの SPI ドライバ移植レイヤを備えた汎用的なドライバ セットです。各種プロセッサ/マイコン向けに実装例が提供されています。
アプリケーション コード例	テキサス・インスツルメンツの HCS ファミリのハイサイド スイッチを使用する際の一般的な機能や差別化ポイントを紹介する、共通のコード例セットです。

HCS プラットフォーム ドライバおよびアプリケーション コード例を含むソフトウェア パッケージは、対応するソフトウェアページで入手できます。ドライバおよびコード例は、柔軟な移植や再利用が可能な BSD ライセンスのオープン ソースです。ソフトウェア パッケージは、HCS-SMARTFUSE-DRIVERS にあります。

HSS-HCSMOTHERBRDEVM に関連したスマート ヒューズ コンフィギュレータ ソフトウェアの機能については、*スマート ヒューズ評価基板*ユーザー ガイドに詳細が記載されていることに注意してください。

ソフトウェア関連資料のすべての構成要素は互いに連携して動作するように設計されており、ソフトウェア開発を簡素化し、HCS ファミリのハイサイド スイッチを使い始めるためのわかりやすい手順を提供します。標準的な開発フローには、次の手順が含まれます。

1. スマート ヒューズ コンフィギュレータを使用して、初期設定を生成します。これらの設定は、ブートアップ時に、マイコンによって SPI 経由でハイサイドスイッチにロードされます。この設定には、電流制限構成、容量性充電モード、および特定のワイヤゲージ曲線に基づいて必要な特定の I2T ターンオンが含まれます。
2. 設定が完了すると、ソフトウェアは、初期設定やプログラミングに使用される汎用 C 構造を含む汎用 C ファイルをエクスポートします。
3. この構成構造体のポインタは、HCS_initializeDevice 関数 (セクション 2.4.9) に渡されます。次にドライバは、デバイスのすべての関連レジスタをプログラムします。この API は通常、マイコンのブートまたは初期化時に 1 回呼び出されます。

これらの手順のそれぞれと個々のコンポーネントについて、以降のセクションで説明します。

2 プラットフォーム ドライバ

2.1 ドライバ コンセプト

テキサス・インスツルメンツが提供する、HCS ファミリのスマート ヒューズ ハイサイド スイッチ向けドライバー式は、汎用的に設計されており、基盤となるデバイスの完全な設定および活用を可能にします。これらのドライバの特長:

- スマート フューズ コンフィギュレータ ソフトウェアからエクスポートされたファイルを使用した初期設定。
- 個別のトランザクション ヘッダの返却に対応した、汎用的なレジスタの読み書き機能です。
- ハイサイド スイッチの ADC レジスタからの生の ADC 結果を、人間が読みやすい浮動小数点値に変換するための便利な関数です。
- デバイス レベルとチャネル レベルの両方の診断ステータスを提供する機能。

ドライバと、提供されるサンプル コードとの関係を図 2-1 に示します。

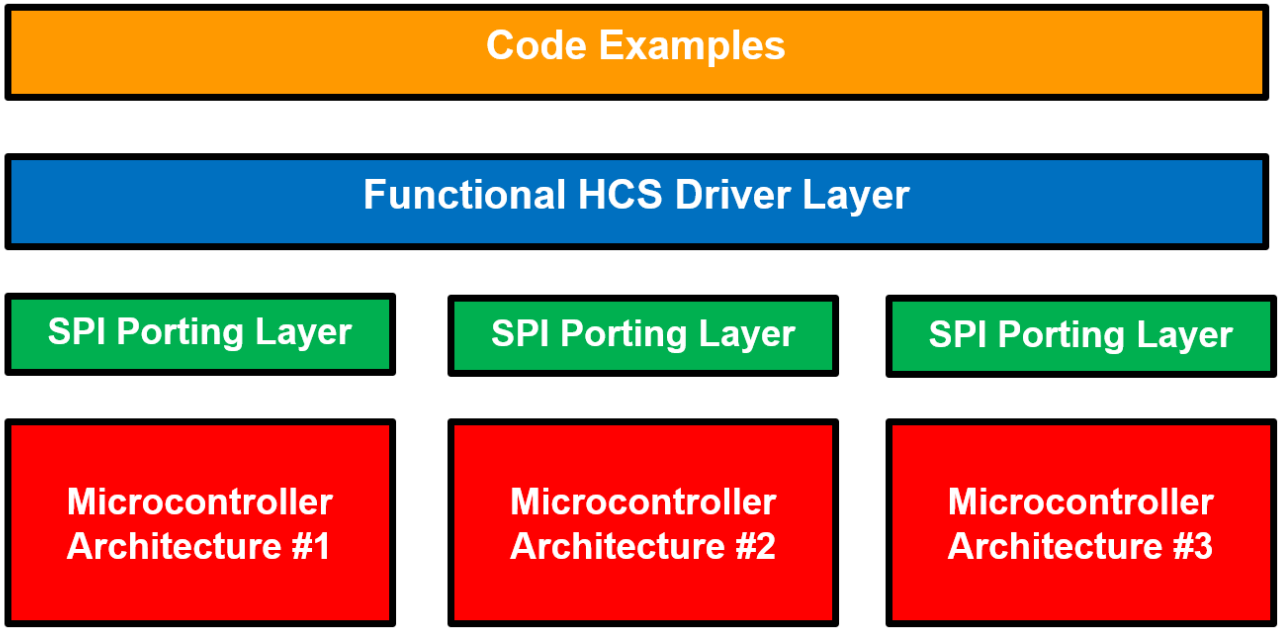

図 2-1. ゲードライバのアーキテクチャ

トップ レベルのドライバ API には、HCS ファミリのスマート ヒューズ デバイスを示すための HCS_prefix が含まれており、ソフトウェア パッケージの ***hcs_control_driver.h*** および ***hcs_control_driver.c*** で提供されています。トップ レベルのサンプル コードでは、これらの API を使用して、ハイサイド スイッチの制御および構成のための汎用機能セットを提供します。物理 SPI 通信の場合、***hcs_control_driver.h*** で一連の外部関数が宣言されています。

```
/* --------------------------- Porting Functions ---------------------------- */
/*
 * These functions need to be implemented by each individual device port. The functions
 * handle the low-level hardware specific implementation with the respective
 * architecture's specific hardware peripherals (SPI and GPIO)
 */
bool HCS_port_spiSendData(uint8_t *data, uint8_t len, uint8_t* respData);
void HCS_port_assertSPI(void);
void HCS_port_deassertSPI(void);
```

これらの関数は、個々のアーキテクチャの実装ごとに定義されており、各プラットフォームのハードウェア SPI 相互作用を処理します。これらの関数を追加のアーキテクチャに移植する方法の詳細については、「他のプラットフォームへの移植」(セクション 2.3) を参照します。機能、パラメータ、戻り値を含む API の完全なリストについては、API ガイド (セクション 2.4) を参照します。

2.2 サポート対象のプラットフォーム

スマート ヒューズ ドライバおよび設定パッケージには、さまざまなマイコンやプロセッサ向けの実装例が含まれています。追加の実装は定期的に追加されていますが、現在のアーキテクチャと関連する開発キットのリストを表 2-1 に示します。

表 2-1. サポート対象のプラットフォーム

アーキテクチャ	開発ボード	エコシステム ノート
テキサス・インスツルメンツ MSPM0G3507-Q1	LP-MSPM0G3507	Code Composer Studio Theia
STMicroelectronics STM32H723ZGT6	NUCLEO-H723ZG	STM32CubeIDE

セクション 2.3 に記載されている関連するデバイス レベルの機能を実装することで、デバイス サポートを追加できます。

HSS-HCMOTHERBRDEVM は複数の汎用 SPI ヘッダーを搭載しており、接続先のハイサイド スイッチ ドーター カードに外部 SPI 信号を*プラグインするため*に使用できます。サンプル コードと HCS プラットフォーム ドライバの開発中に、以下に示す開発ボードを HSS-HCMOTHERBRDEVM と組み合わせて使用し、検証を実施しています。この構成で使用されている LP-MSPM0G3507 評価基板の例については、図 2-2 を参照します。

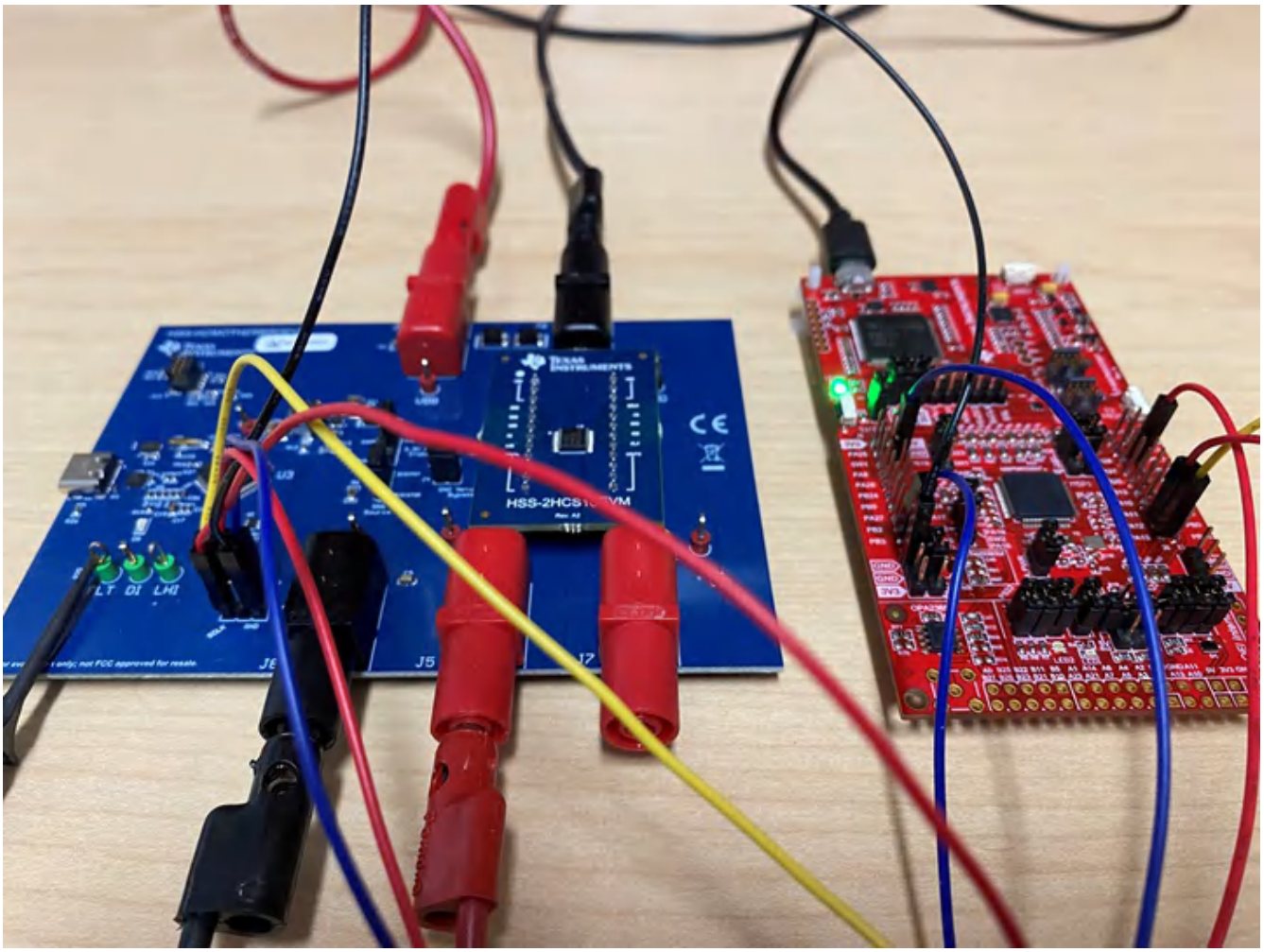

図 2-2. 接続された MSPM0

以下に、NUCLEO-H723ZG ボードを使用した接続を示します。

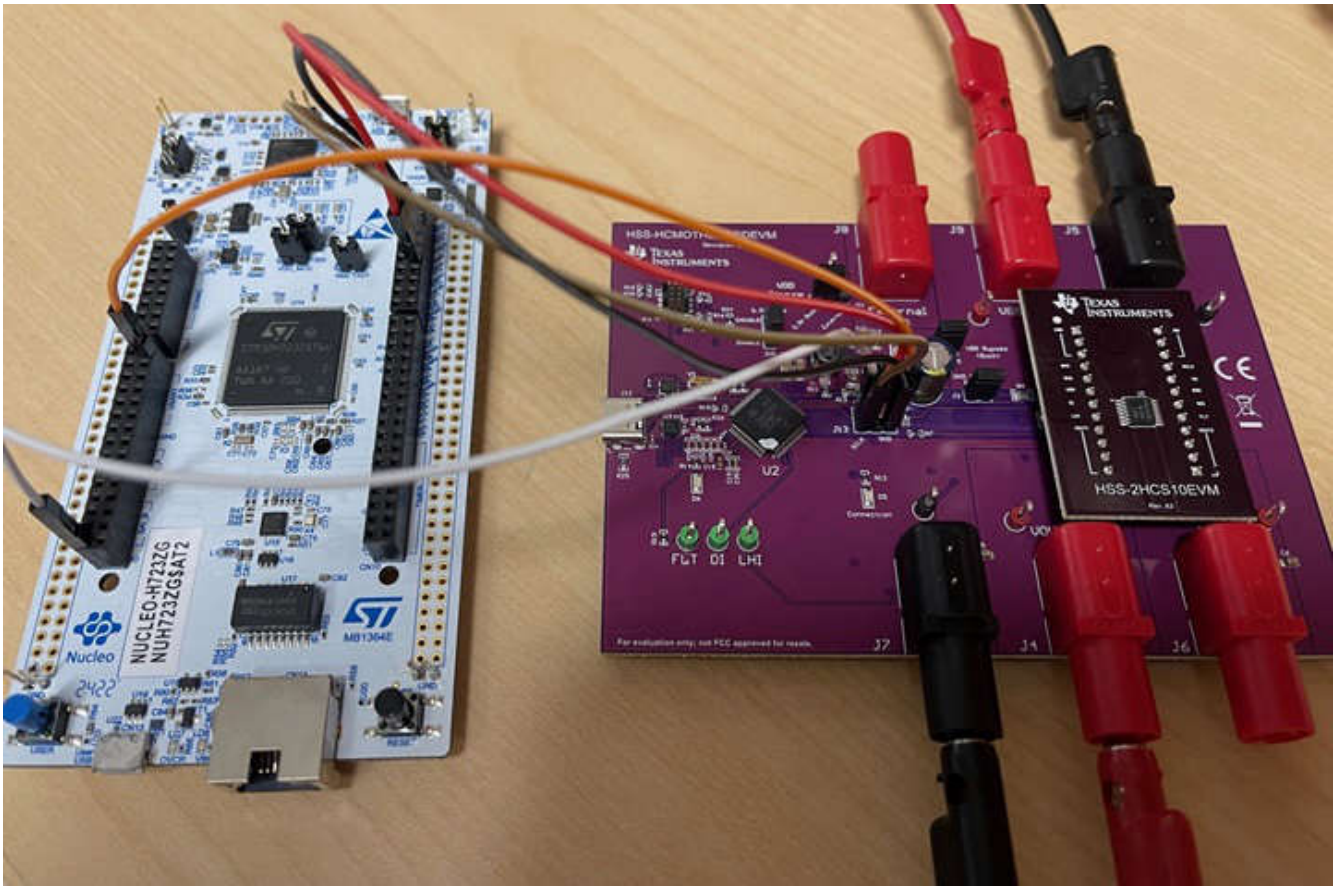

図 2-3. NUCLEO-H723ZG の接続

2.3 他のプラットフォームへの移植

さまざまなアーキテクチャ実装の例が提供されていますが、HCS プラットフォーム ドライバは、SPI と C 言語をサポートするあらゆるマイクロプロセッサやアーキテクチャに容易に移植できるように設計されています。ドライバのヘッダ ファイル (***hcs_control_driver.h***) では、外部リファレンスを使用して、次の 2 つの関数が宣言されます。

```
/* --------------------------- Porting Functions ---------------------------- */
/*
 * These functions need to be implemented by each individual device port. The functions
 * handle the low-level hardware specific implementation with the respective
 * architecture's specific hardware peripherals (SPI and GPIO)
 */
bool HCS_port_spiSendData(uint8_t *data, uint8_t len, uint8_t* respData);
void HCS_port_assertSPI(void);
void HCS_port_deassertSPI(void);
```

移植は簡単に行うことを意図していますが、各機能の説明と移植のためのガイダンスを表 2-2 に示します。

表 2-2. アーキテクチャ移植機能

機能	概要	戻り値
HCS_port_spiSendData	全二重 SPI トランザクションを実行します。 ***データ*** は送信されるデータを表し、***respData*** は受信されるデータを表します。***len*** はトランザクションの長さです。この関数は、SPI トランザクション全体が完了するまで、スリープまたはポーリングのいずれかによってブロッキング動作を行う必要があります。	トランザクションの結果の stdbool 表現。トランザクションが問題なく完了した場合は ***true*** 、それ以外の場合は ***false*** にします。
HCS_port_assertSPI	SPI バスのチップ セレクト (CS) ラインを Low に設定します。これは、シングルチェーン トランザクションとデイジーチェーン トランザクションの両方で使用されるだけでなく、上記の HCS_port_spiSendData 関数で使用されます。	なし
HCS_port_deassertSPI	SPI バスのチップ セレクト (CS) ラインを High に設定します。これは、シングルチェーン トランザクションとデイジーチェーン トランザクションの両方で使用されるだけでなく、上記の HCS_port_spiSendData 関数で使用されます。	なし

2.4 API ガイド

参考資料

- tHCSResponseCode Union (セクション 2.4.1)
- HCS_convertCurrent (セクション 2.4.2)
- HCS_convertTemperature (セクション 2.4.3)
- HCS_convertVoltage (セクション 2.4.4)
- HCS_getChannelFaultStatus (セクション 2.4.5)
- HCS_getDeviceFaultSatus (セクション 2.4.6)
- HCS_gotoLPM (セクション 2.4.7)
- HCS_gotoSleep (セクション 2.4.8)
- HCS_initializeDevice (セクション 2.4.9)
- HCS_readRegister (セクション 2.4.10)
- HCS_setSwitchState (セクション 2.4.11)
- HCS_updateConfig (セクション 2.4.12)
- HCS_wakeupDevice (セクション 2.4.13)
- HCS_writeRegister (セクション 2.4.14)

2.4.1 tHCSResponseCode Union リファレンス

データ フィールド

```
typedef union
{
    uint8_t byte;
    struct
    {
        unsigned I2T_FLT : 1;
        unsigned LPM_FLT : 1;
        unsigned CHAN_TSD : 1;
        unsigned ILIMIT_FLT : 1;
        unsigned SHRT_VBB_FLT : 1;
        unsigned OL_FLT : 1;
        unsigned SUPPLY_FLT : 1;
        unsigned GLOBAL_ERR_WRN : 1;
    } bits;
} tHCSResponseCode;
```

2.4.2 float_t HCS_convertCurrent (uint16_t rawValue、uint16_t ksnsVal、uint16_t snsRes)

ADC の未処理の電流値を、読み取り可能な浮動小数点値に変換します。

これは、ADC 結果レジスタのいずれかから読み取った未処理の値を受け取り、それを人間が読みやすい浮動小数点値に変換するための便利な関数です。

表 2-3. パラメータ

in	*rawValue*	変換する未加工の値
in	*ksnsVal*	データ シートからの KSNS 定数
in	*snsRes*	SNS ピンの抵抗値

表 2-4. 戻り値

returnCode	電流の浮動小数点表現

2.4.3 float_t HCS_convertTemperature (uint16_t rawValue)

ADC 温度の未処理の値を、読み取り可能な浮動小数点値に変換します。

これは、ADC 結果レジスタのいずれかから読み取った未処理の値を受け取り、人間が読みやすい浮動小数点値に変換するための便利な関数です。

表 2-5. パラメータ

in	*rawValue*	変換する未加工の値

表 2-6. 戻り値

returnCode	温度の浮動小数点表現

2.4.4 float_t HCS_convertVoltage (uint16_t rawValue)

未処理の ADC 電圧値を読み取り用の浮動小数点値に変換します。

これは、ADC の結果値のいずれかから読み取った未処理の値を受け取り、それを人間が読みやすい浮動小数点値に変換するための便利な関数です。

表 2-7. パラメータ

in	*rawValue*	変換する未加工の値

表 2-8. 戻り値

returnCode	電圧の浮動小数点表現

表 2-9. 戻り値

returnCode	**tHCSResponseCode** のインスタンス

2.4.5 tHCSResponseCode HCS_getChannelFaultStatus (uint8_t chanNum、uint16_t * fltStatus)

各チャネルの故障ステータスを読み取ります。

個々のチャネルの故障ステータスを読み取り、そのステータスを fltStatus に保存します

表 2-10. パラメータ

in	*chanNum*	読み取るチャネル番号
out	*fltStatus*	ここで故障ステータスをメモリすることができます

2.4.6 tHCSResponseCode HCS_getDeviceFaultSatus (uint16_t * fltStatus)

デバイスの故障ステータスを読み取ります。

デバイスの故障ステータスを読み取り、そのステータスを fltStatus に保存します

表 2-11. パラメータ

out	*fltStatus*	ここで故障ステータスをメモリすることができます

表 2-12. 戻り値

returnCode	**tHCSResponseCode** のインスタンス

2.4.7 tHCSResponseCode HCS_gotoLPM (tps2hcsxx_man_lpm_exit_curr_ch1_mask_t ch1ExitCurrent、tps2hcsxx_man_lpm_exit_curr_ch2_mask_t ch2ExitCurrent、uint16_t existingValue)

この例では、デバイスを LPM モードにします。

この機能は、コマンドを送信して LPM モードに移行できます。本デバイスが終了電流レベルを超えた場合、MCU は FLT ピンの立ち下がりエッジを監視する役割を果たします。または、HCS_wakeupDevice 関数を使用してデバイスをウェイクアップすることもできます。

表 2-13. パラメータ

in	*ch1ExitCurrent*	チャネル 1 の終了電流
in	*ch2ExitCurrent*	チャネル 2 の終了電流
in	*existingValue*	LPM レジスタの存在する値

2.4.8 tHCSResponseCode HCS_gotoSleep (void)

この例では、デバイスをスリープ モードにします。

この機能は、コマンドを送信して SLEEP モードに移行できます。スリープ モードから復帰した後、デバイスのレジスタ内のすべての値がリセットされる可能性があることに注意します。このデバイスは、HCS_wakeupDevice 関数によってウェークアップする必要があります。

表 2-14. 戻り値

returnCode	**tHCSResponseCode** のインスタンス

2.4.9 tHCSResponseCode HCS_initializeDevice (TPS2HCSXXQ1_CONFIG * config)

デバイス構成構造体でデバイスを初期化します。

この機能は、スマート ヒューズ コンフィギュレータ GUI ソフトウェアから生成された構成構造を取得し、接続されているハイサイド スイッチにすべての値を送信できます。ここで一般的な手法は、MCU の初期ブート時に、またはすべてのレジスタ構成が失われてデバイスがスリープ モードからウェイクアップするときにこの関数を呼び出すことです。

表 2-15. パラメータ

in	*config*	コンフィギュレーション構造体へのポインタ

表 2-16. 戻り値

returnCode	**tHCSResponseCode** のインスタンス

2.4.10 tHCSResponseCode HCS_readRegister (uint8_t addr、uint16_t * readValue)

指定されたレジスタ アドレスについて、未処理のレジスタの読み出しを実行します。

この関数は、指定されたアドレスから単純なレジスタ読み取りを実行し、指定されたポインタにレジスタの内容を格納します。

表 2-17. パラメータ

in	*addr*	読み取るレジスタ アドレス

表 2-17. パラメータ (続き)

out	*payload*	このパラメータに設定されたレジスタの値

表 2-18. 戻り値

returnCode	**tHCSResponseCode** のインスタンス

2.4.11 tHCSResponseCode HCS_setSwitchState (uint8_t swState)

チャネルごとにオン/オフ スイッチの状態を設定します。

これにより、ハイサイド スイッチのチャネルをオンまたはオフにできます。パラメータは、有効にするチャネルをビット単位で表します。

表 2-19. パラメータ

in	*swState*	コンフィギュレーション構造体へのポインタ

表 2-20. 戻り値

returnCode	**tHCSResponseCode** のインスタンス

*2.4.12 tHCSResponseCode HCS_updateConfig (TPS2HCS10Q1_CONFIG * config)*

提供された設定をスマート ヒューズ設定で更新します。

この API は、ハイサイドスイッチから取得した値で、指定された設定構造体を埋めることができます。これは、ホスト ソフトウェアがレジスタ内の設定を変更し、その内容を初期設定構造体に反映させたい場合に使用できます。

表 2-21. パラメータ

out	*config*	コンフィギュレーション構造体へのポインタ

表 2-22. 戻り値

returnCode	**tHCSResponseCode** のインスタンス

2.4.13 tHCSResponseCode HCS_wakeupDevice (void)

デバイスをスリープ モードからウェイクアップさせます。

この API は、スリープ モードからデバイスをウェイクアップさせるために、存在しないレジスタ アドレス 0xFF にダミーの書き込みを行うだけのシンプルな処理です。デバイスは CS ピンの遷移によってスリープ モードからウェイクアップするため、ダミーの書き込みによってその遷移が発生し、デバイスがウェイクアップします。

表 2-23. 戻り値

returnCode	**tHCSResponseCode** のインスタンス

2.4.14 tHCSResponseCode HCS_writeRegister (uint8_t addr、uint16_t payload)

指定されたレジスタ アドレスについて未処理のレジスタへの書き込みを実行します。

この関数は、指定されたアドレスに対して、与えられたペイロードを用いて単純なレジスタ書き込みを行います

表 2-24. パラメータ

in	*addr*	書き込むレジスタ アドレス
in	*payload*	addr に書き込む値

表 2-25. 戻り値

returnCode	**tHCSResponseCode** のインスタンス

3 構成および評価ツール

スマート ヒューズ コンフィギュレータ ツールは、HSS-HCMOTHERBRDEVM と併用できるソフトウェア ホスト ツールであり、HCS ハイサイド スイッチのライブ設定や、電流検出や故障状態などの診断情報の読み出しに使用できます。また、バージョン 1.9.4 以降では、物理的な EVM ボードを使用せずに*構成モード*に移行できます。構成モードでは、ユーザーは電流制限、容量性充電モード、診断レポートなど、デバイスのさまざまな設定を変更できます。また、ユーザーは I2T チューナーを使用して、置き換える溶断ヒューズの配線プロファイルおよび性能に合うようにデバイスを設定できます。構成モードに入るには、図 3-1 で示されているには、*ヘルプ* -> *デモ/構成モード*を選択します。

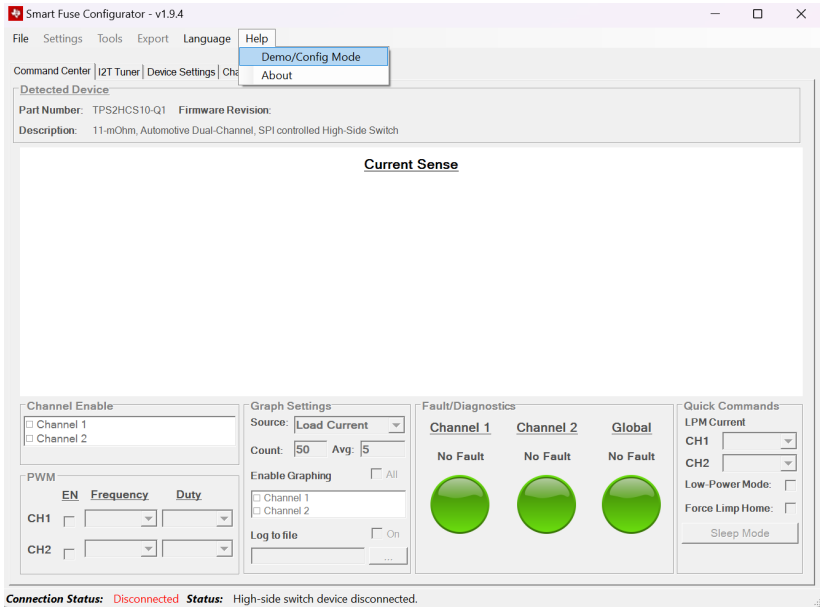

図 3-1. 構成モード

デモ モードに入ったら、ユーザーは『Smart Fuse 評価モジュール ユーザー ガイド』に記載された方法でソフトウェアを使用できます。なお、このモードでは EVM との実際の通信は行われず、GUI に表示される診断情報も実際の状態を反映したものではありません。デモ モードは *ヘルプ* -> *デモ/構成モード*を選択するか、デバイスに EVM を接続します。

アプリケーションのニーズを満たすようにデバイスを構成したら、*エクスポート* -> *構成ファイル*を選択して構成をエクスポートできます。

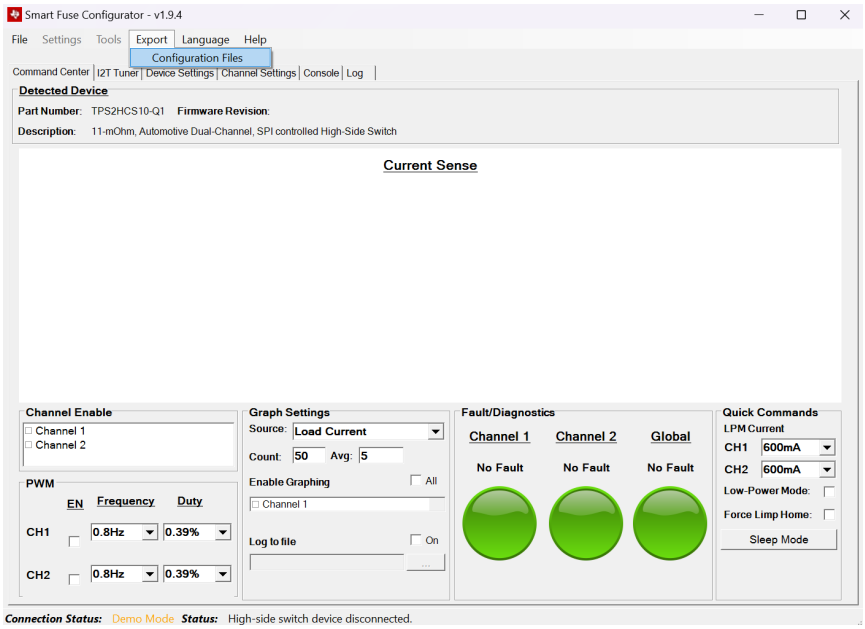

図 3-2. 構成のエクスポート

このダイアログからエクスポートされるファイルは、提供されているコード例 (デフォルトは ***tps2hcsxx_config.h*** および ***tps2hcsxx_config.c***) に同梱されている構成ファイルと同じです。手軽な導入方法として、設定ファイルを空の例 (セクション 4.1) にエクスポートし、スマートヒューズ開発を開始するための空のプログラムで使用されるデフォルト ファイルを上書きできます。

エクスポートされた構成ファイルは、プロジェクト ページで提供されるデバイス固有のヘッダー ファイルによって異なることに注意してください。このヘッダー ファイルには、特定のハイサイド スイッチの型番に関連するレジスタ定義と列挙情報がすべて含まれています。

エクスポートされたファイルには、デバイスのすべての構成レジスタに対応するレジスタ定義が含まれています。

```
typedef struct TPS2HCSXXQ1_CONFIG
{
    TPS2HCSXX_CRC_CONFIG_OBJ        crcConfig;
    TPS2HCSXX_LPM_OBJ               lpmConfig;
    TPS2HCSXX_FAULT_MASK_OBJ        faultMaskConfig;
    TPS2HCSXX_SW_STATE_OBJ          swState;
    TPS2HCSXX_DEV_CONFIG_OBJ        devConfig;
    TPS2HCSXX_ADC_CONFIG_OBJ        adcConfig;
    TPS2HCSXX_PWM_CH1_OBJ           pwmCh1Config;
    TPS2HCSXX_ILIM_CONFIG_CH1_OBJ   ilimCh1Config;
    TPS2HCSXX_CH1_CONFIG_OBJ        diagConfigCh1;
    TPS2HCSXX_I2T_CONFIG_CH1_OBJ    i2tConfigCh1;
    TPS2HCSXX_PWM_CH2_OBJ           pwmCh2Config;
    TPS2HCSXX_ILIM_CONFIG_CH2_OBJ   ilimCh2Config;
    TPS2HCSXX_CH2_CONFIG_OBJ        diagConfigCh2;
    TPS2HCSXX_I2T_CONFIG_CH2_OBJ    i2tConfigCh2;
 } TPS2HCSXXQ1_CONFIG;
```

この関数へのポインタは、プラットフォーム ドライバの HCS_initializeDevice 関数に渡され (通常はマイコンの起動時)、ハイサイド スイッチの初期設定を行うために使用されます。構造体の定義の後に、その構造体のインスタンスが宣言され、スマート ヒューズ コンフィギュレータ ツールで設定されたすべての値を表す値が格納されます。ユーザーはこのインスタンスを出発点として利用し、変更が必要な場合はコード上で手動で更新するか、スマート ヒューズ コンフィギュレータ プログラムから設定ファイルを再生成することもできます。

4 サンプル コード

スマート ヒューズ ソフトウェア パッケージの一部である一連のサンプル コードが公開されており、スマート ヒューズ デバイス ファミリの機能または差異化と、HCS プラットフォーム ドライバの使いやすさの両方を強調しています。表 4-1 に、利用可能なサンプル コードの概要を示します。

表 4-1. スマート ヒューズのコード例

コード例の名前	概要
空	スマート コード例からエクスポートされた構造体を使用して、ハイサイド スイッチを初期化するシンプルなコード例です。
I2T トリップ	I2T 機能と、I2T 故障を検出/回復する方法を示します
低電力モード	デバイスを低消費電力モードに移行させ、ウェイクアップ イベントを待機します
電流センス	HCS ファミリの高精度な電流検出機能の使い方を示し、低電流および高電流の両方に対して読み取りを最適化する方法を紹介します

4.1 空の例

空のコード例は、スマート ヒューズ アプリケーションの出発点として利用できる、シンプルなコード サンプルです。このコード例が行うのは、起動後に基盤となる SPI ペリフェラルを設定し、その後の制御をユーザーに引き渡すことだけです。初期構成の主要部分を次のように示します。

```
    /* Configuring the device initially */
    HCS_wakeupDevice();
    HCS_initializeDevice(&exportConfig);
```

HCS_wakeupDevice 関数は、デバイスにダミー書き込みを発行するだけです。リセット終了後、HCS ファミリはスリープ モードになります。ウェイクアップ関数は、存在しないレジスタ 0xFF に書き込みを行うことで、デバイスがスリープ状態から抜け出していることを確認します。

HCS_initializeDevice 関数は、スマートフューズ コンフィギュレータ アプリケーションからエクスポートされた設定ファイルを受け取り、それをハイサイド スイッチに読み込ませます。この関数は通常、MCU のブートアップ時にハイサイド スイッチを初期化するたびに呼び出されます。

4.2 I2T トリップの例

I2T トリップコードのサンプル コードは、システムで I2T イベントが発生したときにハイサイド スイッチのイベントを処理する方法を示します。このコード例では、デバイスが I2T フォルト用にラッチ モードに設定されていることを想定し、I2T イベントが発生した後でデバイスを*リセット*するための正しいイベント シーケンスを示しています。自動再試行モード (I2T_CONFIG_CHx レジスタの TCLDN_CHx がタイムアウトに設定されている場合) では、デバイスは、チャネルを再度イネーブルする前に、適切な時間だけ自動的に待機します。

なお、このコード例では FAULT ピンが立ち下がりエッジ割り込みとして使用されるように設定されています。I2T トリップが発生すると、オープン ドレイン構成の FAULT ピンが Low に引き下げられ、マイコンに割り込みがかかります。その後、マイコン上のソフトウェアがデバイスの動作を再開させ、必要な対策を実行してデバイスを再度有効にすることができます。ラッチ モードでは、I2T トリップ イベントの発生後に適切な手順を以下に示します。

1. 影響を受けるチャネルを無効にする
2. I2T_CONFIG_CHx レジスタの TCLDN_CHx を適切なクールダウン時間に設定します
3. マイコンをスリープさせて、指定された時間だけ待機する
4. I2T_CONFIG_CHx レジスタの TCLDN_CHx をラッチモードに戻す
5. チャネルを再度イネーブルにする

次に、関連するコードの一部を示します。

```
        if(currentValue & TPS2HCSXX_FLT_STAT_CH1_I2T_FLT_CH1_MASK)
        {
            /* Disabling the channel */
            HCS_setSwitchState(0);

            /* Setting the device to 2s retry state */
            exportConfig.i2tConfigCh1.value.bits.TCLDN_CH1 =
                (tcldn_ch1_en_4p0s_mask);
            HCS_writeRegister(TPS2HCSXX_I2T_CONFIG_CH1_REG,
                        exportConfig.i2tConfigCh1.value.word);

            /* Waiting for two milliseconds. At this point, normally
                yield the tasks if in an RTOS, but wait for
                an interrupt here.  */

            DL_TimerA_startCounter(TIMER_0_INST);
            while(timerTriggered == false)
            {
                __WFI();
            }
            timerTriggered = false;

            /* Setting back to latch mode */
            exportConfig.i2tConfigCh1.value.bits.TCLDN_CH1 = 0;
            HCS_writeRegister(TPS2HCSXX_I2T_CONFIG_CH1_REG,
                        exportConfig.i2tConfigCh1.value.word);

            /* Re-enabling the channel */
            HCS_setSwitchState(1);
        }
```

4.3 低消費電力モードの例

低消費電力モードのコード例では、HCS ハイサイド スイッチを低電力モードに設定し、故障イベントでウェイクアップする方法を示しています。このコード例を使用するには、マイコンにコードを書き込む前に、チャネル 1 に 800mA 未満の電流を消費する負荷を接続してください (これは HCS_gotoSleep 関数で設定された LPM 解除電流です)。サンプル コードをダウンロードしたら、負荷電流を 800mA より大きくします。これにより、LPM が解除され、FAULT ピンがトリガーされて、マイコンが割り込みまたはイベント処理を行う可能性があります。関連するアプリケーション コードを以下に示します。

```
    while(1)
    {
        /* Putting the device into LPM. 800mA exit current on CH1 */
        HCS_gotoLPM(man_lpm_exit_curr_ch1_en_800mA_mask,
                    man_lpm_exit_curr_ch2_en_800mA_mask,
                    0);

        /* Wait for the fault line to trigger low on PB3 */
        __WFI();

        /* If waking up from the interrupt, then check to make sure a signal was
            for an LPM wakeup. The idea here is that the user increases the
            load current somehow to "force the device" from LPM. */
        resCode = HCS_readRegister(TPS2HCSXX_FLT_STAT_CH1_REG,
                                   &currentValue);
        resCode.byte |=  HCS_readRegister(TPS2HCSXX_FLT_STAT_CH1_REG,
                                   &currentValue).byte;

        if(currentValue & TPS2HCSXX_FLT_STAT_CH1_LPM_WAKE_CH1_MASK)
        {
            /* Set a breakpoint here for demonstration */
            asm ("nop");
        }
    }
```

4.4 電流センスの例

電流検出のコード例では、HCS ファミリのスマート ヒューズ ハイサイド スイッチが、スケーラブルで非常に柔軟な電流センス機能を備えていることを示しています。このコード例では、1ms のタイマーを設定し、一定間隔で起動してハイサイド スイッチのチャネル 1 の負荷電流をサンプリングします。負荷電流が 500mA 未満の場合、デバイスはハイサイド スイッチに対して以下の設定を有効にします。

- 入力電圧スケーリング (CHx_CONFIG レジスタの ISNS_SCALE_CHx) が有効になり、ADC の入力電圧が 8 倍にスケーリングされます
- 開放負荷検出 (CHx_CONFIG の OL_ON_EN_CHx) が有効になり、KSNS 比が低い値に変更されます (詳細はデータシートの電気的仕様を参照)

各イベントごとにブレークポイントが設定されており、ユーザーが停止して電流検出の動作を理解できるようになっています。ソフトウェアで低電流検出モードに移行すると、状態変数 (inLowCurrent) が設定され、デバイスは引き続き電流を定期的にサンプリングします。電流レベルが ADC の読み取り値を飽和させた場合、低電流モードは終了し、通常のスケーリングまたは KSNS モードが使用されます。関連するコードのスニペットは、以下のコードに示されています。

```
 while(1)
{
    __WFI();

    /* Reading the load current value. Read the register twice
        as the */
    resCode = HCS_readRegister(TPS2HCSXX_ADC_RESULT_CH1_I_REG,
                               &currentValue);
    resCode.byte |=  HCS_readRegister(TPS2HCSXX_ADC_RESULT_CH1_I_REG,
                               &currentValue).byte;

    /* For each transaction, the high-side switch returns an error
        status code that reports common faults of the device. */
    if(resCode.byte != 0)
    {
        handleError(resCode);
    }

    /* Masking out the relevant bits */
    currentValue &= TPS2HCSXX_ADC_RESULT_CH1_I_ADC_RESULT_CH1_I_MASK;

    /* Check to see if the threshold is below where to turn
        on current sense scaling and change the KSNS ratio and enable
        scaling by 8x. This allows for  */
    if((currentValue < LOW_CURRENT_SNS_THRESHOLD) &&
            (inLowCurrent == false))
    {
        exportConfig.diagConfigCh1.value.bits.OL_ON_EN_CH1 = 1;
        exportConfig.diagConfigCh1.value.bits.ISNS_SCALE_CH1 = 1;
        inLowCurrent = true;
        HCS_writeRegister(TPS2HCSXX_CH1_CONFIG_REG,
                          exportConfig.diagConfigCh1.value.word);

        /* Adding place to set a breakpoint for the sake of demonstration */
        asm ("nop");
    }
    else if((currentValue == LOW_CURRENT_SNS_SATURATION) &&
                (inLowCurrent == true))
    {
        exportConfig.diagConfigCh1.value.bits.OL_ON_EN_CH1 = 0;
        exportConfig.diagConfigCh1.value.bits.ISNS_SCALE_CH1 = 0;
        HCS_writeRegister(TPS2HCSXX_CH1_CONFIG_REG,
                          exportConfig.diagConfigCh1.value.word);

        /* Adding place to set a breakpoint for the sake of demonstration */
        asm ("nop");
    }
}
```

低電流を検出し、スケーリング モードや KSNS モードを有効にすることで、ハイサイド センスは電流検出の分解能を適応的に調整でき、高精度な電流検出が求められるアプリケーションにも対応可能になります。

5 まとめ

HCS ファミリのスマートヒューズ デバイス向けに提供されているソフトウェア ドライバおよびコード例は、自動車用スマートヒューズ アプリケーションのソフトウェア開発において、汎用性の高いさまざまな機能を提供します。これらのドライバは、さまざまなホスト アーキテクチャへの対応が容易であり、シンプルな設計により、開発者は最小限のバックエンド作業で HCS デバイス向けソフトウェア開発をすぐに始めることができます。さらに、スマート ヒューズ コンフィギュレータ ソフトウェア ツールを使用することで、ユーザーはスマート ヒューズ デバイスをシームレスに構成し、最終アプリケーションの正確な要件に合わせてドライバを調整できます。これらのソフトウェア ツールは、HCS デバイス ファミリをスマートヒューズやゾーン アプリケーションに導入する際の障壁を取り除き、ソフトウェア開発とハードウェア開発を柔軟に組み合わせるためのアプローチを提供します。

6 参考資料

- テキサス インスツルメンツ、*TPS2HCS10-Q1 自動車向けのデュアルチャネル 10mΩ スマート ハイサイド スイッチで、I²T ワイヤ保護、低消費電流モード、SPI 通信機能付き*
- テキサス インスツルメンツ、*HCS-SMARTFUSE-DRIVERS 、HCS スマート ヒューズ デバイス向けのシンプルな C ドライバおよびコード例*
- テキサス インスツルメンツ、*HSS-HCMOTHERBRDEVM スマート ヒューズ評価基板*
- テキサス インスツルメンツ、*HSS-2HCS10EVM TPS2HCS10-Q1 スマート ヒューズ ハイサイド スイッチ用のドーターカード*
- テキサス インスツルメンツ、*HSS-HCMOTHERBRDEVM および TI のスマート ヒューズ ハイサイド スイッチ用 HSS-SMART-CONFIGURATOR 構成ツール*
- テキサス インスツルメンツ、*HCS-HEADER-FILES レジスタ定義付きスマート ヒューズ ハイサイド スイッチ用 C ヘッダ ファイル*

7 改訂履歴

資料番号末尾の英字は改訂を表しています。その改訂履歴は英語版に準じています。

Changes from Revision A (July 2024) to Revision B (April 2025)	Page
• ドキュメント全体で部品番号およびコード例を更新	1

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

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