参考資料

Application Note

組み込みコントローラ(EC)を使用して TPS25751 または TPS26750 にパッチバンドルを直接読み込む

Chris Sterzik, Roy Chou

概要

TPS25751 および TPS26750 は、電力供給をサポートするアプリケーション向けに最適化された、構成可能な USB Type-C ®および Power Delivery（PD）コントローラです。一般に、これらの PD コントローラは、ブート時に I2Cc を使用して外部 EEPROM からバイナリイメージを読み込みます。組み込みコントローラ（EC）が利用可能な場合は、EEPROM の代わりに EC を使用して、I2Ct インターフェイスを経由してそのイメージを PD に読み込むことができます。テクニカルリファレンスマニュアル（1 および 2）は、I2Ct バス経由で 複数の PD コントローラに同時に画像を書き込む手順を示しています。このアプリケーションノートでは、このフローについて詳しく説明し、1 つの PD コントローラへの画像書き込みに重点を置いています。I2Ct 経由で I2C コマンドを使用して、PTCH モードから APP モードへの遷移をステップバイステップで解説するほか、サンプルコードも提供しています。

目次

商標

1 はじめに

一部のアプリケーションでは、アプリケーション実行時（APP モード）に PD コントローラと連携するために、組み込みコントローラ（EC）が必要となる場合や、システム内でハウスキーピング処理を担う EC が搭載されている場合があります。コスト削減のため、EEPROM を削除し、バイナリイメージによる PD の保存と更新の機能を EC に移すことができます。このアプリケーションノートでは、組み込みコントローラ（EC）を使用してイメージを読み込み、4 文字コード（4CC）コマンドを用いて PD コントローラを PTCH モードから APP モードへ移行させる手順に焦点を当てています。EC について説明する前に、EEPROM を使用しないアプリケーションの場合の、デッドバッテリ設定について簡単に説明します。

2 ADCINX 設定

ADCIN1 入力ピンは PD コントローラのターゲットアドレスを決定し、ADCIN2 入力ピンはデッドバッテリ構成を定義します。詳細な説明は、デバイスのデータシートに記載されている 4 および 3 を参照してください。「デッドバッテリ」という用語は、PC ノートブックアプリケーションに由来し、バッテリが完全に放電した状態を指します。この状態では、PD コントローラが VBUS からデッドバッテリを充電できるよう、インタラクション（または電力）なしでパワーパスを有効化することが求められます。PD コントローラは 2 つの異なるデッドバッテリ設定を提供しますが、[1]：AlwaysEnableSink および SafeMode。SafeMode は USB Type-C シンク機能を提供しますが、シンクパスは有効化されません。SafeMode では、バイナリイメージが読み込まれるまで、PD ステートマシンは実質的に無効化されます。AlwaysEnableSink USB Type-C 接続が行われると、必ずパワーパスが有効化されますが（ポートパートナーはソースである必要があります）。USB PD 機能はその設定が読み込まれるまでディセーブル（無効）のままです。

このアプリケーションノートに記載されているステップは、デッドバッテリの構成とは無関係です。EC によるデッドバッテリフラグの処理とシステムの動作は、このドキュメントの範囲外です。

[1] ここでは NegotiateHighVoltage についての説明は省略します。

3 固有アドレス インターフェイスプロトコル

Patch Burst Mode（PBM）機能は、SMBUS プロトコルと単純な I2C 書き込みの両方を使用します。SMBUS プロトコルはデータシートに記載されており（3 および 4 を参照）、すべてのレジスタアクセスに適用できます。PBM で使用されるレジスタ書き込みおよび読み取りの例を、SMBUS レジスタ書き込みの例と SMBUS レジスタ読み取り例に示します。SMBUS プロトコルで PBMs コマンドを発行した後、I2C 書き込みは PBMs コマンドで設定されたターゲットアドレスに対して行われます。I2C パッチバーストモード書き込みに、画像を PD コントローラに送信するためのシンプルな I2C 書き込みを示します。

表 3-1. SMBUS レジスタ書き込みの例

タイプ	ACK	アドレス	読み取り	データ	説明
start					
アドレス	TRUE	0x21	FALSE		レジスタアクセス用の PD I2C アドレス
data	TRUE			0x08	レジスタ・アドレス
data	TRUE			0x04	（ターゲットに送信した）バイト数
data	TRUE			0x50	P
data	TRUE			0x42	B
data	TRUE			0x4D	M
data	TRUE			0x73	S
ストップ					

表 3-2. SMBUS レジスタ読み取り例

タイプ	ACK	アドレス	読み取り	データ	説明
start					
アドレス	TRUE	0x21	FALSE		レジスタアクセス用の PD I2C アドレス
data	TRUE			0x09	レジスタ番号
start					リピートスタートで書き込みから読取りみに変更
アドレス	TRUE	0x21	TRUE		レジスタアクセス用の PD I2C アドレス
data	TRUE			0x40	バイト数 [2]
data	TRUE			0x00	
data	TRUE			0x00	
data	TRUE			0x00	
data	TRUE			0x00	
	TRUE			0x30	PBM アドレス
	FALSE			0x31	タイムアウト。コントローラは、最後のバイトの読み取りに対して NACK を返します。
ストップ					

[2] コントローラは、すべてのバイトを読み取るか、一部のバイトのみを選択して読み取ることができます。この例では、コントローラは 6 バイトだけを受信し、最後のバイトをナッキング（拒否）しています。DATA1 レジスタ（0x09）の最終読み取りでは、コントローラは全 0x40 バイトを読み取ります。

表 3-3. I2C パッチバーストモード書き込み

タイプ	ACK	アドレス	読み取り	データ	説明
start					
アドレス	TRUE	0x30	FALSE		PBM I2C アドレス
data	TRUE			0x01	画像バイト 0
data	TRUE			0x00	画像バイト 1
data	TRUE			0xE0	画像バイト 2
data	TRUE			0xAC	画像バイト 3
バイト 4 ～ 4093					
data	TRUE				画像バイト 4094
停止 [3]					
start					
アドレス	TRUE	0x30	FALSE		PBM I2C アドレス
data					画像バイト 4095
バイト 4096 ～ 11390					
data	TRUE			0x00	画像バイト 11391
ストップ					

[3] EC では、送信サイズは 4095 バ イトまでに制限されます。PD コントローラは PBM アドレスポインタを自動インクリメントしますが、I2C のスタートまたはストップではリセットされません。PBM ポインタは、PBMs コマンドを発行することでリセットできます。

4 PTCH モードから APP モードへの切り替え

パッチバンドルはバイナリイメージですが、これにはデバイス構成とファームウェア更新（パッチ）の両方が含まれています（バンドル）。セクション 4.2 に、パッチバンドル（下位領域バイナリとも呼ばれる）を生成する方法を示します。セクション 4.1 に、パッチバーストモード（PBM）プロセスと、その結果として *PTCH* モードから *APP* モードに移行する手順を示します。PBM の一般的な説明は PD コントローラの TRM に記載されており、本例でも、割り込みの使用を除いてその内容に準拠しています。このステップには、PBM プロセス中に割り込みを設定し、維持する方法が含まれます。

I2Ct バスを通じたパッチバンドルのプッシュのフローチャートは、EEPROM が存在せず、PD コントローラの起動がコールドブートまたはハードリセットであることを前提としています。

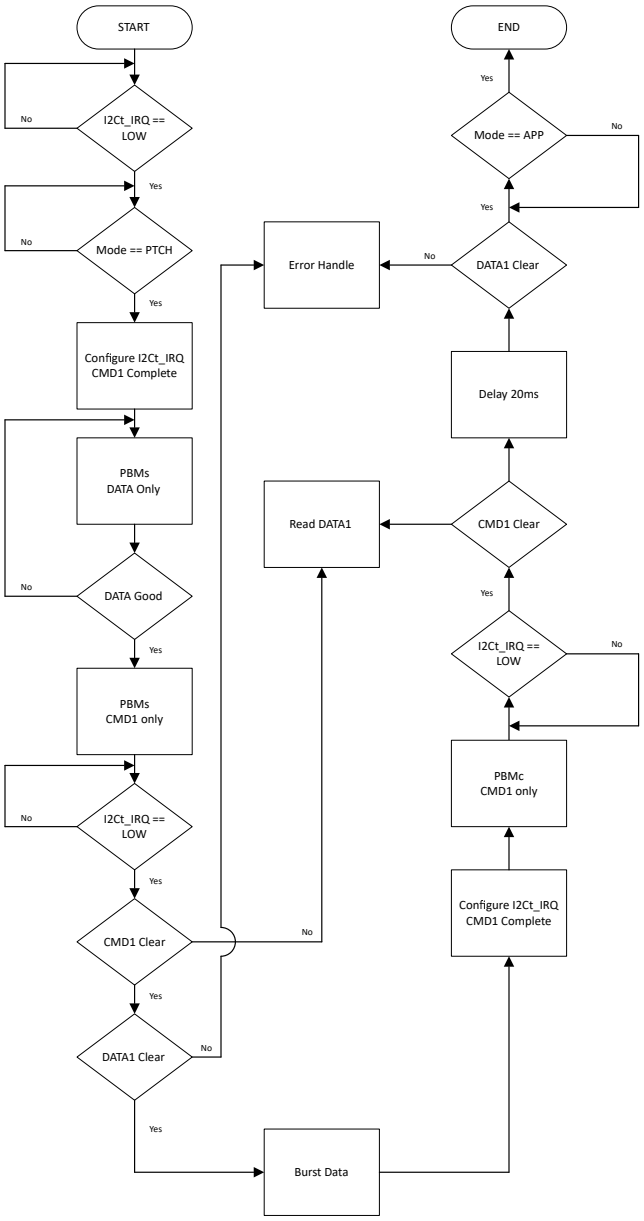

図 4-1. I2Ct バスを通じたパッチバンドルのプッシュ

4.1 PTCH モードから APP モードへのステップ

1. **I2Ct_IRQ == Low:**

   コールドブート（パワーサイクルまたは GAID ）時に、PD コントローラは *PTCH* モードに移行し、*パッチ準備完了[81]* 割り込みのみが自動的に有効になります。割り込みレジスタは、*PTCH* モード中に更新できます。この PBM の実装では、「*パッチ準備完了*」および「CMD1 完了割り込み」を使用します。レジスタをポーリングする代わりに割り込みを利用する主な理由は、PD コントローラの CPU ロードをコマンドに関連するアクティビティだけに減らためすです。

   *パッチ準備完了*割り込みは、PBM プロセスの開始時に使用され、PD コントローラの準備完了を示します。CMD1 完了割り込みは、PBMs および PBMc コマンドが完了したことを EC に通知するために使用されます。この例では、パッチがロードされた後、PD コントローラがアプリケーションモードへ遷移するまでの時間を考慮し、MODE レジスタ（0x03）のポーリング処理を引き続き含めています：*APP*。
2. **Mode == PTCH:**

   PTCH および *APP* モードは、PD コントローラの TRM に記載されています。PD コントローラ評価基板の EEPROM は無効になっている（ SDA が切断されている）ため、PD コントローラは *PTCH* モードに遷移し、その状態を維持します。プロセス開始時の *PTCH* モードのチェックは必要ありませんが、完全性のために含まれています。以下は、コマンドの例で、ロジック アナライザ キャプチャの例は *PTCH* モードの読み取りに示しています。

   [0x21]+ ACK（固有のアドレス/ Wr/A）

   0x03 + ACK（レジスタ番号/A）

   [0x21]+ ACK（固有のアドレス/ R/A）

   0x04（バイト数）

   0x50 0x54 0x43 0x48（*PTCH* は 4ASCII 文字）

図 4-2. *PTCH* モードの読み取り

3. **I2Ct_IRQ の設定、CMD1 の完了：**

   CMD1 完了割り込みは、PBMs コマンドが完了したことを EC に通知するために使用されます。割り込みマスクの設定と割り込みのクリアは、それぞれレジスタ 0x16 および 0x18 で行います。1 と 2 を参照

   [0x21]+ ACK（固有のアドレス/ Wr/A）

   0x16 + ACK（レジスタ番号/A）

   0x0B（バイト数）

   0x00 0x00 0x00 0x40 0x00 0x00 0x00 0x00 0x00 0x00 0x01 (MSB)

図 4-3. 割り込みマスクレジスタ、0x16 の設定

   [0x21]+ ACK（固有のアドレス/ Wr/A）

   0x18 + ACK（レジスタ番号/A）

   0x0B（バイト数）

   0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF 0xFF (MSB)

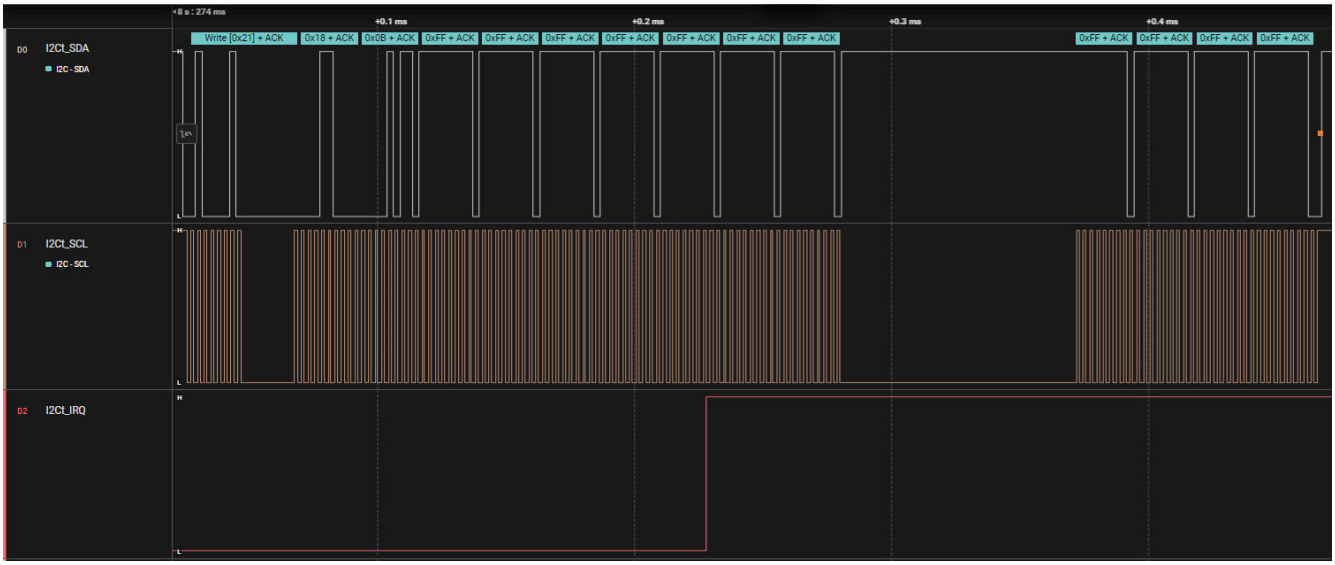

図 4-4. 割り込みクリア レジスタ、0x18

4. ***PBMs データのみ:***

PBMs コマンドは TRM リファレンスで定義されています。この例では、PBMs のパラメータを表 4-1 に記載しています。

表 4-1. PBMs の設定:DATA1 レジスタ

説明	値	コメント
バンドルサイズ	0x00002C80	セクション 5 を参照
I2C バーストデータターゲットアドレス	0x30	0x30、参考資料 1 を参照。
タイムアウト	0x31	3.1 秒;参考資料 1 を参照

[0x21]+ ACK(固有のアドレス/ Wr/A)

0x09 + ACK(レジスタ番号/A)

0x06(バイト数)

0x80 0x2C 0x00 0x00 0x30 0x32(バンドルサイズ、I2C ターゲットアドレス、タイムアウト値)

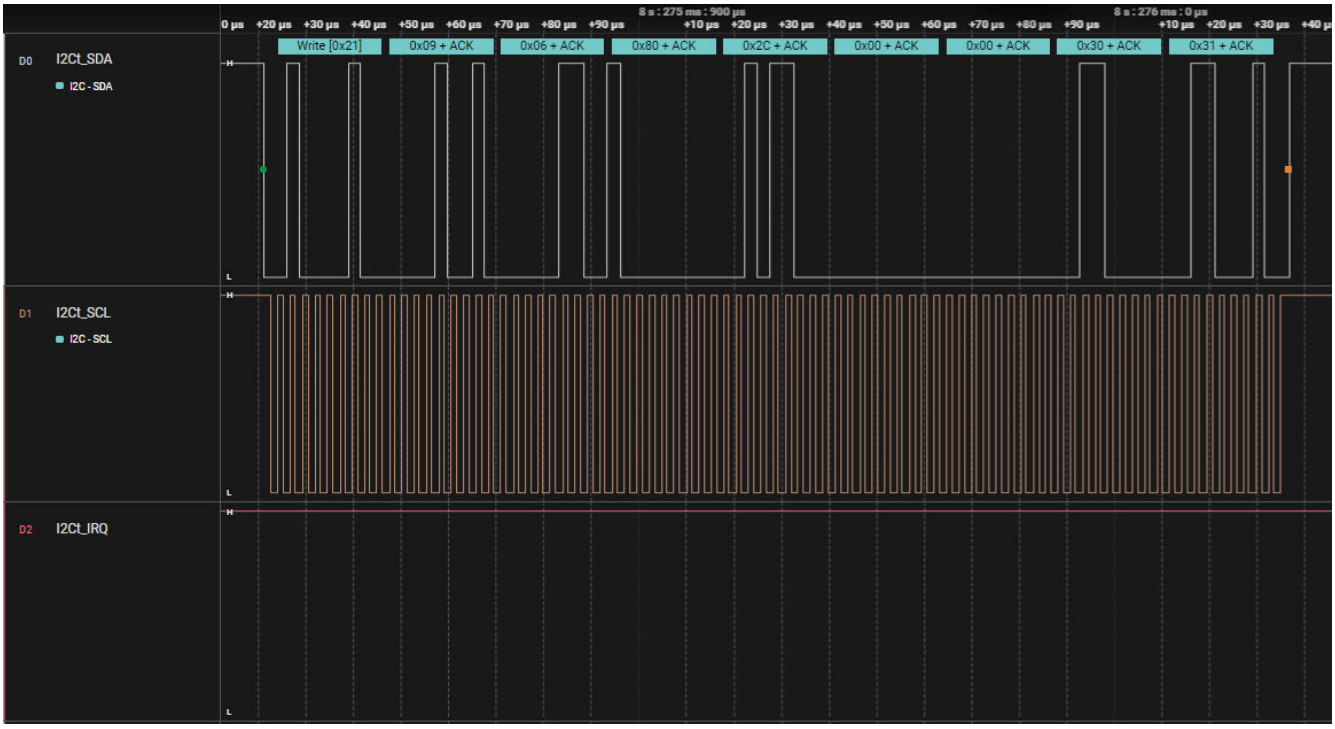

図 4-5. DATA1 レジスタ、0x09、PBMs 用

5. **データグッド:**

PBMs コマンドを送信するには、データレジスタへの書き込みを複数回行う必要があります。この例では、レジスタ 0x08 で PBMs コマンドを書き込む前に、0x09 の値を確認します。レジスタ 0x09 の書き込みと読み取りの間には、500us の遅延があります。

[0x21]+ ACK(固有のアドレス/ Wr/A)

0x09 + ACK(レジスタ番号/A)

[0x21]+ ACK(固有のアドレス/ R/A)

0x40(バイト数)

0x00 0x00 0x00 0x00 0x00 0x00(誤り, DATA1 のリライト)

0x80 0x2C 0x00 0x00 0x30 0x32（正しい、CMD1 の書き込みに進む）

6. **PBMs CMD1：**

   DATA1 を確認した後、CMD1=*PBMs* を書き込みます。I2Ct_IRQ は、図 4-6 に示すように low にアサートされます。

   [0x20]+ ACK（固有のアドレス/ Wr/A）

   0x08 + ACK（レジスタ番号/A）

   0x04（バイト数）

   0x50 0x42 0x4D 0x73（*PBMs* は 4ASCII 文字）

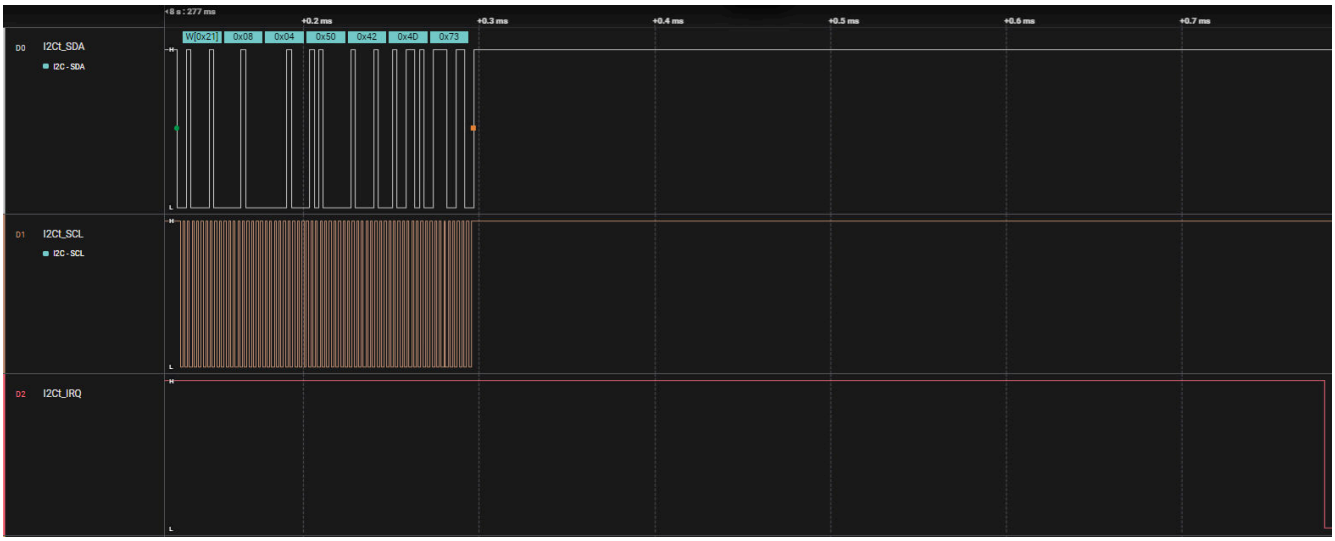

図 4-6. PBMs を CMD1 レジスタ（0x08）に書き込む

7. **I2Ct_IRQ == Low**

   IRQ 信号は CMD1 完了イベントの発生を示し、CMD1 および DATA レジスタを読み取ることで PBMs コマンドの結果を確認できます。期待される結果をステップ 8 と 9 に示します。

8. **CMD1 クリア（PBMs）**

   コマンドレジスタ（0x08）は、内容がクリアされることでコマンドの正常完了を示します。単純化すると、この例では最初のビットをチェックし、PBMs コマンドの破損、または CMD1 の IRQ が未完了であれば、データレジスタに不正な値がロードされたことを示します。[4]。

   [0x21]+ ACK（固有のアドレス/ Wr/A）

   0x08 + ACK（レジスタ番号/A）

   [0x21]+ ACK（固有のアドレス/ R/A）

   0x04（バイト数）

   0x00 0x00 0x00 0x00

[4] CMD1 完了の IRQ が使用されていなかった場合、CMD レジスタは、[0x50、0x42、0x4D、0x73]から[0x00、0x00、0x00、0x00]または[0x21、0x43、0x4D、0x44]のいずれかに遷移するまでポーリングできます。CMD1[0] = 0x21 はコマンドの失敗を、CMD1[0] = 0x00 は正常完了を示します。

9. **DATA1 クリア（PBMs）**

   データレジスタ（0x09）は、先頭バイト PatchStartStatus がクリアされることで、パッチの成功を示します。PatchStartStatue、0x04、0x05、0x06 といったゼロ以外の値は、それぞれ無効なバンドルサイズ、ターゲットアドレス、またはタイムアウト値を示します。1 を参照してください。

   [0x21]+ ACK（固有のアドレス/ Wr/A）

   0x09 + ACK（レジスタ番号/A）

   [0x21]+ ACK（固有のアドレス/ R/A）

   0x40（バイト数）

   0x00 0x00 0x00 0x00 0x30 0x31

10. **バーストデータ**

    このステップでは PMBUS 形式は使用せず、バイナリイメージの内容を PBMs コマンドの表で指定された I2C バーストデータターゲットアドレスに直接書き込みます。バースト形式は、マイコンのアーキテクチャによって影響を受けます。この場合、バーストサイズは 4KB に制限されているため、3 回の連続バースト（4095 バイト、4095 バイト、3202 バイト）が、各バースト間に 500us の遅延を挟んで PD に送信されます。最終バーストの終了時点を基準として PBMc コマンドが送信されると、さらに 500us の遅延が追加されます。

    [0x30]+ ACK（固有のアドレス/ Wr/A）

    lowRegion_i2c_array[0], lowRegion_i2c_array[1]..., lowRegion_i2c_array[4094]

    [0x30]+ ACK（固有のアドレス/ Wr/A）

    lowRegion_i2c_array[4095], lowRegion_i2c_array[4096]..., lowRegion_i2c_array[8189]

    [0x30]+ ACK（固有のアドレス/ Wr/A）

    lowRegion_i2c_array[8190], lowRegion_i2c_array[8191]..., lowRegion_i2c_array[11391]

11. **I2Ct_IRQ、CMD1 完了の設定**

    CMD1 完了割り込みは、PBMc コマンドの完了を EC に通知するために使用されます。割り込みマスクの設定と割り込みのクリアは、それぞれレジスタ 0x16 および 0x18 で行います。割り込みマスクは、ステップ 3 ですでに設定されています。図 4-4 および図 4-7 に示すように、割り込みを繰り返しクリアします。

12. **PBMc CMD1 限定**

    PBMc コマンドには入力データは含まれないため、コマンドのみが送信されます。

    [0x21]+ ACK（固有のアドレス/ Wr/A）

    0x08 + ACK（レジスタ番号/A）

    [0x21]+ ACK（固有のアドレス/ R/A）

    0x04（バイト数）

    0x50 0x42 0x4D 0x63（「PBMc」は 4ASCII 文字）

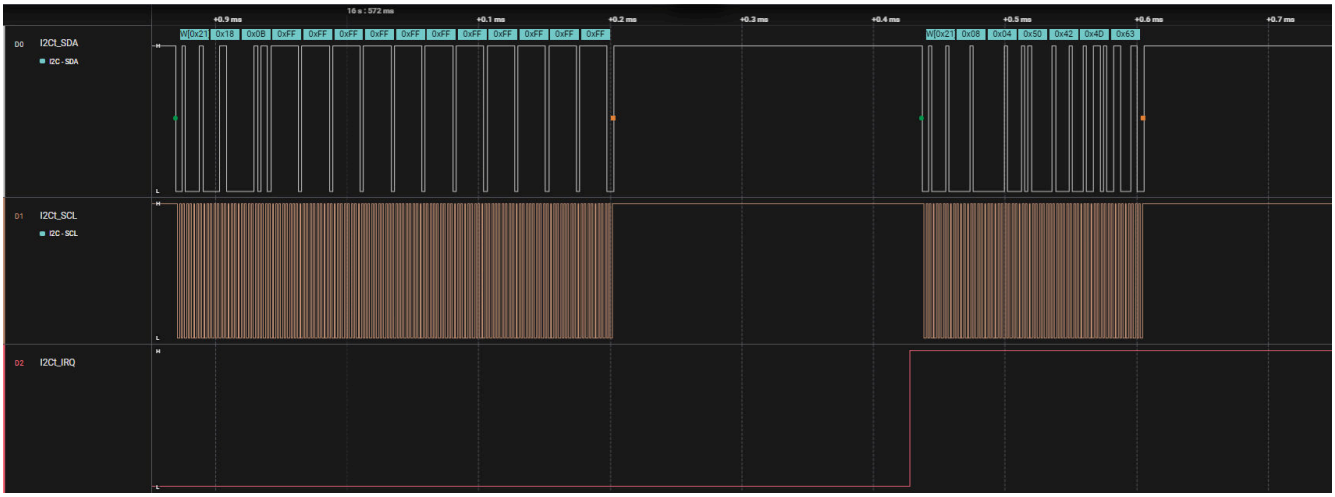

図 4-7. 割り込みのクリア、PBMc の CMD1 レジスタへの書き込み

13. **I2Ct_IRQ == Low**

    IRQ 信号は CMD1 完了イベントの発生を示し、CMD1 レジスタを読み取ることで PBMc コマンドの結果を確認できます。

14. **CMD1 のクリア（PBMc）**

    PBMs CMD1 と同様に、CMD1 レジスタを読み取って元のコマンドがクリアされており、*!CMD* でないことを確認します。

図 4-8. I2Ct_IRQ アサート後の CMD1 の読み取り（PBMc）

15. **遅延 20ms** [5]

    20ms の遅延により、PD コントローラはイメージを読み込み、適用できます。遅延が経過すると、DATA1 および Mode レジスタが読み取られ、成功が確認されます。

[5] この 20ms の遅延は、Patch Loaded 割り込みで置き換えることができます。遅延の代わりにこの割り込みを使用しても時間を短縮することはできませんでした。そのため、このドキュメントでは、デバイスの TRM に記載されている 20ms の遅延を維持しています。1 と 2 を参照

図 4-9. CMD1 および DATA1 レジスタの読み出し遅延間隔

16. **DATA1 のクリア（PBMc）**

この例では、DATA1 レジスタから 40 バイトが読み取ります。

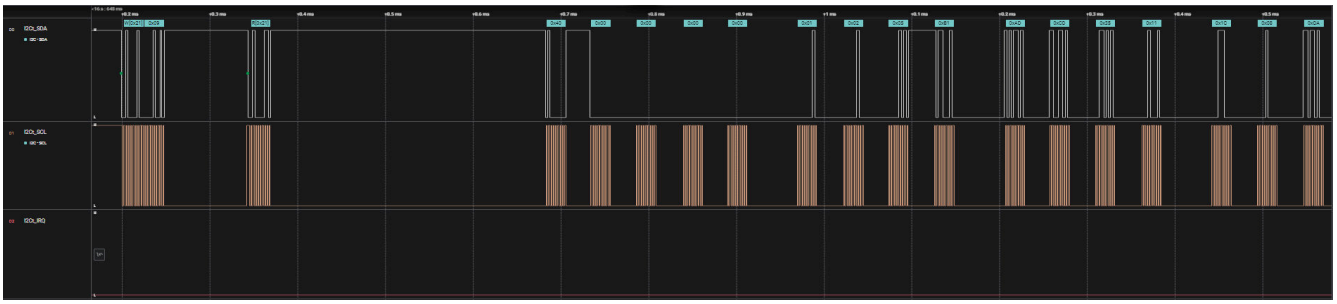

図 4-10. PBMc 完了後の DATA1 レジスタの読み取り

17. **Mode == APP**

最後のステップは、PD が APP モードに移行したことを確認することです。APP モードに移行すると、PD コントローラはカスタム設定が適用された状態で動作可能になります。

[0x21]+ ACK（固有のアドレス/ Wr/A）

0x03 + ACK（レジスタ番号/A）

[0x21]+ ACK（固有のアドレス/ R/A）

0x04（バイト数）

0x41 0x50 0x50 0x20（「アプリケーション」は 4 ASCII 文字）

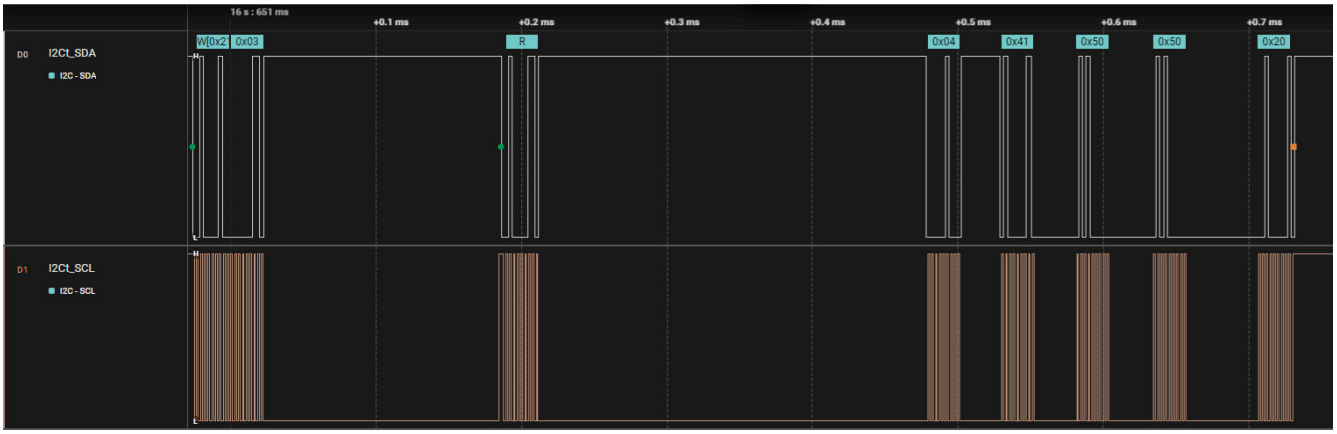

図 4-11. APP モード

4.2 下位領域バイナリの生成ステップ

USBC PD アプリケーションカスタマイズツール（GUI）は、多くの質問を活用してカスタム PD コントローラ設定を生成します。設定が完了すると、設定とファームウェアアップデートを含むパッチバンドルを生成できます。フルフラッシュバイナリは、I2Cc 経由で EEPROM から読み込むときに PD コントローラが期待する形式のイメージを生成します。図 4-12 に示す下位領域のバイナリは、より小さい形式であり、かつ I2Ct 経由で PD コントローラに書き込むことを目的にしています。この下位領域のバイナリは、ステップ 10 で送信されるものです。

1. GUI で質問に回答します。
2. 必要であれば、Advanced Configuration スイッチを使用して追加設定を行います。
3. Export ドロップダウンメニューからを選択して、低領域バイナリを生成します。
4. 適切な形式と希望のファイル名を選択します。

図 4-13 に示すように、GUI には下位領域ファイルをバイナリまたは C ファイルとしてエクスポートするオプションがあります。このアプリケーションノートでは、EC コードを構築するため、C ファイル形式が選択され、これはデバイスプロジェクトに含まれています。GUI から生成されたソースファイルと関連する定義の変更については、セクション 5 を参照してください。

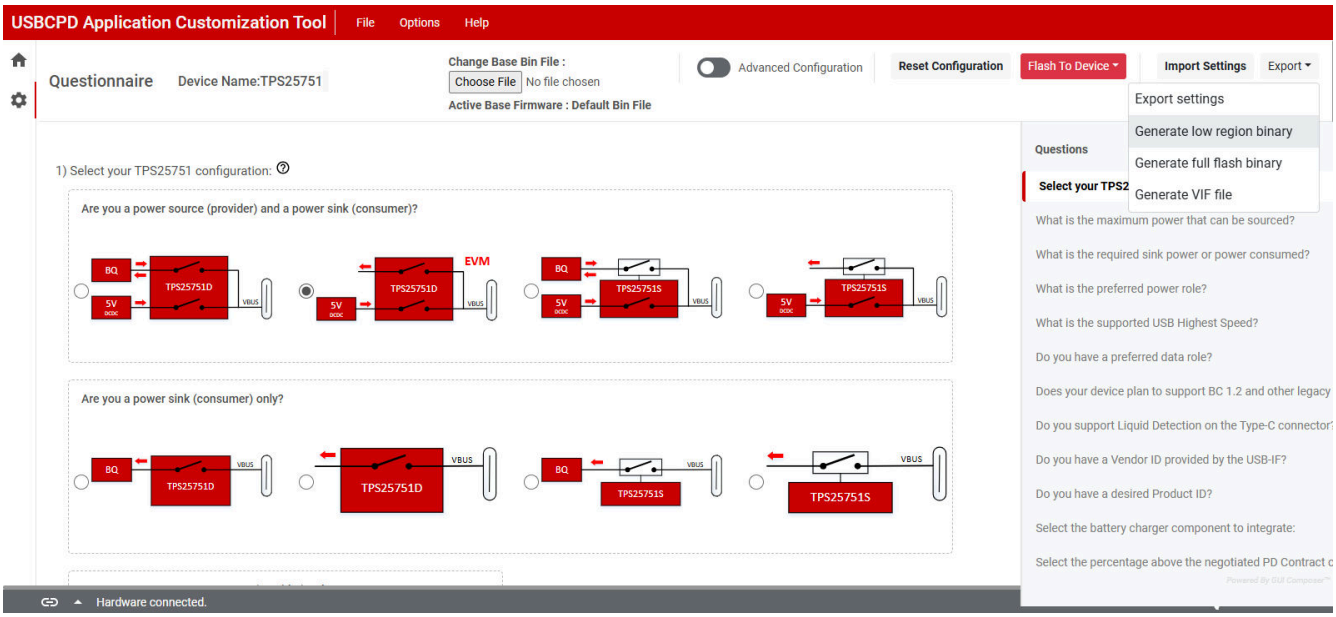

図 4-12. USBCPD アプリケーションカスタマイズツールによる画像生成

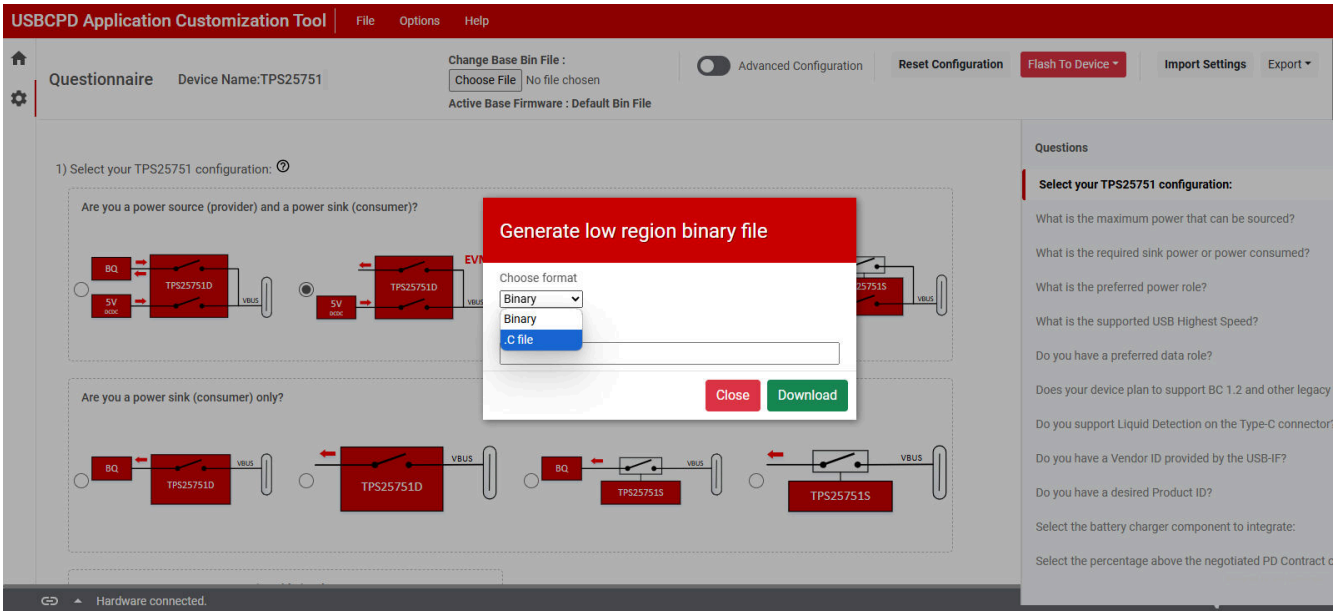

図 4-13. 画像出力タイプの選択

5 サンプル コード

GUI 生成イメージファイルに、以下のコードのような軽微な修正を加えています。

```
#include "ti_msp_dl_config.h"
#include "lrb.h"

const uint8_t tps25751_lowRegion_i2c_array[SIZEOFLRB] = {
0x01, 0x00, 0xe0, 0xac, 0xfe, 0xff, 0xff, 0xff, 0x80, 0x06, 0x00, 0x00, 0x00, 0x26, 0x00, 0x00,
0x71, 0x3e, 0x9c, 0xb8, 0x00, 0x00, 0x00, 0x00, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff,
...
0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
```

ファイル *lrb.h* は、コンテキストの次のコードに示されています。

```
#define SIZEOFLRB  0x2C80   // 11392
/* Maximum size of TX packet, MCU 0xFFF */
#define I2C_TX_MAX_PACKET_SIZE
/* SIZEOFLRB/I2C_TX_MAX_PACKET_SIZE +1 */
#define LRB_NUMBER_OF_PACKETS  (0x0003)
/* SIZEOFLRB % I2C_TX_MAX_PACKET_SIZE */
#define LRB_REMAINDER          (0x0C82)
extern const uint8_t tps25751_lowRegion_i2c_array[SIZEOFLRB];
```

以下のサンプルコードは、MSPM0C110x ドライバライブラリ [6] とセクション 4.1 に示すステップに基づいています。

```
    /*
     *  1. Wait for Ready for Patch interrupt
     */
    debounce = 1;
    /* Skip glitch at boot */
    while(debounce)
    {
        while(DL_GPIO_readPins(
                GPIO_SWITCHES_PORT, GPIO_SWITCHES_USER_SWITCH_1_PIN));
        delay_cycles(DELAY_500us);
        if(!DL_GPIO_readPins(
                GPIO_SWITCHES_PORT, GPIO_SWITCHES_USER_SWITCH_1_PIN))
        {
             debounce = 0;
        }
    }
    /*
     *  2. Mode == PTCH
     */
    readRegister(MODE_REGISTER, MODE_BYTE_CNT, gRxPacket);
    if(gRxPacket[1] != 0x50)
    {
        delay_cycles(DELAY_10ms);
        readRegister(MODE_REGISTER, MODE_BYTE_CNT, gRxPacket);
    }
    /*
     *  3a. Configure I2Ct_IRQ for CMD1 Complete
     */                                                    */
    gTxPacket = (uint8_t*)maskIntReg;
    writeRegister(INT_BYTE_CNT, (uint8_t*)gTxPacket);
    waitingForPD = 1;
    while(waitingForPD)
    {
        /*
         *  3b. Clear Interrupts
         */
        gTxPacket = (uint8_t*)clrIntReg;
        writeRegister(INT_BYTE_CNT, (uint8_t*)gTxPacket);
        //  Wait for interrupt to clear
        while(DL_GPIO_readPins(GPIO_SWITCHES_PORT, GPIO_SWITCHES_USER_SWITCH_1_PIN)==0);
        waitingForGoodData = 1;
        while(waitingForGoodData)
```

[6] MSPM0 SDK（2.05.00.05）/Examples/Development Tools /LP_MSPM0C1104 LaunchPad/DriverLib/i2c_controller_rw_multiple_fifo_interrupts

```
        {
            /*
             *  4. Send PBMs Data
             */
            gTxPacket = (uint8_t*)PBMsData;
            writeRegister(DATA_BYTE_CNT, gTxPacket);
            delay_cycles(DELAY_500us);
            /*
             * 5. Data Good
             */
            readRegister(DATA_REG, (DATA_BYTE_CNT-1), gRxPacket);
            if(gRxPacket[1] == 0x80)
            {
                waitingForGoodData = 0;
            }
        }
        /*
         *  6. Send PBMs to CMD1
         */
        gTxPacket = (uint8_t*)PBMsCmd;
        writeRegister(CMD_BYTE_CNT, gTxPacket);
        /*
         *  7. Wait for I2Ct_IRQ
         */
        while(DL_GPIO_readPins(
                    GPIO_SWITCHES_PORT, GPIO_SWITCHES_USER_SWITCH_1_PIN));
        /*
         *  8. CMD1 Clear
         */
        readRegister(CMD_REG, (CMD_BYTE_CNT-1), gRxPacket);
        /*
         *  9. DATA1 Clear
         */
        readRegister(DATA_REG, (DATA_BYTE_CNT-1), gRxPacket);
        if(gRxPacket[1] == 0x0)
        {
            waitingForPD = 0;
        }
    }
    /*
     *  10. Burst Data
     */
    ii = LRB_NUMBER_OF_PACKETS;
    while(ii)
    {
        if(ii == 1)
        {
            gTxLen = LRB_REMAINDER;
        }
        else
        {
            gTxLen = I2C_TX_MAX_PACKET_SIZE;
        }
        /* The FIFO is 8-bytes deep, and this function returns number of bytes written to FIFO */
        gTxPacket = (uint8_t*)&tps25751_lowRegion_i2c_array[(LRB_NUMBER_OF_PACKETS-
ii)*I2C_TX_MAX_PACKET_SIZE];
        ii = ii-1;
        gTxCount = DL_I2C_fillControllerTXFIFO(I2C_INST, gTxPacket, gTxLen);
        DL_I2C_enableInterrupt(
                I2C_INST, DL_I2C_INTERRUPT_CONTROLLER_TXFIFO_TRIGGER);
        gI2cControllerStatus = I2C_STATUS_TX_STARTED;
        while (!(
            DL_I2C_getControllerStatus(I2C_INST) & DL_I2C_CONTROLLER_STATUS_IDLE))
            ;
        DL_I2C_startControllerTransfer(
            I2C_INST, I2C_TGT_BURST_ADDRESS, DL_I2C_CONTROLLER_DIRECTION_TX, gTxLen);
        while ((gI2cControllerStatus != I2C_STATUS_TX_COMPLETE) &&
            (gI2cControllerStatus != I2C_STATUS_ERROR)) {
            __WFE();
        }
        while (DL_I2C_getControllerStatus(I2C_INST) &
            DL_I2C_CONTROLLER_STATUS_BUSY_BUS)
            ;
        /* Trap if there was an error */
        if (DL_I2C_getControllerStatus(I2C_INST) &
            DL_I2C_CONTROLLER_STATUS_ERROR) {
            __BKPT(0);
        }
```

```
        while(!(
            DL_I2C_getControllerStatus(I2C_INST) & DL_I2C_CONTROLLER_STATUS_IDLE))
            ;
        delay_cycles(DELAY_500us);
    }
    /*
     *  11. Clear Interrupts
     */
    gTxPacket = (uint8_t*)clrIntReg;
    writeRegister(INT_BYTE_CNT, (uint8_t*)gTxPacket);
    //  Wait for interrupt to clear
    while(DL_GPIO_readPins(
        GPIO_SWITCHES_PORT, GPIO_SWITCHES_USER_SWITCH_1_PIN)==0)
        ;
    /*
     *  12. Send PBMc to CMD1 Register
     */
    gTxPacket = (uint8_t*)PBMcCmd;
    writeRegister(CMD_BYTE_CNT,gTxPacket);
    /*
     *  13. I2Ct_IRQ == Low, CMD1 complete
     */
    while(DL_GPIO_readPins(
        GPIO_SWITCHES_PORT, GPIO_SWITCHES_USER_SWITCH_1_PIN));
    /*
     *  14. Read CMD1
     */
    readRegister(CMD_REG, (CMD_BYTE_CNT-1), gRxPacket);
    /*
     *  15. Delay 20ms
     */
    delay_cycles(DELAY_20ms);
    /*
     *  16. Read Data1
     */
    readRegister(DATA_REG, 0x28, gRxPacket);
    /*
     *  17. MODE === APP
     */
    waitingForPD = 1;
    while(waitingForPD)
    {
        readRegister(MODE_REGISTER, MODE_BYTE_CNT, gRxPacket);
        if(gRxPacket[1] == 0x41)
        {
            waitingForPD = 0;
        }
        else
        {
            delay_cycles(DELAY_10ms);
        }
    }
```

6 参考資料

1. テキサス インスツルメンツ、*『TPS25751 テクニカル リファレンス マニュアル』*、テクニカル リファレンス マニュアル。
2. テキサス インスツルメンツ、*『TPS26750 テクニカル リファレンス マニュアル』*、テクニカル リファレンス マニュアル。
3. テキサス インスツルメンツ、*『TPS25751 USB Type-C® および USB PD コントローラ、ソース パワー スイッチ内蔵』* データシート。
4. テキサス インスツルメンツ、*『TPS26750 USB Type-C® および USB PD コントローラ、電源アプリケーション用に最適化されたパワー スイッチ内蔵』*データシート。
5. テキサス インスツルメンツ、*MSPM0 ソフトウェア開発キット (SDK)*、ソフトウェア
6. テキサス インスツルメンツ、*PTCH から APP へ*、E2E™設計サポートフォーラム。
7. テキサス インスツルメンツ、*Salae コード*、E2E™設計サポートフォーラム。
8. テキサス インスツルメンツ、*マスクオフ*、E2E™設計サポートフォーラム。

7 改訂履歴

Changes from Revision * (July 2024) to Revision A (June 2025)	Page
• ドキュメントに TPS26750 を追加	1
• ドキュメント全体にわたって表、図、相互参照の採番方法を更新	1
• TPS25751 への特定のリファレンスを削除し、汎用化	2
• TPS25751 に対する特定のリファレンスを削除	2
• スコープをパッチバーストモードに変更	3

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