参考資料

Application Note

2 ピン・インターフェイスを使用した TWS の高効率充電

Andrew Wallace, Nick Brylski, Gautham Ramachandran, and Albert Lo

概要

このレポートでは、2 ピン・インターフェイスを使用した高効率の充電方法を紹介します。このレポートでは、TWS システムが充電モードと通信モードを切り替えるために必要なハードウェアの概要を説明します。イヤホンのバッテリ電圧に追従するようにケースの出力電圧を動的に調整することで、高い効率を実現します。このレポートでは、ケース内のマイコンとイヤホンのソフトウェア・フローについても説明します。使用するすべてのコードと回路図が含まれています。

目次

図の一覧

表の一覧

商標

すべての商標は、それぞれの所有者に帰属します。

1 はじめに

完全ワイヤレス・ステレオ (TWS) を使用するお客様は、フル・ケース・バッテリ 1 個あたりのイヤホン充電回数を増やし、充電中の発熱を最小限に抑えて、ユーザーの使いやすさを向上させるように努力しています。リニア・バッテリ・チャージャを使用してイヤホンのバッテリを充電するのが最も一般的です。したがって、充電効率を最大化するには、現在のイヤホンのバッテリ電圧に基づいてケースの出力電圧を動的に更新する必要があります。これにより、ケースとバッテリ間の通信インターフェイスが必要になります。イヤホンのバッテリ電圧に加えて、このインターフェイスは充電ステータス、デバイス温度、その他の障害などの有用な情報を充電ケースに返すこともできます。

TWS の設計では、イヤホンとケースの間のインターフェイスに必要なピン数が主要な機械的考慮事項になります。イヤホンとケースの間で通信を実装する多くの場合、各イヤホンに 3～5 本のピンが必要になります。機械的な観点からは、充電の最小値である 2 本のピンが理想的です。この課題は、充電を実行しながら 2 ピンの通信インターフェイスを実装する方法になります。

2 システム概要

図 2-1 に、2 ピン充電システムのブロック図を示します。このシステム内のデバイスの機能と特長を以下に示します。

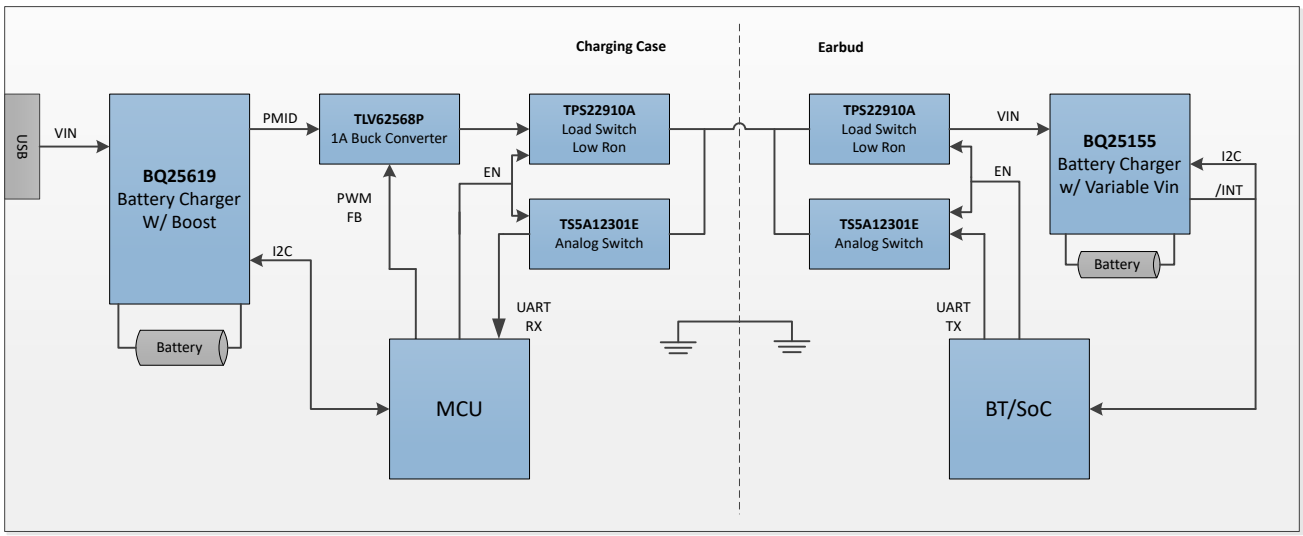

図 2-1. システム・ブロック図

2.1 充電ケース

2.1.1 BQ25619

このデバイスは、高効率、1.5MHz の同期整流スイッチモード降圧チャージャです。パワー・パス機能が内蔵されており、USB アダプタが存在する場合はケースのバッテリとイヤホンのバッテリを同時に充電できます。USB アダプタから切断されると、BQ25619 は 4.6V～5.15V の選択可能な昇圧電圧を出力できます。これにより、システム内で個別の昇圧コンバータが不要になります。このシステムでは、昇圧モードを 4.6V に設定して、より高効率で充電できるようにしています。

2.1.2 TLV62568P

これは 1A の降圧コンバータで、ケースの出力電圧を制御するために使用されます。この出力電圧は、RC フィルタ付き PWM 信号をフィードバック・ピンに印加することで調整されます。ここにリンクされているホワイト・ペーパーは、動的電圧制御用のフィードバック・システムを設計する手順の概要を示しています。このシステムは動的な電圧制御を使用して、イヤホンのリニア・バッテリ・チャージャ・デバイスへのドロップアウトを最小限に抑えます。ヘッドルームを小さくすることで、充電中のイヤホンのリニア・バッテリ・チャージャの消費電力と温度上昇を最小限に抑えることができます。

2.1.3 TPS22910A

これは、ケースの出力電源レールの制御に使用するロード・スイッチです。また、システムが通信モードのときは、降圧コンバータの出力容量を信号ラインから分離します。これは、クイック出力放電 (QOD) のないアクティブ・ロー・ロード・スイッチです。このデバイスを使用すると、パワー・パスをシステムのデフォルトに設定できるため、ケースやイヤホンのバッテリ切れケースがシステムに電力を供給することはありません。

2.1.4 TS5A12301E

このデバイスはアナログ・スイッチで、イヤホンとケースの間で UART 通信を渡すために使用されます。このデバイスは、ロジック信号を 1.2V までスイッチングでき、電源オフ保護機能を備えています。電源オフ保護機能により、デバイスが充電モードのときにマイコンの UART ピンを保護できます。このアプリケーションでは、電源オフ保護機能を備えていないアナログ・スイッチ・デバイスは失敗します。これは、イヤホンの入力電圧が 3.3V～4.6V の範囲内にある場合、多くのスイッチの入出力電圧定格 Vcc + 0.5V を上回るためです。この機能がない場合、この定格を超えるとマイコンの GPIO ピンで過電圧イベントが発生する可能性があります。

2.1.5 マイコン

このホワイト・ペーパーで説明するシステムでは、MSP430FR5529 マイコンを使用してケース処理機能を制御します。このアプリケーションでは、マイコンが通信方式用に 1 つの UART チャネルと 1 つの GPIO を備えている必要があります。BQ25619 を制御するには、I^2C チャネルが必要です。また、5 本の GPIO ピンを BQ25619 に接続して、デバイスの入力と制御を行うこともできます。TLV62568P を制御するには、デバイスの出力を動的に制御するために PWM 対応の GPIO ピンが必要です。また、デバイスの PMID_GOOD ピンを読み取るために追加の GPIO を 1 つ使用することもできます。

セクション 10.1 に、充電ケースのソフトウェアを示します。

2.2 イヤホン

2.2.1 BQ25155

このデバイスは、ADC を内蔵したリニア・リチウム・イオン・バッテリ・チャージャです。充電中に入力電圧を可変にすることができます。この機能により、このレポートで概要を説明した効率的な充電方法が実現します。このデバイスには、バッテリ電圧 (VBAT)、バッテリ温度 (TS)、外部 ADC チャネル (ADCIN) など複数のピンを監視する ADC が内蔵されています。このアプリケーション・ノートのこちらで説明している過熱バッテリ放電機能など、さまざまな機能を実現する 3 つのコンパレータも用意されています。また、このデバイスには割り込みピンがあり、多くの異なるフラグでマイコンの割り込みをトリガできます。このフラグ・ベースの割り込みシステムを使用して、このホワイト・ペーパーで概要を説明している割り込み駆動通信を有効にします。イヤホンの保管寿命を最大化するため、BQ25155 シップモードは 10nA という超低消費電流を実現しています。スペースに制約のあるアプリケーションでは、BQ25155 の 12-mm^2 ソリューション・サイズを活用できます。このアプリケーションでは、低い VIN での充電を可能にするため、BQ25155 の VINDPM 機能がディスエーブルになりました。

2.2.2 TPS22910A

これは、イヤホンの入力電源レールの制御に使用するロード・スイッチです。また、システムが通信モードのときに、BQ25155 の入力容量を信号ラインから分離します。これは、クイック出力放電 (QOD) のないアクティブ "Low" ロード・スイッチです。QOD を搭載したデバイスは、放電抵抗を経由して通信ラインをグランドに短絡するため、アプリケーションで使用できません。デバイスのアクティブ "Low" 機能により、パワー・パスをシステムのデフォルト値に設定できるため、ケースやイヤホンのバッテリ切れケースがシステムに電力を供給することはありません。

2.2.3 TS5A12301E

このデバイスはアナログ・スイッチで、イヤホンとケースの間で UART 通信を渡すために使用されます。このデバイスは、ロジック信号を 1.2V までスイッチングでき、電源オフ保護機能を備えています。電源オフ保護機能により、デバイスが充電

モードのときにマイコンの UART ピンを保護できます。このアプリケーションでは、電源オフ保護機能を備えていないアナログ・スイッチ・デバイスは失敗します。これは、イヤホンの入力電圧が 3.3V～4.6V の範囲内にある場合、多くのスイッチの入出力電圧定格 Vcc + 0.5V を上回るためです。この機能がない場合、この定格を超えるとマイコンの GPIO ピンで過電圧イベントが発生する可能性があります。

2.2.4 BT/SOC

このホワイト・ペーパーで説明するシステムでは、MSP430FR5529 マイコンを使用してイヤホンの処理機能を制御します。TWS アプリケーションでは、通常、このソケットはオーディオ・ストリーミング機能を備えた Bluetooth SOC です。このアプリケーションでは、通信方式をイネーブルにするには、デバイスに 1 つの UART チャネル、1 つの I^2C チャネル、1 つの割り込み対応 GPIO、およびスイッチ制御用の 1 つの GPIO が必要です。BQ25155 を制御するには、3 つの追加 GPIO ピンを使用できます。

セクション 10.2 に、イヤホン用のソフトウェアを示します。

3 充電ケース・アルゴリズムの実装

3.1 初期化とメイン・コード

この通信方式では、このケースはマスタとして機能し、イヤホンはスレーブとして機能します。図 3-1 は、イヤホン・アルゴリズムのフロー・チャートです。

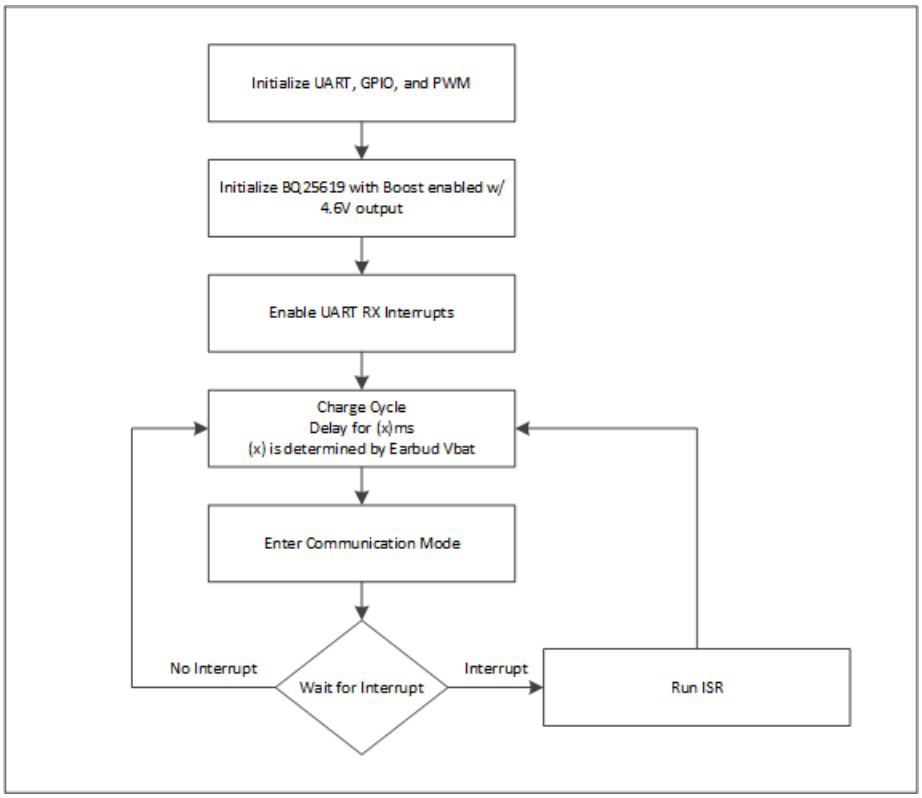

図 3-1. ケース・アルゴリズム

イヤホンは、ユーザーのシステム要件を満たすよう最初に初期化されます。ここで設定するのは、BQ25619 の出力電圧と電流、PWM 周波数、スタートアップ PWM デューティ・サイクル、UART 初期化などです。初期化後、システムの割り込みがイネーブルになります。この時点で、システムはイヤホンが挿入された時点で充電サイクルを開始する準備ができています。

イヤホンを検出すると、充電サイクルが開始されます。ケースは、ユーザー設定時間の間 4.6V を出力します。これにより、イヤホンのバッテリが切れず、通信サイクル中にデバイスが応答できるようになります。ユーザー設定時間が経過すると、ケースは TPS22910A ロード・スイッチをディスエーブルにし、TS5A12301E UART スイッチをイネーブルにすることで、通信サイクルを開始します。このケースは、イヤホンからの UART 送信が割り込みをトリガするのを待ちます。

これらの通信サイクルは、バッテリの充電曲線に基づいてユーザー定義の間隔で発生します。充電サイクルの開始時に、放電されたバッテリが 3.0V～3.7V の範囲になることが予想されます。これは、バッテリが放電されたときの深さと、お客様がバッテリのカットオフ電圧に対して選択した設定によって異なります。この時点で、イヤホンのバッテリ電圧は急速に変化するため、通信間隔を短くする必要があります (5 秒まで)。イヤホンのバッテリ電圧が上昇すると、電圧は緩やかに変化し、間隔を長くできるため、システムが通信モードに移行する必要がある時間を最小限に抑えることができます。

3.2 UART 割り込みおよび出力電圧の調整

UART RX 割り込みがトリガされると、割り込みによって充電完了バイトが受信されたかどうかがチェックされます。充電が完了すると、マイコンはケースを低消費電力モードに設定し、システム状態の変化を待ちます。充電完了バイトを受信しなかった場合、マイコンは受信したイヤホンのバッテリ電圧を保存し、割り込みを続行します。

その後、新しく受信したイヤホンのバッテリ電圧を使用して、PWM デューティ・サイクルを計算します。この計算は、TLV62568P のフィードバック・ピンに接続されている抵抗デバイダとフィルタに基づいています。これらの値に対して、Excel カリキュレータである、DAC を使用した出力電圧調整のデザイン・ツールは、こちらで見つかります。

$$PWMDuty = \left(7.607 - \left(1.434 \times \left(Vbat + 0.33\right)\right)\right) \times \frac{40}{3.3} \quad (1)$$

この計算は、イヤホンのバッテリ充電器が必要とするヘッドルームを考慮して、現在のイヤホンのバッテリ電圧より約 200mV 高い出力電圧を生成するように調整する必要があります。この計算を行った後、PWM デューティ・サイクル値は、3V 未満または 4.5V を超える電圧が印加されないよう制限されます。値が良好であることが判明した場合、マイコンは PWM デューティ・サイクルを調整し、出力電圧がそれに追従します。

充電モードに再移行するには、アナログ・スイッチをオフに切り替え、ロード・スイッチを再度アクティブにします。その後、割り込みが終了します。

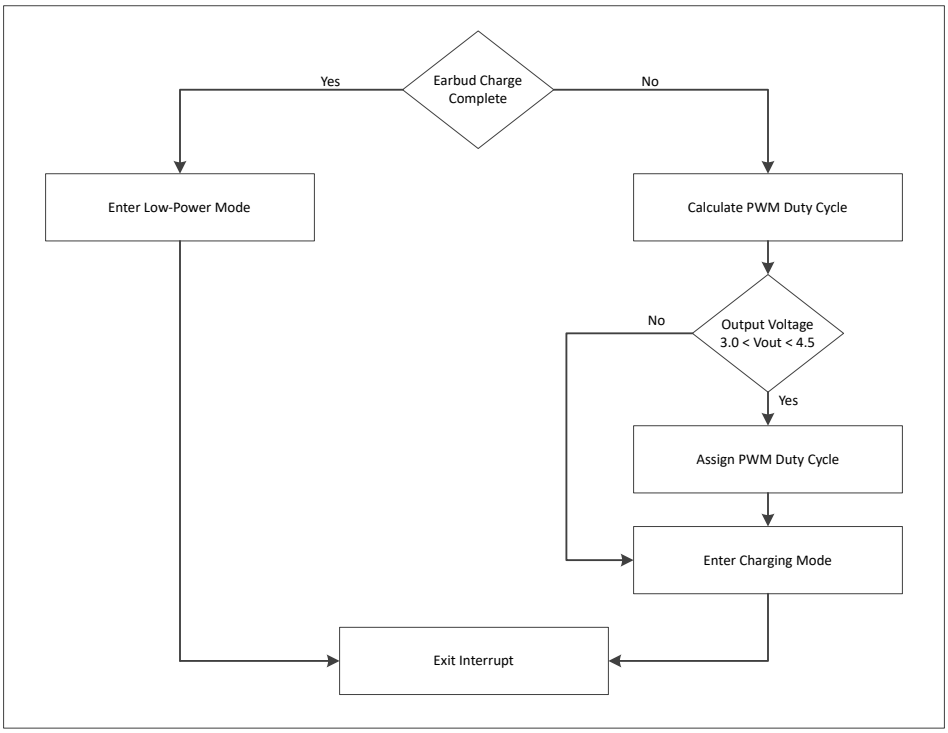

図 3-2. PWM アルゴリズム

4 イヤホン・アルゴリズムの実装

4.1 初期化とメイン・コード

このシステムが実装している通信では、イヤホンはスレーブとして機能します。これにより、ケースが電力モードのときにイヤホンが通信モードに移行しないようにします。このイヤホンは通信モードに移行し、VIN_PGOOD_FLAG (レジスタ・アドレス 0x3) がアサートされ、BQ25155 の内部 ADC が 0V であることを検出した場合のみメッセージを送信します。この信号は、ケースが通信モードに移行したことを示します。図 4-1 は、イヤホン・アルゴリズムのフロー・チャートです。

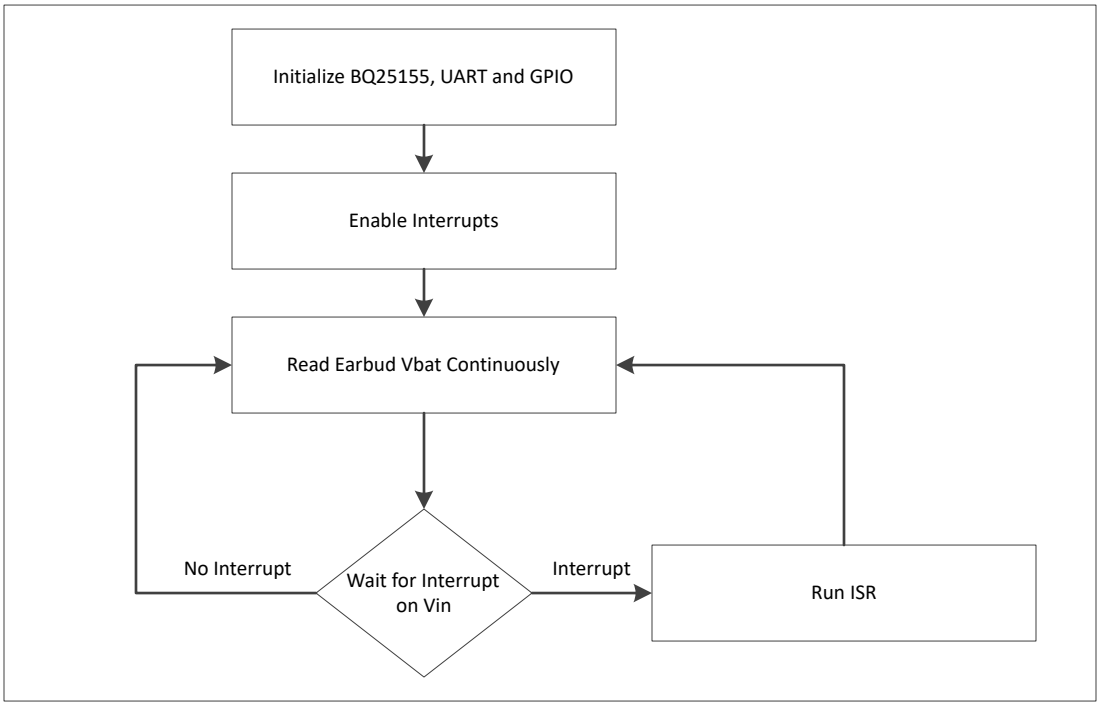

図 4-1. イヤホンのアルゴリズム

イヤホンは、ユーザーのシステム要件を満たすよう最初に初期化されます。ここで設定するのは、BQ25155 の充電電流、ADC 変換レートなどです。初期化後、システムの割り込みがイネーブルになります。この時点で、システムはケースに接続された時点で充電サイクルを開始する準備ができています。

以下の 2 つの表に、変更されたレジスタを示します。

表 4-1. イヤホン BQ25155 レジスタ

名称	値	目的
ICHG_CTRL	0x50	ICHG を 100mA に設定
CHARGERCTRL0	0x92	ウォッチドッグ・タイマをディスエーブルにする
ADCCTRL0	0x58	ADC を変換ごとに 3mS の連続読み取りに設定する
ADCCTRL1	0x00	コンパレータをディスエーブルにする
ADC_READ_EN	0xFE	ADC 読み取りチャネルをイネーブルにする

表 4-2. ケース BQ25619 レジスタ

名称	値	目的
REG01(Charger Control 0)	0x3A	昇圧モードをイネーブルにする
REG05(Charger Control 1)	0x8E	ウォッチドッグ・タイマをディスエーブルにする

表 4-2. ケース BQ25619 レジスタ (continued)

名称	値	目的
REG06(Charger Control 2)	0xC6	昇圧電圧を 4.6V に設定する

システムが初期化され、ケースに接続されると、充電が開始され、ケースによってトリガされる通信サイクルに応答するのを待ちます。通信サイクルを待機する間、イヤホンは 0.5 秒ごとにバッテリ電圧を保存します。これを行う理由は、VIN が 0 になって通信サイクルがトリガされると、イヤホンのバッテリ電圧がわずかに低下し、この時間中に測定値を取得した結果、必要な充電電圧より低い電圧が送信されるためです。

ケースが通信モードに入ると、入力電圧が 0 に低下します。これにより BQ25155 は、VIN が許容可能な電圧を下回ったことを示すフラグを設定し、その INT ピンで割り込みをトリガします。これにより ISR がトリガされます。

4.2 割り込みと送信

マイコンが ISR に入ると、最初に割り込みが VIN_PGOOD_FLAG によって発生したかどうかを確認します。確認する理由は、BQ25155 に設定可能な他の多くの割り込み可能フラグがあるためです。このアプリケーションでは、VIN フラグのみを使用しています。エンド・ユーザーは、BQ25155 によって設定される他のフラグに対して、異なるアクションを実行することを選択できます。

VIN フラグがアサートされた場合、BQ25155 の内部 ADC を使用して割り込みを制限します。これは、250ms にわたって VIN を 3ms ごとに読み取り、最新の 3 つの値を比較することで行います。250ms 以内に割り込みを確認するために連続した 3 つの値が見つからない場合、割り込みはタイムアウトします。

割り込みが制限されている場合は、充電完了レジスタが確認されます。充電が完了すると、充電完了ビットが送信されます。充電が完了していない場合、メイン・ループで読み取ったイヤホンのバッテリ電圧が UART 経由で送信されます。イヤホンのロード・スイッチをディスエーブルにし、アナログ・スイッチをイネーブルにすると、送信が完了します。その後、データは UART 送信バッファにプッシュされ、ケースに送信されます。イヤホンは、スイッチを切り替えることですぐに充電モードに戻ります。これにより、通信に対する応答としてケースが充電モードに戻ったときに、イヤホンのロジック・ピンに誤って電力を供給することを防止できます。この手順の後、イヤホンはメイン・ループに戻ります。

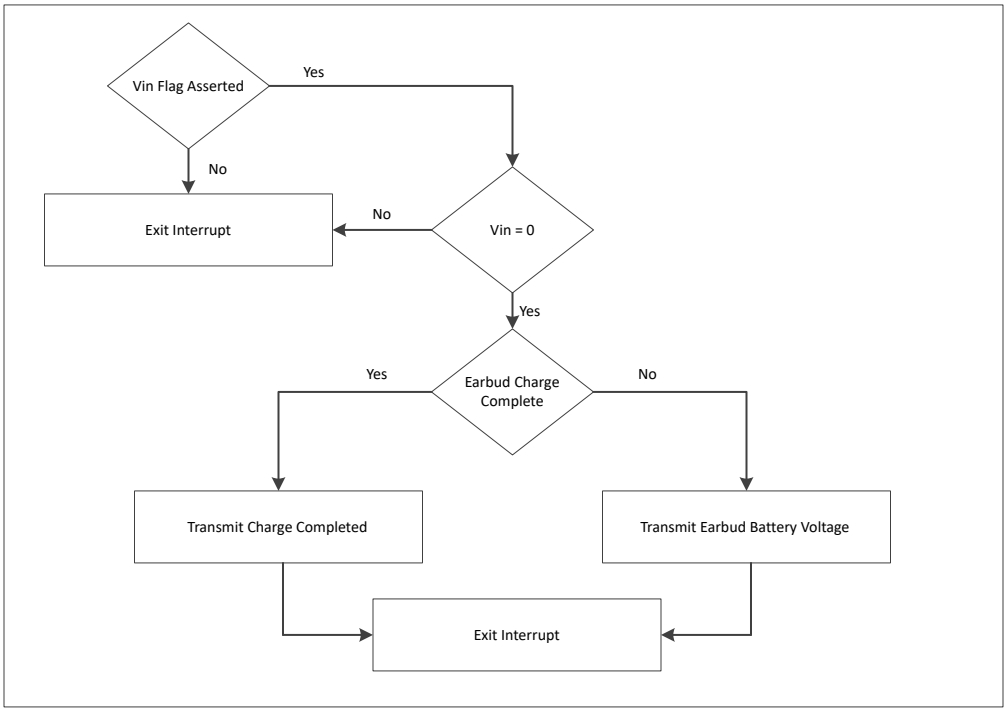

図 4-2. イヤホン割り込みルーチン

5 テスト方法

これらの結果は、195mAh のイヤホン・バッテリの完全な充電サイクルを 200mAh の高速充電電流で実行することで得られたものです。このケースは、フル充電された 400mAh バッテリから電力を供給されています。効率データは、瞬時入力電圧と電流、および出力電圧と電流を測定して計算しました。これらの値は、ケースのバッテリ端子とイヤホンのバッテリ端子でそれぞれ測定されました。瞬時電圧と電流を求めた後、それらを乗算して、システムの瞬時入出力電力を供給しました。その後、瞬時電力を充電サイクル全体に統合して、システムの合計入出力エネルギーを供給しました。

このデータは、約 83 分の充電サイクルにわたってサンプル・レート 500 秒でオシロスコープを使用して収集しました。電圧データは、各バッテリの端子間で直接測定しました。電流データは、INA240 デバイスを使用して、10mΩ のセンス抵抗をバッテリとシステムの入出力端子と直列に接続して測定しました。

熱データは、FLIR サーマル・カメラを使用して取得しました。周囲室温は 25℃で、システムに直接換気は行われていません。

6 テスト結果

このセクションでは、システムの充電効率と放熱を示すことで、3 種類の TWS 充電システムの性能について説明します。下記の 表 6-1 比較図では、結果を簡単に比較しています。

表 6-1. 結果の比較

	動的電圧の調整	BQ25619 4.6V 出力	標準昇圧 5V 出力
出力電圧	3V～5V (PWM 可変)	4.6V	5V
充電効率	84.7%	83.2% (-1.5%)	76.3% (-8.4%)
イヤホンの発熱	27.7℃	31.9℃ (+15.2%)	33.3℃ (+20.2%)
ケース・バッテリ持続時間	最良	より良好	ベースライン

6.1 動的電圧の調整

最初に示したシステムは、このホワイト・ペーパーで説明したシステムです。このデザインは、2 ピンの通信に対応した効率的な充電方式により、最高の性能を実現します。ケースの出力電圧とイヤホンのバッテリ電圧の差を小さくすることで、リニア・バッテリ・チャージャによって消費される電力を低減できます。これにより、最適な放熱性能とケースのバッテリ寿命の延長が可能になります。

提案されたソリューションを使用した充電サイクルのスコープ・キャプチャを以下に示します。ケースの入力での最大電流は 265mA で、イヤホンのバッテリに 200mA を供給します。表示される各パルスは、通信サイクルです。

図 6-1. 動的電圧調整設計の充電サイクル

図 6-2 に、FLIR サーマル・カメラから取得した画像を示します。この画像は、充電プロセス中のシステムの最大温度を示しています。27.7℃の温度は、室温 25℃をわずか 2.7℃上回っています。

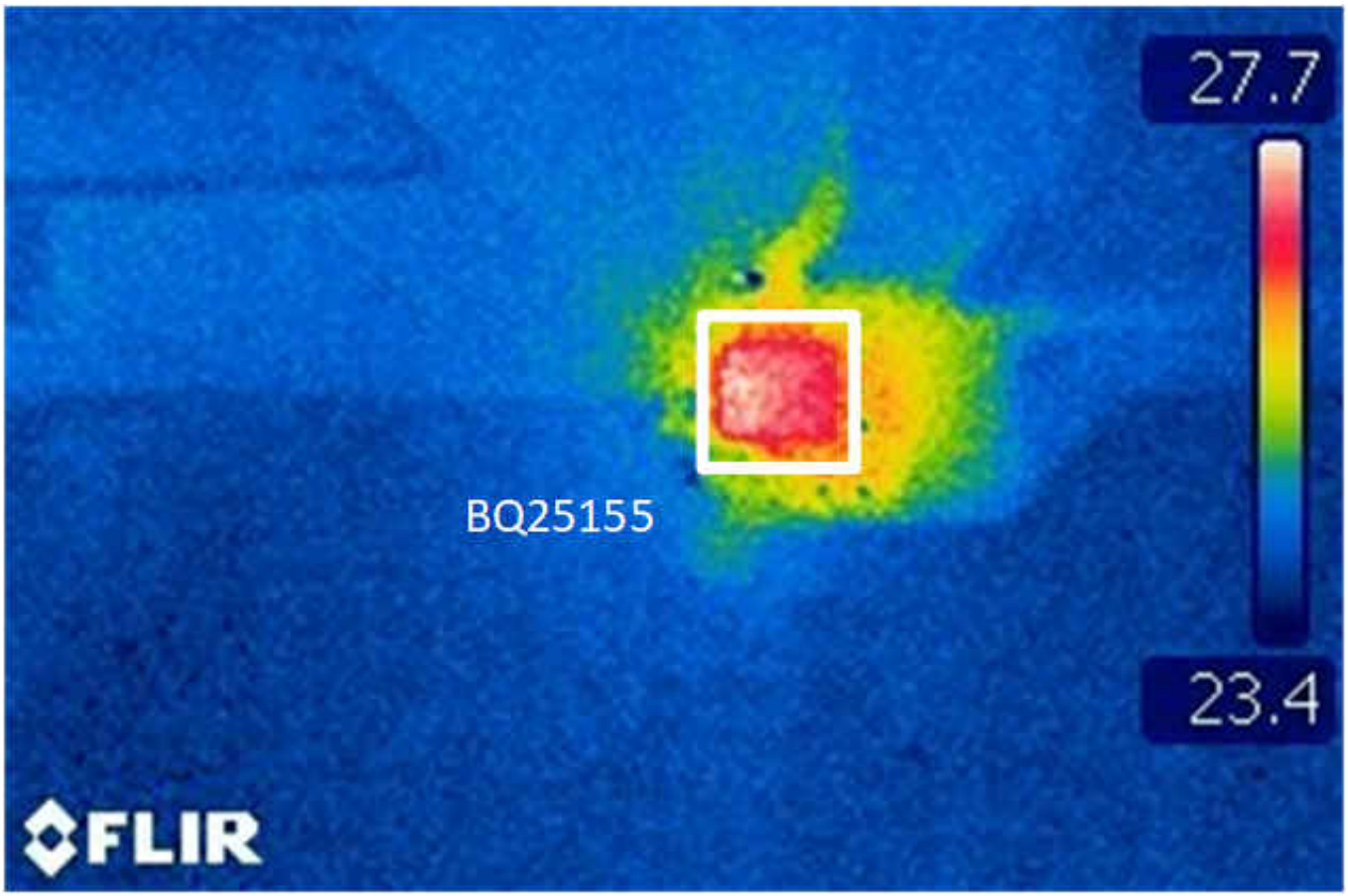

図 6-2. 動的電圧調整設計の放熱性能

6.2 出力 4.6V の BQ25619

2 つ目のシステムは、BQ25619 の昇圧コンバータを使用しており、出力は 4.6V です。このソリューションは、5V の中間電圧を持つシステムよりも高い効率を実現しますが、依然として放熱性能に苦労しています。

図 6-3. 4.6V 出力設計の BQ25619 の充電サイクル

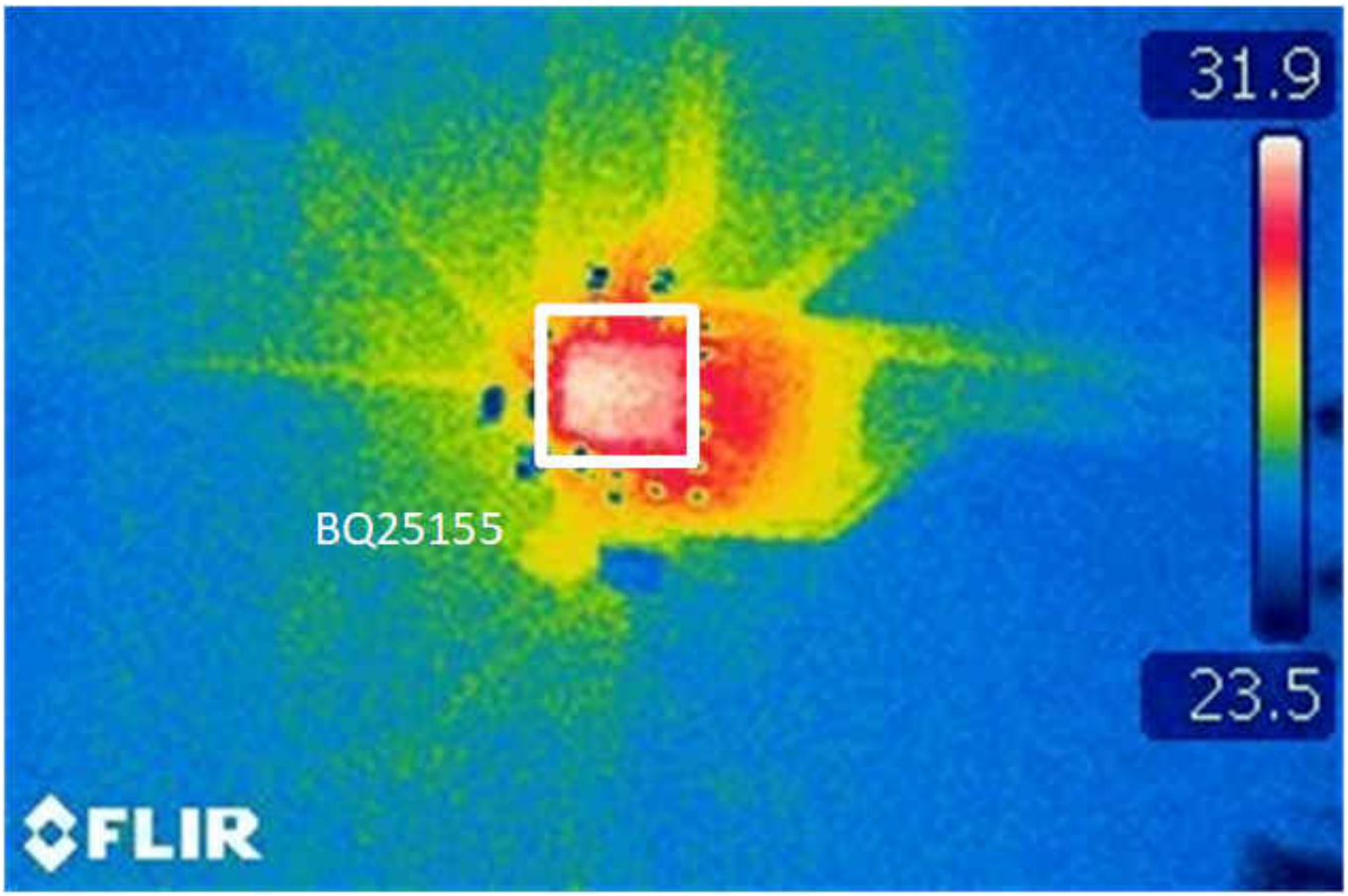

図 6-4. 4.6V 出力設計の BQ25619 の放熱性能

6.3 出力 5V の標準昇圧

3 番目のシステムは、ケース出力に標準の 5V を適用するベースラインのケースです。このシステムは、効率と発熱の両方について最適とは言えない性能を実現しています。

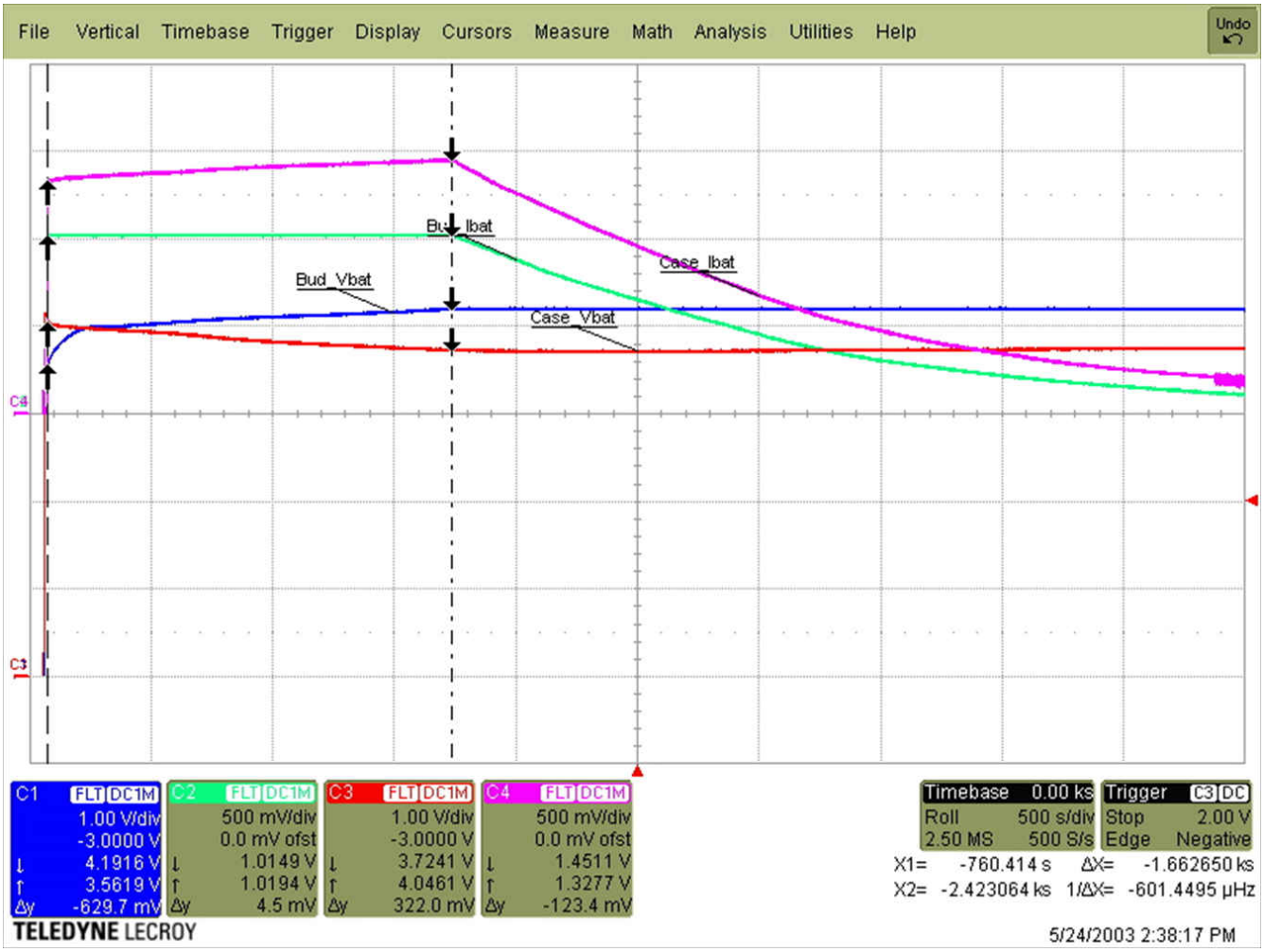

図 6-5. 5V 出力設計の充電サイクル

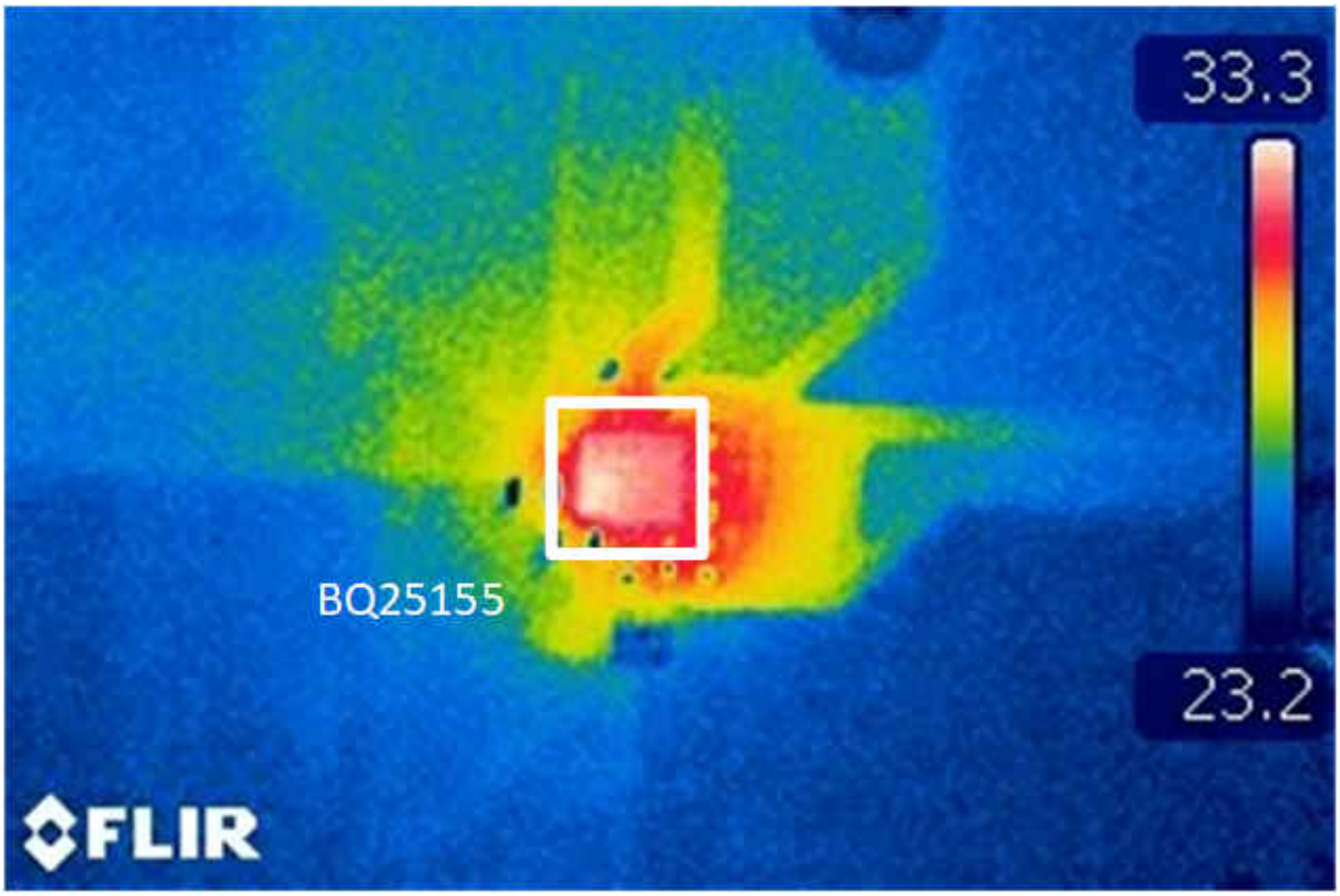

図 6-6. 5V 出力設計の放熱性能

7 まとめ

このテスト結果は、テキサス・インスツルメンツの設計にとって熱に関する大きな利点を示しています。これは、充電ケース内で BQ25619 のレギュレーション電圧を動的に調整することで行われています。このレポートでは、スペース効率の優れた 2 ピン・インターフェイスを使用して、これを実現する方法を示しました。TWS システムに関するテキサス・インスツルメンツのその他の資料については、こちらをクリックしてください。

8 回路図

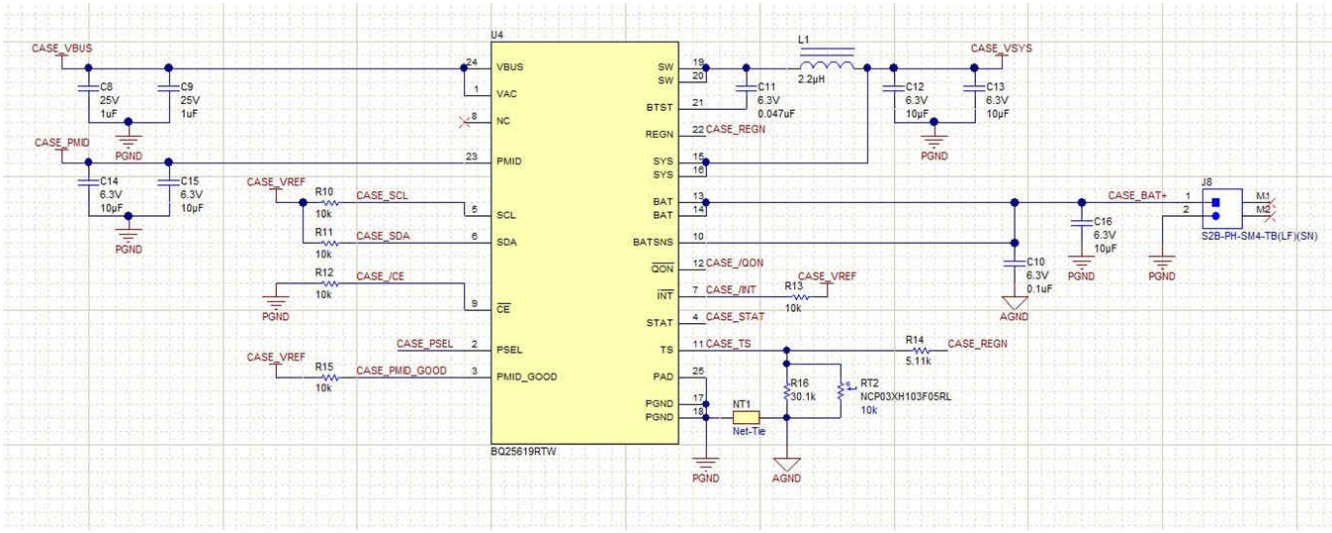

図 8-1. BQ25619 の回路図

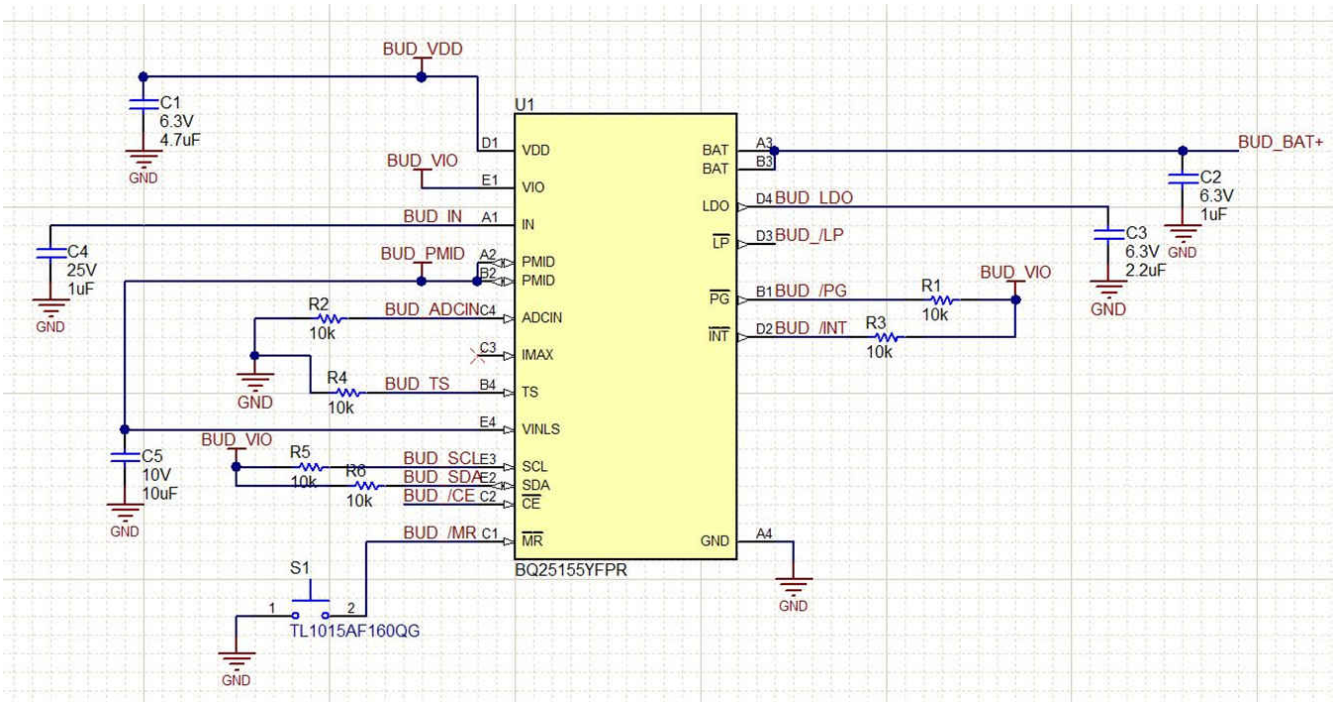

図 8-2. BQ25155 の回路図

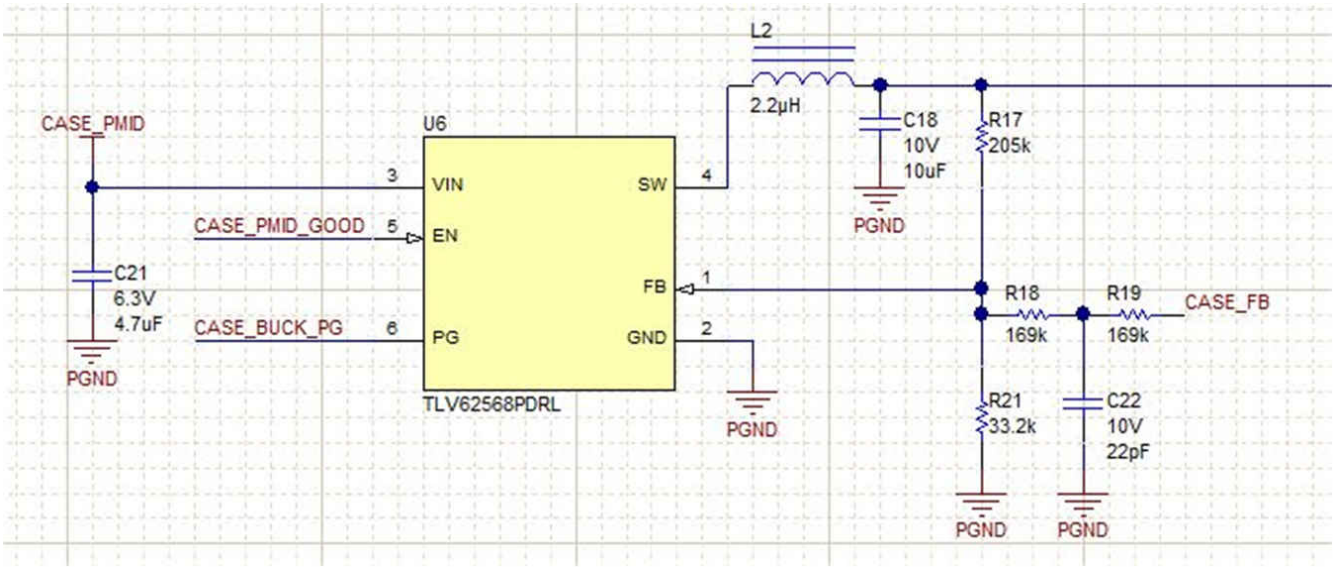

図 8-3. TLV62568P の回路図

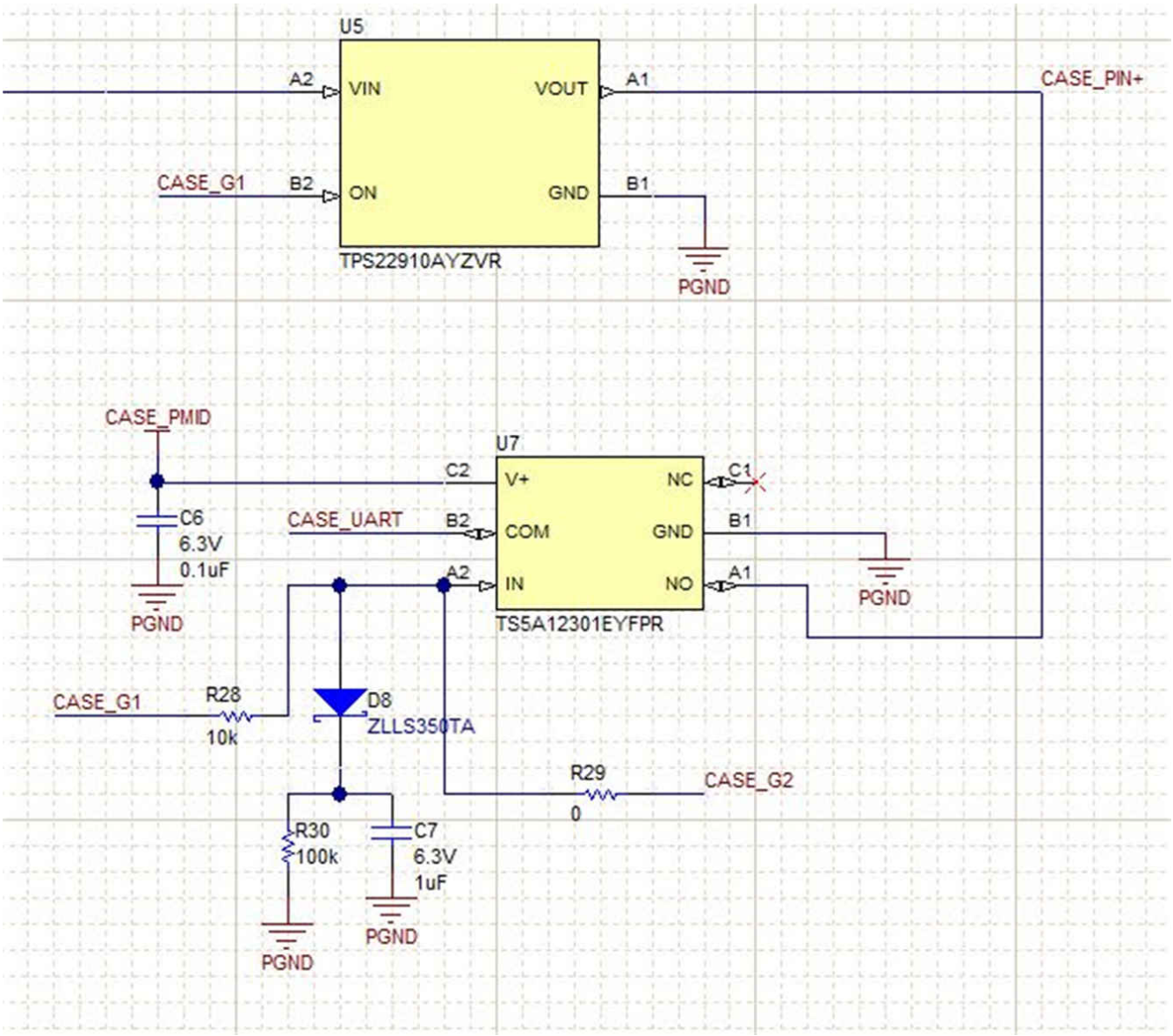

図 8-4. ケース・スイッチの回路図

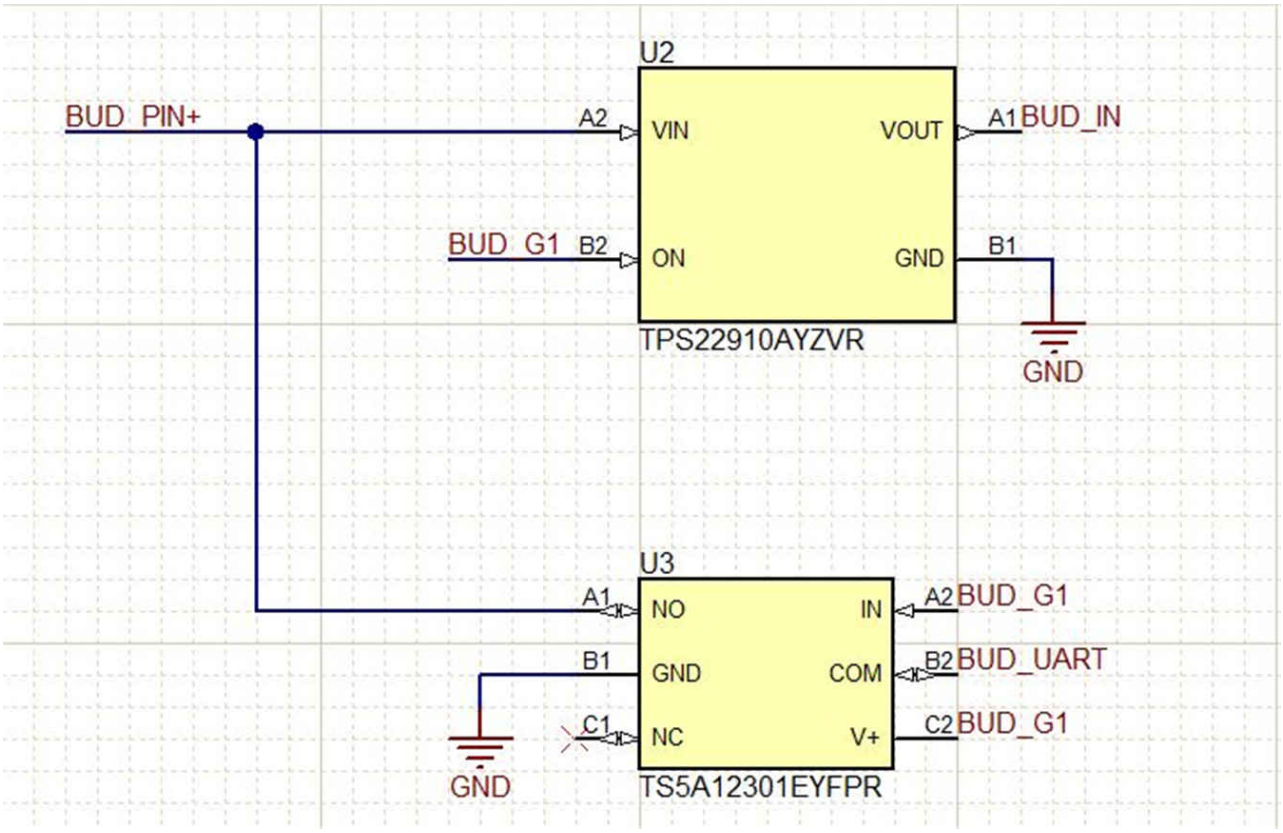

図 8-5. イヤホン・スイッチの回路図

9 PCB レイアウト

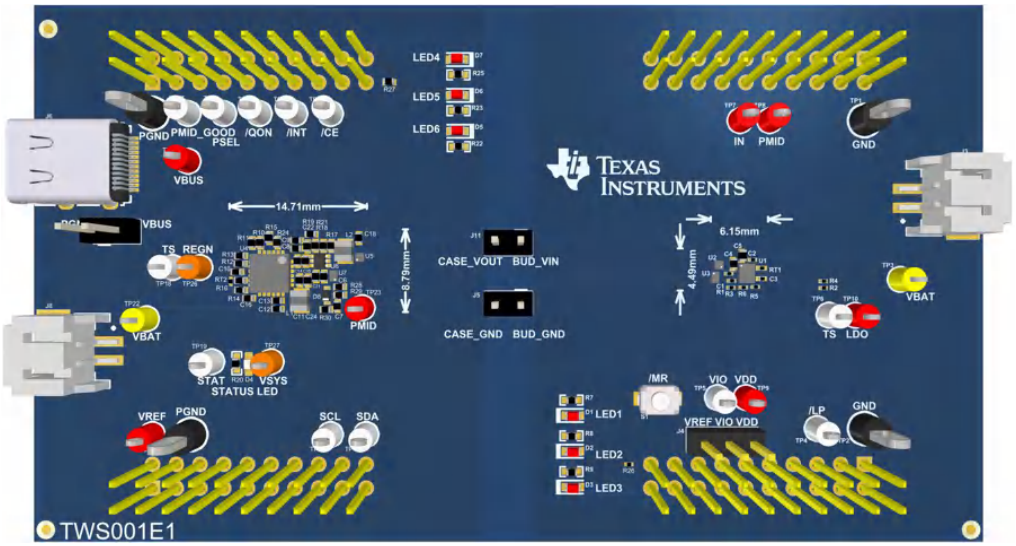

図 9-1. PCB の上面図

10 ソフトウェア

10.1 充電ケース main.c

```
/* --COPYRIGHT--,BSD
 * Copyright (c) 2016, Texas Instruments Incorporated
 * All rights reserved.
 *
 * 修正のあるなしに関わらず、ソース形式またはバイナリ形式での
 * 再配布および使用は、以下の条件を満たしている場合に
 * 許可されます。
 *
 * *   ソース・コードの再配布は上記の著作権表示、この条件のリスト
 *     および以下の免責事項を保持している必要があります。
 *
 * *   バイナリ形式での再配布は上記の著作権表示、この条件のリスト
 *     、および以下の免責事項を配布に付属のドキュメント
 *     および / またはその他の資料に再現する必要があります。
 *
 * *   テキサス・インスツルメンツまたはその貢献者のいずれの名前も
 *     特に事前の書面による許可なく、本ソフトウェアに由来の
 *     製品を宣伝または販売促進することはできません。
 *
 * 本ソフトウェアはその著作権保有者およびその貢献者により「現状のまま」提供され、
 * 明示あるいは黙示を問わず、これに限定されることなく
 * 商品性の黙示保証、特定目的への適合性に対して
 * 一切保証しないものとします。いかなる場合にも、著作権保有者または
 * 貢献者は、直接的、間接的、付随的、特別、拡大、懲罰的または結果的損害
 * （商品またはサービスの代替品の調達、使用、データ、または利益の逸失
 * または業務の中断を含むがこれに限りません）
 * の可能性を忠告されていたにもかかわらず本ソフトウェアの使用により生じた
 * 当該損害について、その原因を問わず
 * 契約上の責任、厳格責任、または不法行為責任（その他の過失を含む）
 * の法理に基づいて、一切の責任を負わないものとします。
 * --/COPYRIGHT--*/
/*
 * ======== main.c ========
*/
#include <string.h>
#include <stdint.h>
#include <inttypes.h>
#include <string.h>
#include <stdio.h>
#include "board_functions.h"
#include <stdlib.h>
#include "driverlib.h"
#include "StdI2C.h"
#include "BQ25150.h"
//#include "board_timer.h"
#include "USB_config/descriptors.h"
#include "USB_API/USB_Common/device.h"
#include "USB_API/USB_Common/usb.h"                 // USB 固有の関数
#include "USB_API/USB_CDC_API/UsbCdc.h"
#include "USB_app/usbConstructs.h"
#include "OLED/Oled_SSD1306.h"
#include "StdPollI2C.h"
//#include <Wire.h>
/*
 * your own board.
 */
#include "hal.h"
//#include "Energia.h"
//#include "pins_energia.h"
// 関数の宣言
uint8_t retInString(char* string);
void initTimer(void);
void setTimer_A_Parameters(void);
// イベントによって設定されたグローバル・フラグ
volatile uint8_t bCDCDataReceived_event = FALSE; // open rx 演算なくデータが rx された
// ことを示します
char str[50];
#define NUM_SAMPLES 8
#define clkspeed 3          // 24Mhz クロック選択
//#define clkspeed 1        // 8Mhz クロック選択
short samples[NUM_SAMPLES] ;
int sample_index = 0 ;
char str[50];
char cmdstring[5];
```

```
char cs2[3];
uint8_t response = 0;
//unsigned char buffer[10]={1,2,3,4,5,6,7,8,9,10};
volatile char wholeString[MAX_STR_LENGTH] = "";
volatile uint8_t modecounter = 0;
volatile uint8_t pluscounter = 0;
///Timer_A_initUpModeParam Timer_A_params = {0};
int i;
unsigned char* PRxData;                     // RX データへのポインタ
unsigned char RXByteCtr;
volatile unsigned char RxBuffer[128];       // 128 バイトの RAM を割り当てます
unsigned char* PTxData;                     // TX データへのポインタ
unsigned char TXByteCtr;
const unsigned char TxData[] =              // 転送するデータのテーブル
{
 0x11,
 0x22,
 0x33,
 0x44,
 0x55
};
volatile uint8_t Button_Released = 0;
unsigned int result;
const char* hexstring = "0xabcdef0";
int raddr;
int rdata;
char buf[5];
uint8_t uraddr;
uint8_t urdata;
uint16_t Err;
// 発信文字列を保持します
char outString[MAX_STR_LENGTH] = "";
uint8_t connectedflag = 0;
uint8_t echoflag = 1;
int ubtncounter = 0;
uint8_t RegValuesL = 0;
uint8_t RegValuesM = 0;
uint8_t RegValueMSB = 0;
uint8_t ADCCheck = 0;
uint32_t stobusf;
char vBat;
char vBatString;
uint32_t stobuf;
uint32_t stobuf1;
double stobuf2;
double PwmDuty;
uint8_t RegValues = 0;
double batteryVoltageCheck = 0;
//__delay_cycles(1000); 1000 = 100us
uint8_t rthex;
// トグル遅延を設定 / 宣言します
//uint16_t SlowToggle_Period = 20000 - 1;
//uint16_t FastToggle_Period = 1000 - 1;
//  ======== main ========
void main(void)
{
    WDT_A_hold(WDT_A_BASE); // ウォッチドッグ・タイマを停止します
    // USB API に必要な最低 Vcore 設定は PMM_CORE_LEVEL_2 です。
    PMM_setVCore(PMM_CORE_LEVEL_2);
    USBHAL_initPorts();           // ローパワー (出力 Low) の GPIOS を構成します
    USBHAL_initClocks(8000000 * clkspeed);   // クロックを構成します。MCLK=SMCLK=FLL=8MHz; ACLK=REFO=32kHz
    initTimer();            // LED トグルのタイマを準備します
    initI2C();
//  ======== UART Setup ========
    GPIO_setAsInputPinWithPullDownResistor(GPIO_PORT_P3,GPIO_PIN4); // P3.4 = Input With pulldown
    P3SEL = BIT3+BIT4;                          // P3.4,5 = USCI_A0 TXD/RXD
    UCA0CTL1 |= UCSWRST;                        // **ステート・マシンをリセットにします**
    UCA0CTL1 |= UCSSEL_2;                       // SMCLK
    UCA0BR0 = 6;                                // 1MHz 9600 (ユーザー・ガイドを参照)
    UCA0BR1 = 0;                                // 1MHz 9600
    UCA0MCTL = UCBRS_0 + UCBRF_13 + UCOS16;     // Modln UCBRSx=0, UCBRFx=0,
                                                // オーバー・サンプリング
    UCA0CTL1 &= ~UCSWRST;                       // **USCI ステート・マシンを初期化します**
//  ======== GPIO Setup ========
    GPIO_setAsOutputPin(LED_4);
    GPIO_setOutputLowOnPin(LED_4);
    GPIO_setAsInputPin(CASE_PMID_GOOD);
    GPIO_setAsInputPinWithPullUpResistor(CASE_INT);
    GPIO_setAsInputPin(CASE_BUCK_PG);
```

```
    GPIO_setAsInputPinWithPullUpResistor(CASE_START);
    GPIO_setAsOutputPin(CASE_FB);
    GPIO_setOutputLowOnPin(CASE_FB);
    GPIO_setAsOutputPin(CASE_PSEL);
    GPIO_setOutputLowOnPin(CASE_PSEL);
    GPIO_setAsOutputPin(CASE_QON);
    GPIO_setOutputHighOnPin(CASE_QON);
    GPIO_setAsOutputPin(CASE_CE);
    GPIO_setOutputLowOnPin(CASE_CE);
    GPIO_setAsOutputPin(CASE_G1);
    GPIO_setOutputLowOnPin(CASE_G1);
    GPIO_setAsOutputPin(LED_5);
    GPIO_setOutputLowOnPin(LED_5);
//    GPIO_setAsOutputPin(LED_6);
//    GPIO_setOutputLowOnPin(LED_6);
//  ======== PWM Setup ========
    P2DIR |= BIT5; //ピン 2.5 を出力方向に設定します。
    P2SEL |= BIT5; //ピン 2.5 を当社の PWM 出力として選択します。
    TA2CCTL2 = OUTMOD_7;
    TA2CCR0 = 40; //タイマ A0 Capture/Compare 0 レジスタの期間を 40us に設定します。
    TA2CCR2 = 14; //電源がオンの時間 (ミリ秒)、デフォルト 14 = 35%, ~= 4.5v OUTPUT
    TA2CTL = (TASSEL_2 | MC_1); //TASSEL_2 は SMCLK をクロック・ソースとして選択し、MC_1 は TA0CCR0 の値をカウントするよう通知します。
 //  ======== BQ25619 Register Setup ========
    StdI2C_P_TX_Single(BQ25619_ADDR, BQ25619_REG01, 0x3A, &Err); // BST_CONFIG = 1, Boost mode Enable, REG01 = 01h, value = 0x3A
    StdI2C_P_RX_Single(BQ25619_ADDR, BQ25619_REG01, &RegValues, &Err);
    StdI2C_P_TX_Single(BQ25619_ADDR, BQ25619_REG05, 0x8E, &Err); // ウォッチドッグをディスエーブルにします
    StdI2C_P_RX_Single(BQ25619_ADDR, BQ25619_REG05, &RegValues, &Err);
    StdI2C_P_TX_Single(BQ25619_ADDR, BQ25619_REG06, 0xC6, &Err); // 昇圧電圧を 0xE6 =  5V に設定し、0xC6 の場合は 4.6V
    StdI2C_P_RX_Single(BQ25619_ADDR, BQ25619_REG06, &RegValues, &Err);
    GPIO_setOutputHighOnPin(LED_4);
    waitms(500);
    GPIO_setOutputLowOnPin(LED_4);
    waitms(500);
//  ======== Ready while loop ========
    //この while ループは割り込みをディスエーブルした状態でプログラムを保持します。これによりイヤホンと同期できます
    //BQ_START ピン 4.3 をグランドに短絡させてループを終了します
    while(GPIO_getInputPinValue(CASE_START) == 1)
    {
        GPIO_toggleOutputOnPin(LED_5);
        __delay_cycles(3000000);
        GPIO_toggleOutputOnPin(LED_5);
        __delay_cycles(3000000);
        GPIO_toggleOutputOnPin(LED_5);
        __delay_cycles(3000000);
        GPIO_toggleOutputOnPin(LED_5);
        __delay_cycles(15000000);
    }
    GPIO_setOutputLowOnPin(LED_5);
//  ======== Interrupt Enables ========
    __enable_interrupt();  // 割り込みをグローバルにイネーブルします
    UCA0IE |= UCRXIE;
//  ======== Main while loop ========
    //通信サイクル時間を調整できます
    while(1)
    {
        GPIO_toggleOutputOnPin(LED_4);
//        __delay_cycles(24000000); //delay 1s
            __delay_cycles(120000000); //delay 5s
//        __delay_cycles(240000000); //delay 10s
//        __delay_cycles(1440000000); //delay 60s
        if(batteryVoltageCheck >= 3.8){
            __delay_cycles(120000000); //delay 5s
        }
        if(batteryVoltageCheck >= 3.85){
            __delay_cycles(240000000); //delay 10s
        }
        if(batteryVoltageCheck >= 3.95){
            __delay_cycles(240000000); //delay 10s
        //    __delay_cycles(240000000); //delay 10s
        //    __delay_cycles(240000000); //delay 10s
        }
        if(batteryVoltageCheck >= 4.05){
            __delay_cycles(1440000000); //delay 60s
        //    __delay_cycles(1440000000); //delay 60s
        }
        if(batteryVoltageCheck >= 4.19){
```

```
            __delay_cycles(1440000000); //delay 60s
            __delay_cycles(1440000000); //delay 60s
            __delay_cycles(1440000000); //delay 60s
            __delay_cycles(1440000000); //delay 60s
//            __delay_cycles(1440000000); //delay 60s
//            __delay_cycles(1440000000); //delay 60s
//            __delay_cycles(1440000000); //delay 60s
        }
        GPIO_setOutputHighOnPin(CASE_G1);      //通信モードに入ります
    }
}
/*
 * ======== TIMER1_A0_ISR ========
 */
#if defined(__TI_COMPILER_VERSION__) || (__IAR_SYSTEMS_ICC__)
#pragma vector=TIMER0_A0_VECTOR
__interrupt void TIMER0_A0_ISR (void)
#elif defined(__GNUC__) && (__MSP430__)
void __attribute__ ((interrupt(TIMER0_A0_VECTOR))) TIMER0_A0_ISR (void)
#else
#error Compiler not found!
#endif
{
    // wake on CCR0 count match
    TA0CCTL0 = 0;
    __bic_SR_register_on_exit(LPM0_bits|GIE);
}
//  ======== UART RX Interrupt ========
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=USCI_A0_VECTOR
__interrupt void USCI_A0_ISR(void)
#elif defined(__GNUC__)
void __attribute__ ((interrupt(USCI_A0_VECTOR))) USCI_A0_ISR (void)
#else
#error Compiler not supported!
#endif
{
  switch(__even_in_range(UCA0IV,4))
  {
  case 0:break;                             // Vector 0 - no interrupt
  case 2:                                   // Vector 2 - RXIFG
      GPIO_toggleOutputOnPin(LED_5);
      RegValueMSB = UCA0RXBUF;              //受信バイトを RegValueMSB に割り当てます
      if (RegValueMSB == 0xD5){ //充電完了バイトを確認します
          StdI2C_P_TX_Single(BQ25619_ADDR,0x01 , 0x1A, &Err);//充電完了時に BQ25619 昇圧モードをディスエーブ
ルにします
      }
      else{
      stobuf1 = RegValueMSB * 6;
      stobuf2 = (double)stobuf1 / 256;       //Vbat を計算します
      batteryVoltageCheck = stobuf2;
      if(batteryVoltageCheck <= 4.05){
          //PwmDuty = 40-((stobuf2-2.6)/.095);
          PwmDuty =((7.606829268 - (1.434146341*(stobuf2 + .33)))*(40/3.3)); //~= .3v headroom for
longer intervals
      }
      // stobuf2  is holder for Vbat
      //PwmDuty の範囲は 0 = 0% = 5.3V から 40 = 100% = 3V です
      if (PwmDuty >= 14 && PwmDuty <= 40){ //PwmDuty を制限し、電圧を 4.5 を超えて上げないでください
          TA2CCR2 = PwmDuty; //PwmDuty サイクルを割り当てます
      }
      else{
      TA2CCR2 = 14;
      }
      __delay_cycles(40000);
      GPIO_setOutputLowOnPin(CASE_G1);      //パワー・モードに入ります
      }
    break;
  case 4:break;                             // Vector 4 - TXIFG
  default: break;
  }
}
//Released_Version_5_10_00_17
```

10.2 イヤホン main.c

```
/* --COPYRIGHT--,BSD
 * Copyright (c) 2016, Texas Instruments Incorporated
 * All rights reserved.
 *
 * 修正のあるなしに関わらず、ソース形式またはバイナリ形式での
 * 再配布および使用は、以下の条件を満たしている場合に
 * 許可されます。
 *
 * *   ソース・コードの再配布は上記の著作権表示、この条件のリスト
 *     および以下の免責事項を保持している必要があります。
 *
 * *   バイナリ形式での再配布は上記の著作権表示、この条件のリスト
 *     、および以下の免責事項を配布に付属のドキュメント
 *     および / またはその他の資料に再現する必要があります。
 *
 * *   テキサス・インスツルメンツまたはその貢献者のいずれの名前も
 *     特に事前の書面による許可なく、本ソフトウェアに由来の
 *     製品を宣伝または販売促進することはできません。
 *
 * 本ソフトウェアはその著作権保有者およびその貢献者により「現状のまま」提供され、
 * 明示あるいは黙示を問わず、これに限定されることなく
 * 商品性の黙示保証、特定目的への適合性に対して
 * 一切保証しないものとします。いかなる場合にも、著作権保有者または
 * 貢献者は、直接的、間接的、付随的、特別、拡大、懲罰的または結果的損害
 * （商品またはサービスの代替品の調達、使用、データ、または利益の逸失
 * または業務の中断を含むがこれに限りません）
 * の可能性を忠告されていたにもかかわらず本ソフトウェアの使用により生じた
 * 当該損害について、その原因を問わず
 * 契約上の責任、厳格責任、または不法行為責任（その他の過失を含む）
 * の法理に基づいて、一切の責任を負わないものとします。
 * --/COPYRIGHT--*/
// ======== main.c ========
#include <string.h>
#include <stdint.h>
#include <inttypes.h>
#include <string.h>
#include <stdio.h>
#include "board_functions.h"
#include <stdlib.h>
#include <msp430.h>
#include "driverlib.h"
#include "StdI2C.h"
#include "BQ25150.h"
//#include "board_timer.h"
#include "USB_config/descriptors.h"
#include "USB_API/USB_Common/device.h"
#include "USB_API/USB_Common/usb.h"                    // USB 固有の関数
#include "USB_API/USB_CDC_API/UsbCdc.h"
#include "USB_app/usbConstructs.h"
#include "OLED/Oled_SSD1306.h"
#include "StdPollI2C.h"
#include "hal.h"
// 関数の宣言
uint8_t retInString(char* string);
void initTimer(void);
void setTimer_A_Parameters(void);
// イベントによって設定されたグローバル・フラグ
volatile uint8_t bCDCDataReceived_event = FALSE; // open rx 演算なくデータが rx された
// ことを示します
char str[50];
#define NUM_SAMPLES 8
#define clkspeed 3          // 24Mhz クロック選択
//#define clkspeed 1        // 8Mhz クロック選択
short samples[NUM_SAMPLES] ;
int sample_index = 0 ;
char str[50];
char cmdstring[5];
char cs2[3];
uint8_t response = 0;
//unsigned char buffer[10]={1,2,3,4,5,6,7,8,9,10};
volatile char wholeString[MAX_STR_LENGTH] = "";
volatile uint8_t modecounter = 0;
volatile uint8_t pluscounter = 0;
///Timer_A_initUpModeParam Timer_A_params = {0};
int i;
unsigned char* PRxData;                        // RX データへのポインタ
```

```
unsigned char RXByteCtr;
volatile unsigned char RxBuffer[128];       // 128 バイトの RAM を割り当てます
unsigned char* PTxData;                     //  TX データへのポインタ
unsigned char TXByteCtr;
const unsigned char TxData[] =              // 転送するデータのテーブル
{
 0x11,
 0x22,
 0x33,
 0x44,
 0x55
};
volatile uint8_t Button_Released = 0;
unsigned int result;
const char* hexstring = "0xabcdef0";
int raddr;
int rdata;
char buf[5];
uint8_t uraddr;
uint8_t urdata;
uint16_t Err;
// 発信文字列を保持します
char outString[MAX_STR_LENGTH] = "";
uint8_t connectedflag = 0;
uint8_t echoflag = 1;
int ubtncounter = 0;
uint8_t RegValueLSB = 0;
uint8_t RegValueMSB = 0;
uint8_t VbatMSB = 0;
uint8_t ADCCheck = 0;
uint32_t stobusf;
uint8_t ADCcount = 0;
uint8_t VinReadA = 0;
uint8_t VinReadB = 0;
uint8_t VinReadC = 0;
uint8_t VinLow = 0;
double vIn = 0;
char vBat[];
//uint8_t RegValueM = 0;
//uint8_t RegValueL = 0;
uint32_t stobuf;
uint32_t stobuf1;
double stobuf2;
uint8_t RegValues = 0;
//__delay_cycles(1000); 1000 = 100us
uint8_t rthex;
// トグル遅延を設定 / 宣言します
//uint16_t SlowToggle_Period = 20000 - 1;
//uint16_t FastToggle_Period = 1000 - 1;
/*
 * ======== main ========
 */
void main(void)
{
    WDT_A_hold(WDT_A_BASE); // ウォッチドッグ・タイマを停止します
    // USB API に必要な最低 Vcore 設定は PMM_CORE_LEVEL_2 です。
    PMM_setVCore(PMM_CORE_LEVEL_2);
    USBHAL_initPorts();           // ローパワー (出力 Low) の GPIOS を構成します
    USBHAL_initClocks(8000000 * clkspeed);   // クロックを構成します。MCLK=SMCLK=FLL=8MHz; ACLK=REFO=32kHz
    //USBHAL_initClocks(24000000);
    initTimer();            // LED トグルのタイマを準備します
    //USB_setup(TRUE, TRUE); // USB およびイベントを初期化し、ホストが存在する場合は、接続します
    initI2C();
//  ======== UART Setup ========
    P3SEL = BIT3+BIT4;                          // P3.4,5 = USCI_A0 TXD/RXD
    __delay_cycles(20000);
    UCA0CTL1 |= UCSWRST;                        // **ステート・マシンをリセットにします**
    UCA0CTL1 |= UCSSEL_2;                       // SMCLK
    UCA0BR0 = 6;                                // 1MHz 9600 (ユーザー・ガイドを参照)
    UCA0BR1 = 0;                                // 1MHz 9600
    UCA0MCTL = UCBRS_0 + UCBRF_13 + UCOS16;     // Modln UCBRSx=0, UCBRFx=0,
                                                // オーバー・サンプリング
    UCA0CTL1 &= ~UCSWRST;                       // **USCI ステート・マシンを初期化します**
//  ======== GPIO Setup ========
    //LED Setup
    GPIO_setAsOutputPin(LED_1);
    GPIO_setOutputLowOnPin(LED_1);
    GPIO_setAsOutputPin(LED_2);
    GPIO_setOutputLowOnPin(LED_2);
```

```
    GPIO_setAsOutputPin(LED_3);
    GPIO_setOutputLowOnPin(LED_3);
    //GPIO のセットアップ
    GPIO_setAsInputPinWithPullUpResistor(BQ_INT);
    GPIO_setAsInputPin(BQ_PG);
    GPIO_setAsInputPinWithPullUpResistor(BQ_START);
    GPIO_setAsOutputPin(BQ_CE);
    GPIO_setOutputLowOnPin(BQ_CE);
    GPIO_setAsOutputPin(BQ_LP);
    GPIO_setOutputHighOnPin(BQ_LP);
    GPIO_setAsOutputPin(BQ_G1);
    GPIO_setOutputLowOnPin(BQ_G1);
    GPIO_toggleOutputOnPin(LED_1);
    waitms(500);
    GPIO_toggleOutputOnPin(LED_1);
    waitms(500);
    GPIO_toggleOutputOnPin(LED_1);
    waitms(500);
    GPIO_toggleOutputOnPin(LED_1);
//  ======== BQ25155 Register Setup ========
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_MASK0, 0x5E, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_MASK1, 0xBF, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_MASK2, 0xF1, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_MASK3, 0x77, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_VBAT_CTRL, 0x3C, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ICHG_CTRL, 0x50, &Err);//Ichg を 200mA に設定し、0xA0 には
200mA、0x50 には 100mA、0x20 には 40mA を設定します
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_PCHRGCTRL, 0x02, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_TERMCTRL, 0x14, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_BUVLO, 0x00, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_CHARGERCTRL0, 0x92, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_CHARGERCTRL1, 0xC2, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ILIMCTRL, 0x06, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_MRCTRL, 0x2A, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ICCTRL0, 0x10, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ICCTRL1, 0x00, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ICCTRL2, 0x40, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCCTRL0, 0x40, &Err); //0x58 は ADC を 3ms conv の連続に設
定し、0x40 は 24ms conv の連続に設定します。
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCCTRL1, 0x00, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCALARM_COMP1_M, 0x23, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCALARM_COMP1_L, 0x20, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCALARM_COMP2_M, 0x38, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCALARM_COMP2_L, 0x90, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCALARM_COMP3_M, 0x00, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADCALARM_COMP3_L, 0x00, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_ADC_READ_EN, 0xFE, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_TS_FASTCHGCTRL, 0x34, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_TS_COLD, 0x7C, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_TS_COOL, 0x6D, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_TS_WARM, 0x38, &Err);
    StdI2C_TX_Single(BQ25150_ADDR, BQ25150_TS_HOT, 0x28, &Err);
//  ======== Ready while loop ========
    //この while ループは割り込みをディスエーブルした状態でプログラムを保持します。これによりケースと同期できます
    //BQ_START ピン 3.7 をグランドに短絡させ、ループを終了します
    while(GPIO_getInputPinValue(BQ_START) == 1) //割り込みをイネーブルにする前に開始状態で待機します
    {
        GPIO_toggleOutputOnPin(LED_2);
        __delay_cycles(3000000);
        GPIO_toggleOutputOnPin(LED_2);
        __delay_cycles(3000000);
        GPIO_toggleOutputOnPin(LED_2);
        __delay_cycles(3000000);
        GPIO_toggleOutputOnPin(LED_2);
        __delay_cycles(24000000);
    }
    GPIO_setOutputLowOnPin(LED_2);
//  ======== Interrupt Enables ========
    __enable_interrupt();  // 割り込みをグローバルにイネーブルします
    GPIO_enableInterrupt(BQ_INT);
    GPIO_selectInterruptEdge(BQ_INT, GPIO_HIGH_TO_LOW_TRANSITION);
    GPIO_clearInterrupt(BQ_INT);
//  ======== Main while loop ========
    //Vbat を読み取り、ケースが Vin を Low に駆動するのを待機します
    while(1)
    {
         StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_ADCDATA_VBAT_M, &VbatMSB, &Err); //Vbat を読み取りま
す
         StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_ADCDATA_VIN_M, &RegValueMSB, &Err); //Vin を読み取り
```

```
ます
        GPIO_toggleOutputOnPin(LED_1);
        __delay_cycles(12000000);
    }
}
/*
 * ======== TIMER1_A0_ISR ========
 */
#if defined(__TI_COMPILER_VERSION__) || (__IAR_SYSTEMS_ICC__)
#pragma vector=TIMER0_A0_VECTOR
__interrupt void TIMER0_A0_ISR (void)
#elif defined(__GNUC__) && (__MSP430__)
void __attribute__ ((interrupt(TIMER0_A0_VECTOR))) TIMER0_A0_ISR (void)
#else
#error Compiler not found!
#endif
{
    // wake on CCR0 count match
    TA0CCTL0 = 0;
    __bic_SR_register_on_exit(LPM0_bits|GIE);
}
//  ======== BQ25155 Interrupt ========
#if defined(__TI_COMPILER_VERSION__) || defined(__IAR_SYSTEMS_ICC__)
#pragma vector=PORT1_VECTOR
__interrupt
#elif defined(__GNUC__)
__attribute__((interrupt(PORT1_VECTOR)))
#endif
void Port_1(void)
{
    switch(__even_in_range(P1IV,0x03))
    {
    case P1IV_P1IFG3:
        StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_STAT0, &RegValueMSB, &Err); //STAT0 REG を読み取ります
        RegValueMSB &= 0x01; // VIN_PGOOD_STAT がアサートされているか確認します
        if(RegValueMSB == 0x00){ //VIN_PGOOD_STAT がアサートされていない場合、VIN が Low に駆動されているか確認し
ます
//                __delay_cycles(6000000);
//
//                StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_ADCDATA_VIN_M, &RegValueMSB,
&Err); //Vin を読み取ります
//                stobuf = RegValueMSB;
//                stobuf1 = stobuf * 6;
//                vIn = (double)stobuf1 / 256;
          ADCcount = 0;
          VinLow = 0;
          VinReadA = 5;
          VinReadB = 5;
          VinReadC = 5;
          while(ADCcount < 85 && VinLow == 0){
            __delay_cycles(120000);
            StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_ADCDATA_VIN_M, &RegValueMSB, &Err); //VIN を読み
取ります
            stobuf = RegValueMSB;
            stobuf1 = stobuf * 6;
            vIn = (double)stobuf1 / 256;
            VinReadC = VinReadB;
            VinReadB = VinReadA;
            VinReadA = vIn;
            ADCcount++;
            if(VinReadA < 3 && VinReadB < 3 && VinReadC < 3){
                VinLow = 1;
                VinReadA = 0;
                VinReadB = 0;
                VinReadC = 0;
                vIn = 0;
            }
          }
                if(VinLow == 1){//Vin が 3V 未満の場合、通信プロセスを開始します
                    StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_STAT0, &RegValueMSB, &Err); //STAT0
REG を読み取ります
                    RegValueMSB &= 0x20; // CHARGE_DONE_STAT がアサートされているか確認します
//                    RegValueMSB = 0x20; //コメント解除して、テストのため充電完了ビットをアサートします
                    if(RegValueMSB){                        // CHARGE_DONE_STAT が transmit 0xD5,
0xD5 = Vbat = 5v にアサートされている場合、分かりやすくするため、これは任意の充電完了ビットとして選択されます
                        GPIO_setOutputHighOnPin(BQ_G1);     // 通信モードに入ります
                        __delay_cycles(4000);
                        while (!(UCA0IFG&UCTXIFG));         // USCI_A0 TX バッファの準備は完了していますか?
                        UCA0TXBUF = 0xD5;                   // TX -> 0xD5 = 5V、充電範囲外
```

```
                        __delay_cycles(4000);
                        GPIO_toggleOutputOnPin(LED_3);
                        GPIO_setOutputLowOnPin(BQ_G1);      //パワー・モードに入ります
                        GPIO_toggleOutputOnPin(LED_2);
                        __delay_cycles(4000000);
                        GPIO_toggleOutputOnPin(LED_2);
                        __delay_cycles(4000000);
                        GPIO_setOutputHighOnPin(LED_2);
                    }
                    else{
                    stobuf = VbatMSB;
                    stobuf1 = stobuf * 6;
                    stobuf2 = (double)stobuf1 / 256; //stobuf2 = Vbat の 2 倍
                    GPIO_setOutputHighOnPin(BQ_G1);    // 通信モードに入ります
                    __delay_cycles(4000);
                    while (!(UCA0IFG&UCTXIFG));        // USCI_A0 TX バッファの準備は完了していますか?
                    UCA0TXBUF = VbatMSB;               // TX -> vBat MSB
                    __delay_cycles(4000);
                    GPIO_toggleOutputOnPin(LED_3);
                    GPIO_setOutputLowOnPin(BQ_G1);     //パワー・モードに入ります
                    }
                }
        }
        //すべての割り込みフラグをクリアします
        StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_FLAG0, &RegValues, &Err);
        StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_FLAG1, &RegValues, &Err);
        StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_FLAG2, &RegValues, &Err);
        StdI2C_P_RX_Single(BQ25150_ADDR, BQ25150_FLAG3, &RegValues, &Err);
        GPIO_clearInterrupt(BQ_INT);
        break;
    default:
        break;
    }
}
//Released_Version_5_10_00_17
```

11 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 * (June 2022) to Revision A (May 2023)	Page
• 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

重要なお知らせと免責事項

TI 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、TI 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 TI 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 TI 製品を使用するアプリケーションの開発の目的でのみ、TI 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。TI 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、TI およびその代理人を完全に補償するものとし、TI は一切の責任を拒否します。

TI の製品は、TI の販売条件、または ti.com やかかる TI 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。TI がこれらのリソースを提供することは、適用される TI 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、TI 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