PSoC™ 1 の I²C 入門

About this document

関連製品ファミリ

CY8C21x23、21x34、21x45、22x45、23x33、24x23A、 24x33、24x94、27x43、28xxx、29x66

ソフトウェアバージョン

PSoC™ Designer 5.4

Scope and purpose

本アプリケーション ノート (AN50987) では、I²C 規格の概要と PSoC™ 1 デバイスが I²C 通信を処理する方法について説明します。本アプリケーション ノートをお読みいただくことで、I²C の仕組み、PSoC™ 1 内で実装する方法、また設計に適切なユーザー モジュールの選択方法を理解できます。サンプル プロジェクトでは、バス上の他の I²C デバイスと通信するために PSoC™ 1 を I²C マスター／スレーブとしてコンフィギュレーションする方法を示します。

Table of contents

Table of contents

1 はじめに

インター インテグレーテッド サーキット (IIC または I²C) は、Phillips Semiconductor (現 NXP) 社によって開発された、一般的なチップ ツー チップ シリアル通信規格です。I²C は、IC が同一のプリント基板 (PCB) 上で通信する簡単な方法を提供します。I²C は、ピン 2 本と最小限の外部コンポーネントだけを必要とする、単純な物理層で構成されています。I²C が SPI のような通信規格と比べて優れている 1 つの点は、I²C はデバイス間の簡単でエラーのない通信を可能にする通信プロトコルを内蔵していることです。

サイプレス PSoC™ 1 デバイスは、設計に I²C を実装する上で、いくつかの選択肢を提供します。これらの選択肢は、PSoC™ Designer™統合開発環境 (IDE) に含まれているユーザー モジュール (UM) の形で提供されます。本アプリ
ケーション ノートでは、PSoC™ 1 デバイスが I²C を処理する方法をご理解いただけるように、まず I²C の基礎について説明します。I²C の基礎をすでにご理解して頂いている方は **PSoC™ 1 の I2C** へお進みください。I²C 設計の問題の解決法をお探しの場合には、**特別な I²C 注意事項**へお進みください。

本アプリケーション ノートは、PSoC™ 1 デバイスおよび PSoC™ Designer IDE の経験者を対象としています。

2 I²C の基礎

2.1 物理層

Figure 1 に、I²C デバイスがどのようにして I²C バスに接続するかを示しています。I²C バスは、シリアル データ ライン (SDA) とシリアル クロック ライン (SCL) の 2 つの物理ラインで構成されています。バス上のすべてのデバイスはこれら 2 本の物理ラインに接続しなければなりません。唯一必要な外部ハードウェアは、SDA および SCL ライン上の、V_{DD} (ハイ レール) に接続するプルアップ抵抗です。詳細については、**プルアップ抵抗**をご参照ください。

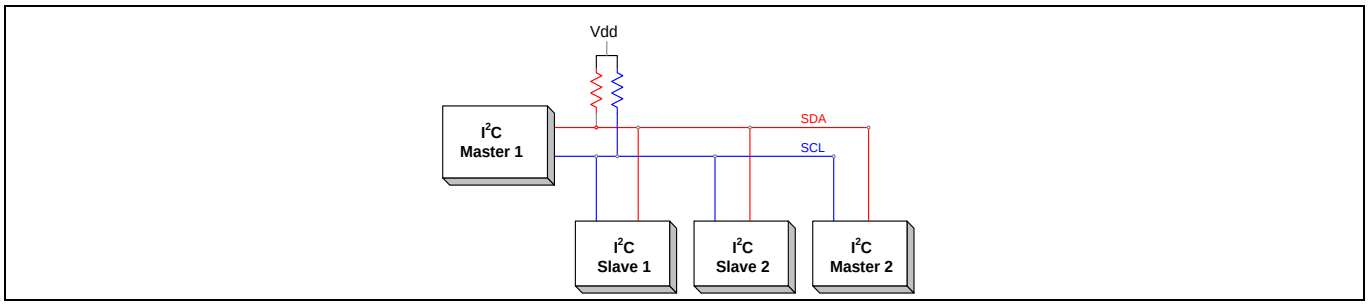

Figure 1 一般的な I²C バス

SDA および SCL は双方向ラインです。データはマスターからスレーブへ、またスレーブからマスターへ通信されます。SDA および SCL は、オープン ドレイン駆動モードに対応し、LOW に駆動されます。デバイスは、ラインを論理 LOW に駆動するか、またはハイインピーダンス状態にすることができます。そのため、ラインを HIGH に駆動できるデバイスはありません。この構成により、バス上の電源とグランドの短絡を回避できます。**Figure 2** をご参照ください。HIGH 論理レベルは、SDA および SCL におけるプルアップ抵抗によって生成されます。

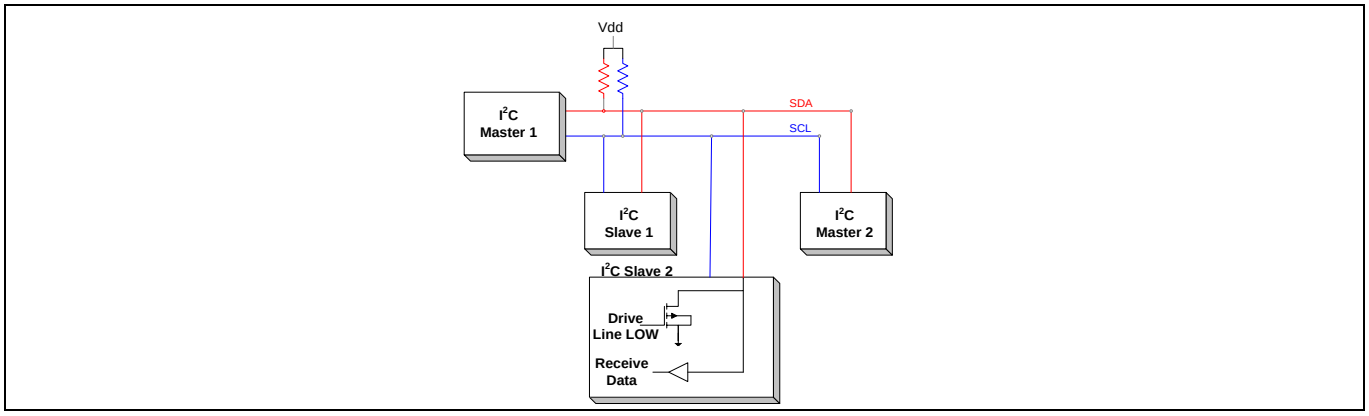

Figure 2 オープンドレインによる LOW 駆動のピン構成

I²C バスにおけるデバイスにはマスター スレーブ関係が成り立ちます。マスターはバス上ですべてのデータ転送を開始し、すべてのクロック信号を生成します。各スレーブは固有のアドレスがあります。マスターはデータ転送を開始する前に、まず、特定のスレーブのアドレスを指定し、アクノリッジ信号を受信する必要があります。同一のバスに、複数のマスターとスレーブが共存できます。I²C バスは様々な周波数で動作します。通常は 100kHz と 400kHz の周波数です。I²C 仕様によって 1MHz と 3.4MHz の高周波数も許可されています。

I²C バスは、デバイスによって異なる電圧レベルで動作します。2 個のデバイスの通信が可能であるか否かを判断するためには、それぞれのデータシートを参照し、論理レベルに互換性があるか確認してください。

2 個のデバイスの電圧と論理レベルに互換性がない場合は、I²C 仕様は、異なる電圧レベルのバスを接続するための解決策を提供します。

2.2 プロトコル層

I²C のプロトコル層を理解することは、このデジタル通信技術をマスターする上での次なる重要なステップです。

それぞれの I²C トランザクションは、START (または反復 START)、アドレス、データ、および STOP で構成されます。

2.3 START または反復 START

マスター デバイスがクロック ラインを制御します。つまり、マスターがバス上におけるすべての通信を開始します。バスの制御権を得て、トランザクションを開始するためには、マスターはまず、START 条件を送信します。**Figure 3** を参照してください。

START 条件はマスターがバスの制御権を得て、アドレスを送信する準備ができていることをバス上のすべてのデバイスに知らせます。バスはビジー状態にあると見なされるため、他のマスターが転送を開始するためには、STOP 条件によってバスが解放されるまで待つ必要があります。

START = SCL が HIGH の時に SDA が HIGH から LOW へ移行

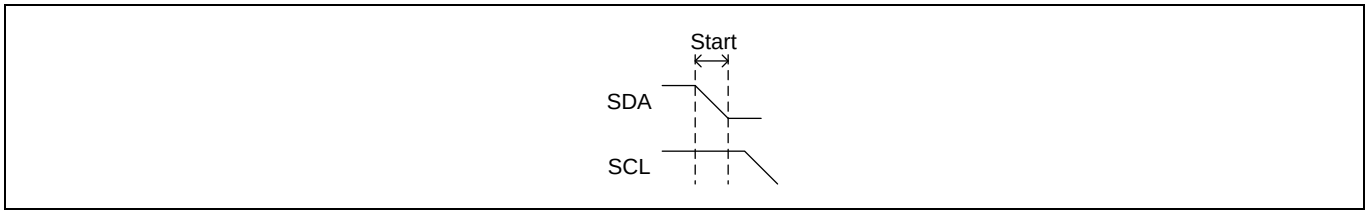

Figure 3 START 条件

反復 START 条件は物理的には START 条件と同じです。これは、マスターがバスを制御したままで、バスがフリーではないことを通知します。

2.4 アドレス

アドレスは、START 条件の後にマスターが最初に送信するデータ バイトです。各スレーブ デバイスは固有のアドレスがあります。ほとんどの I²C アドレスは 7 ビット長です。それに読み出し／書き込み (R/W) ビットを追加すると、完全な 8 ビット バイトになります。この読み出し／書き込み (R/W) ビットは、トランザクションの残り部分の通信方向を示します。**Figure 4** を参照してください。読み出しビットは、マスターがスレーブからデータを読み出したいことを示します。書き込みビットは、マスターがスレーブにデータを書き込みたいことを示します。すべてのアドレスとデータ バイトは、最上位ビット (MSB) から送信されます。

例えば、7 ビット アドレスが 0x20 の場合、読み出し用の完全な 8 ビット バイトは 0x40 で、書き込み用の完全な 8 ビットバイトは 0x41 となります。

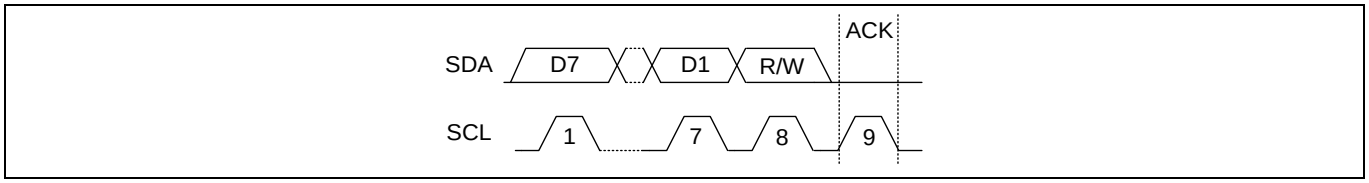

Figure 4　　7 ビット アドレス

2.5　ACK/NAK

マスターはアドレスを送信した後、スレーブからのアクノリッジ (ACK) 信号を待ちます。アクノリッジは、各データ バイトの終わりに追加されたステータス ビットです。つまり、各 I²C トランザクションは 9 ビット長です。

バス上の各スレーブは、受信アドレスを読み出し、それを自身の内部アドレスと比較する責任があります。アドレスが一致した場合、スレーブは 9 番目のクロック サイクルで ACK を送信する必要があります。アドレスが一致しない場合、スレーブは、アクノリッジしません (NAK)。

- ACK = 0 (LOW 論理レベル)
- NAK = 1 (HIGH 論理レベル)

2.6　10 ビット アドレス

I²C 仕様は 10 ビット アドレス指定も許可しています。**Figure 5** をご参照ください。これを行うため、マスターは、2 つのアドレス バイトをスレーブに送信しなくてはなりません。最初のバイトには、シーケンス 11110、その後にアドレスの最上位 2 ビット、その後に R/W ビットが含まれます。スレーブはこのバイトをアクノリッジする必要があります。続いて、マスターはアドレスの残りの 8 ビットを送信します。最後に、スレーブは ACK/NAK を送信します。

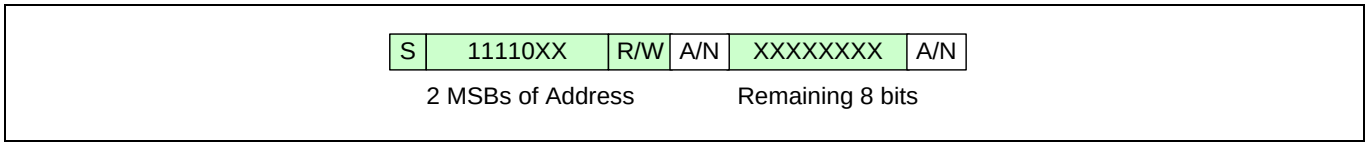

Figure 5　　10 ビット アドレス

2.7　データ

マスターがアドレスを送信し、スレーブがそれをアクノリッジした後、一度にデータを 8 ビットずつ転送できます。マスターとスレーブはともに送受信できるため、両方とも SDA ラインを制御します。

SDA ラインにおけるデータは、SCL の立ち上がりエッジの前に安定する必要があります。SDA におけるデータは SCL が LOW の時のみに変更できます。**Figure 6** を参照してください。

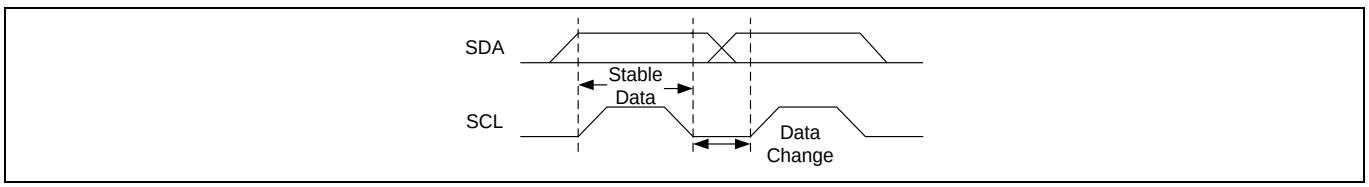

Figure 6　　SDA データの変更

マスターが書き込み操作を行う場合、SDA に 8 データ ビットを書き込み、SCL に 8 クロック サイクルを提供します。マスターが 8 データ ビットを送信した後、スレーブは 9 番目のクロック サイクルで ACK または NAK を送信しなければなりません。**Figure 7** をご参照ください。

- ACK = スレーブにはさらなるデータのための空き容量がある
- NAK = スレーブは、これ以上のデータを受け取れない

もう 1 つの状態は、マスターがスレーブからデータを読み出したい場合です。この場合、マスターが 8 クロック サイクルを提供しますが、スレーブが 8 データ ビットを SDA 上に送信します。8 データ ビットが転送された後、マスターは、ACK または NAK を送信する必要があります。

- ACK = マスターはさらなるデータを読み出したい
- NAK = マスターは読み出しを終了

Note: *送信するデータがもうなくなったことをスレーブはマスターに通知できません。*

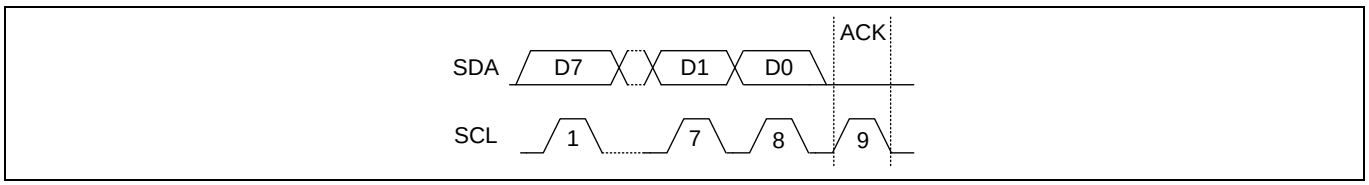

Figure 7 8 データ ビットとそれに続く ACK ビット

2.8 ストップ

全バイトが送信されたら、マスターは STOP 条件を送信しなければなりません。STOP 条件は、現時点のトランザクションが完了して、バスが開放されたことを示します。**Figure 8** を参照してください。

STOP 条件 = SCL が HIGH の時に、SDA で LOW から HIGH へ移行

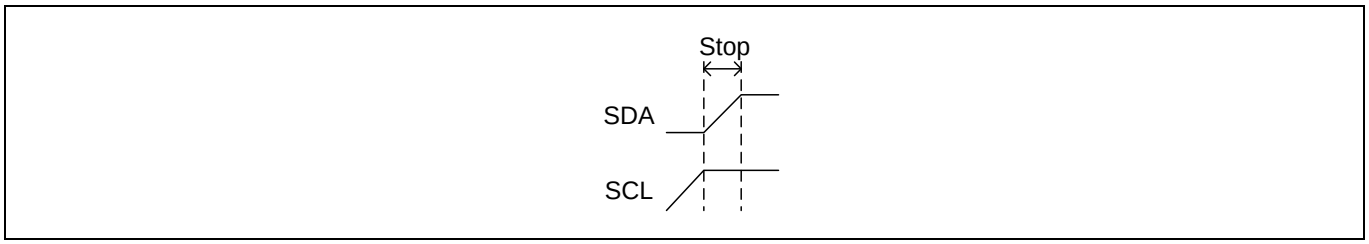

Figure 8 STOP 条件

マスターは STOP 条件の代わりに反復 START 条件を送信できます。反復 START 条件は、物理的には START 条件と同じです。マスター デバイスが新しいトランザクションを開始するためにスレーブ デバイスに通信したいか、またはデータ フローの方向を変えたい場合、マスターは反復 START 条件によってバスを制御し続けられます。

反復 START 条件は、マスターが特定のスレーブにデータを書き込み、そして次にそのスレーブからのデータを読み出したい場合に必要です。反復 START を使用すると、マスターはバスを制御し続けられます。マスターが STOP 条件を使用すると、バス上の他のマスターが制御権を得られます。

2.9 まとめ

Figure 9 に、マスターによる 2 バイトの読み書きの完全なトランザクションを示します。最初は 2 バイトを書き込んでから 2 バイトを読み出すという反復 START も示します。

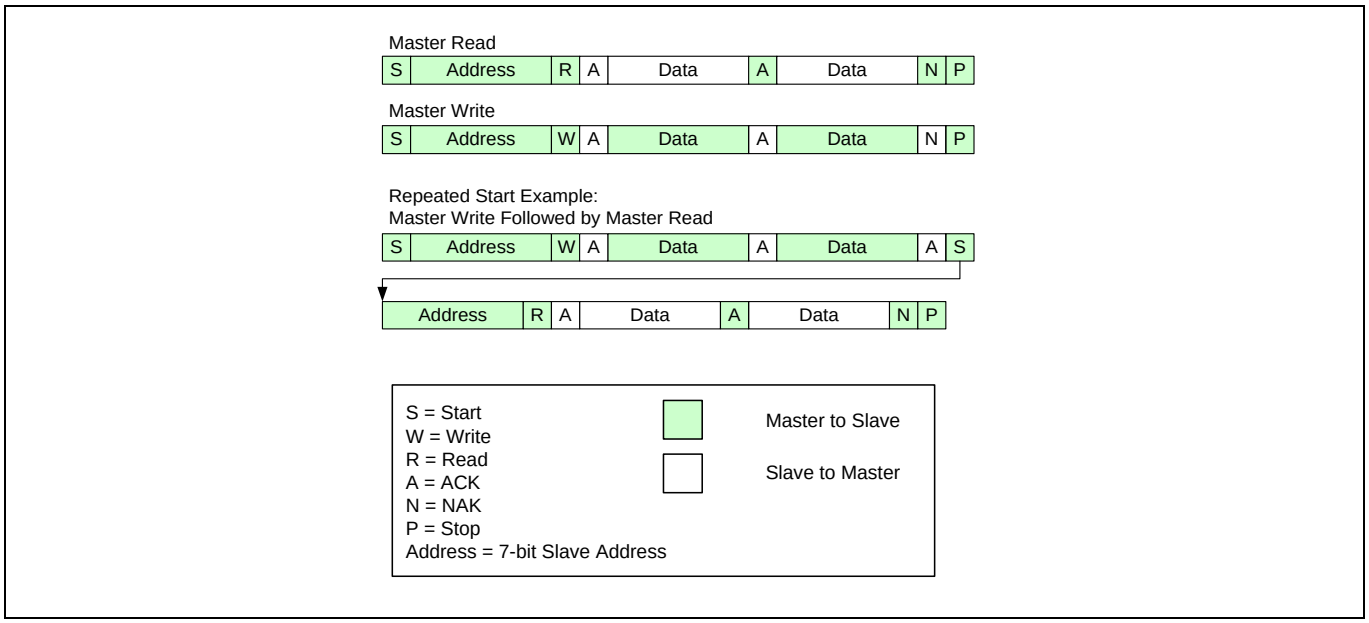

Figure 9 完全なトランザクション

2.10 アービトレーション

I²C プロトコルは、同一のバス上で複数のマスターが通信することを許可します。2 つのマスターが、同時に通信を開始する場合があります。データの損失を防ぐため、各 I²C マスターは I²C バスをチェックして、バス上のデータがバス上にあるべき正しいデータであることを確認する必要があります。データが一致しない場合、アービトレーションに負けた I²C マスターは、SDA ラインの駆動を停止し、バスが解放され、データを再度送信できるようになるまで待たなければなりません。

Figure 10 に示すように、1 つのマスターが LOW に駆動しようとしている間に、SDA を HIGH にしようとしたマスターは、アービトレーションに負け、待つ必要があります。

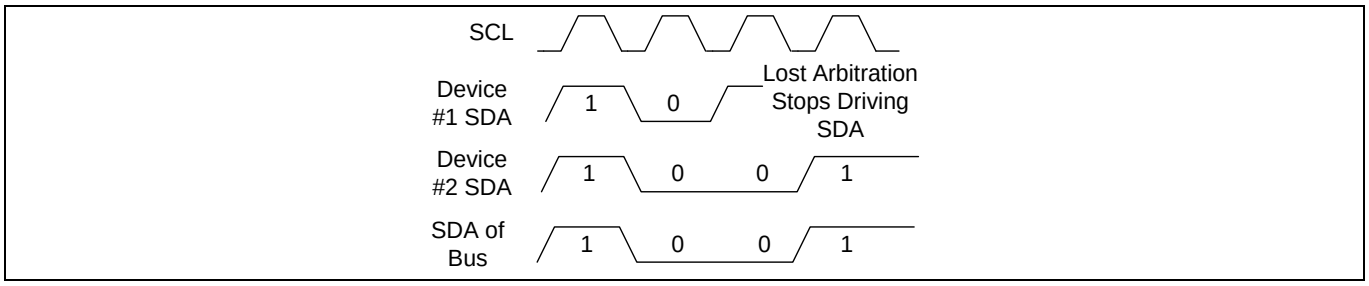

Figure 10 アービトレーション

I²C とそのプロトコルに関する詳細情報については、Philips (NXP) の **I2C の仕様**を参照してください。

3 PsoC™ 1 の I²C

PsoC™ 1 では、専用 I²C ハードウェア ブロックが I²C トランザクションを処理します。これにより、CPU の大量の処理負荷を削減し、重要なリアル タイム タスク用に CPU を開放できます。このブロックの基本構造はほとんどの PSoC™ 1 デバイスで同じです。製品ファミリ間における違いについては、**Table 1** を参照してください。

Table 1 I²C ハードウェア ブロックの相違点

デバイス	マスター	スレーブ	ハードウェア アドレスの一致	2 個の I²C ブロック
20x34	無	有	無	無
20xx6A	無	有	有	無
21x23	有	有	無	無
21x34	有	有	無	無
22xxx/21x45	有	有	無	無
23x33	有	有	無	無
24x23A	有	有	無	無
24x94	有	有	無	無
27x43	有	有	無	無
28xxx	有	有	有	有
29x66	有	有	無	無

3.1 ハードウェア

Figure 11 に示すように、ハードウェア ブロックでは、ポート 1.5、1.0、1.7、および 1.1 を I²C バスに接続できます。ポート 1.1 と 1.0 はプログラミングに使用されるため、I²C にそれらを使用しないことを推奨します。

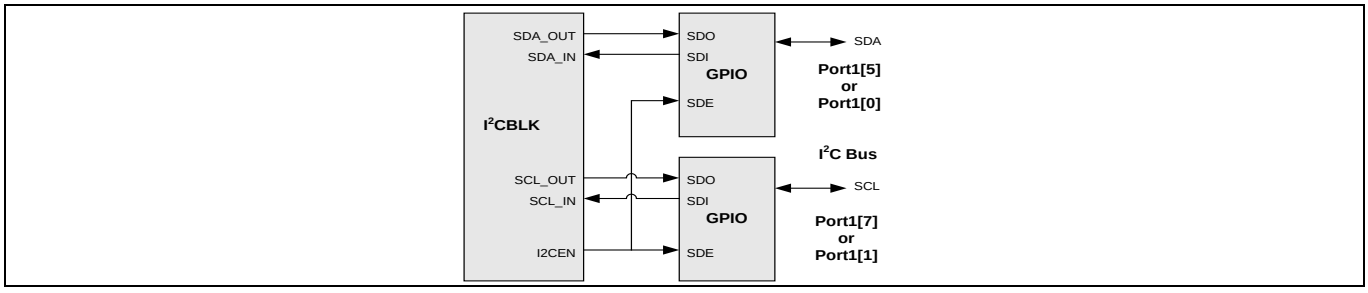

Figure 11 I²C ハードウェア ブロック

ハードウェア ブロックは、I²C トランザクションのステータスおよびタイミング要件すべてに対処する簡単なブロックです。このブロックはマスター モードの時に I²C クロックを生成します。また、このブロックは、I²C データを PSoC™ 1 へシフトイン/シフトアウトし、I²C トランザクションとエラーの状態を報告します。

3.2 割込みとトランザクションのキュー

ブロックが一度に送受信できるのは 1 バイトだけです。バイトごとに、ブロックは割込みを生成します。CPU は、割込みサービスを開始し、ブロックにさらなるデータを提供するか、ブロックが受信したデー

タを読み出す必要があります。ブロックは CPU が SCL 信号を解放するまで SCL を LOW に保持するため (このプロセスはクロック ストレッチと呼ばれる)、CPU は直ちにサービスを開始する必要はありません。詳細については、**クロック ストレッチおよび割込みレイテンシ**を参照してください。

ブロックは一度に 1 つのトランザクションのみをキューに登録できます。複数の START 条件をブロックのキューに登録することはできません。そのため、ユーザー コードは、次のトランザクションが開始される前に、現時点の I²C トランザクションが完了したことを確認しなければなりません。

ブロックは、マルチ マスター環境で自動的にアービトレーション状態を検出し報告します。アービトレーション イベントが生じた場合、ブロックは CPU にアービトレーションに負けたことを報告します。ユーザー コードはアービトレーションに負けたかどうかを確認します。そうである場合、コードは転送を再試行します。

I²C ハードウェア ブロックはバス エラー、ストップ、バイト完了の 3 条件で割り込みを生成します。

バス エラーはバスに誤った START または STOP がある場合に発生します。エラーが起こると、バス上のすべてのデバイスは現時点の転送を停止し、アイドル状態に戻る必要があります。

有効にされていると、バス上で STOP 条件が発生する度にストップ割込みが発生します。

バイト完了割込みはデータ フローの方向によって異なる位置で引き起こされます。**Table 2** を参照してください。この表はアドレスとデータの両方の転送に適用されます。

Table 2　　バイト完了割込み

モード	マスター	スレーブ
トランスミッター	8 データ ビット + ACK/NAK の後	8 データ ビットの後
レシーバー	8 データ ビットの後	8 データ ビット + ACK/NAK の後

I²C ハードウェア ブロックの機能の詳細情報については、**付録 A** および **PSoC™テクニカル リファレンス マニュアル**の I²C 節を参照してください。

3.3　　CY8C28xxx のハードウェア ブロック

CY8C28xxx 製品ファミリは、2 個の独立した I²C ハードウェア ブロックを提供するため、一度にハードウェアを 1 つ以上の I²C バスに接続できます。さらに、各ブロックはハードウェア アドレス一致機能も提供します。ハードウェアはアドレスが一致している時のみ、CPU へ割込みを生成します。しかし、PSoC™ 1 をスリープ状態からウェイクアップさせることはありません。アドレス一致の割り込みの後、ハードウェアは、**Table 2** に記載される条件に基づいて CPU を割り込みます。

CY8C28xxx デバイス ファミリの各ハードウェア ブロックにより、ポート 1.2 と 1.6、またはポート 3.0 と 3.2 における I²C ピン接続が可能になります。

2 個の I²C ハードウェア ブロックを内蔵することにより、有用な用途が多くあります。**Table 3** に、実現できる様々な用途の一部を示します。

Table 3　　2 つの I²C 構成による一般的な用途

コンフィギュレーション	一般的な用途
2 つの I²C スレーブ	マルチ I²C バス システム I²C 共有メモリ デバイス
1 つの I²C スレーブ、 1 つの I²C マスターまたはマルチマスター	I²C バス スイッチ I²C ホットスワップ コントローラー I²C バッファ デバッギング
2 つの I²C マスターまたはマルチマスター	帯域幅の増大

4 I²C ユーザー モジュール

インフィニオンは、 PSoC™ Designer 内のユーザー モジュール カタログに、コンフィギュレーション済みでプリコードされた I²C ユーザー モジュール (UM) を提供します。**Figure 12** を参照してください。

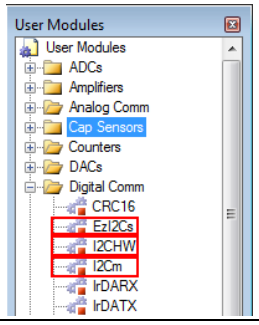

Figure 12 PSoC™ Designer 内の I²C のユーザー モジュール

これらのユーザー モジュールは、専用のハードウェア ブロックを使用する、またはソフトウェア制御のマスターであるため、デザインに配置しても、PSoC™ Designer 内の Chip View に現れません。これらユーザー モジュールは、28xxx ファミリのみにおいてデジタル ブロックを使用します。この場合、Chip View では、I²C ハードウェア ブロックが可視であるため、ユーザーは使用したブロックと残りのブロックの数が分かります。**Figure 13** を参照してください。

Figure 13 28xxx ファミリでの 2 個の I²C ブロック

EzI2Cs および I²CHW を含む I²C のユーザー モジュールは、I²C ハードウェア ブロックの抽象レベルを提供します。ここでは、各ユーザー モジュールと、設計の中でそれを適用する必要な時点を簡単に説明します。詳細については、ユーザー モジュールのデータシートを参照してください。

4.1 EzI2Cs

EzI2Cs は、スレーブとしてのみ使用できます。マスター バージョンはありません。設計でマスター動作が必要な場合、I²CHW または I²Cm を使用してください。

EzI2Cs は、名前が示すとおり、 実装が簡単な I²C スレーブ インターフェースです。それは、I²C ハードウェア ブロック上のファームウェア層です。それは、**付録 A** にあるハードウェア ブロックの説明に記述されている、ファームウェア要件の多くを実装します。

EzI2Cs は、特徴的で、ユーザーは、I²C バスの仕組みについて最小限の知識しか必要としません。それによりユーザー コードでデータ構造をセットアップし、I²C マスターに公開できます。すべての I²C トランザクションは、割込みを介してバックグランドで行われます。ユーザー モジュールが開始した後は I²C 機能を気にする必要はありません。アプリケーション コードは、マスターが読み出すデータ構造のデータ更新と、マスターが書き込むデータの確認と処理を行うだけで済みます。

このユーザー モジュールは、PSoC™がデータをマスター デバイスに連続的に流す必要があり、最小限の I²C コードのみ書きたい場合に最適です。例えば、このユーザー モジュールは、 CAPSENSE™アプリケーションでよく使われています。PsoC™は CAPSENSE™ボタンの状態を読み出し、マスター デバイスは継

I2C ユーザー モジュール

続的に PSoC™と通信して、ボタンが押されたかどうかを確認します。この場合、CAPSENSE™変数を EzI2Cs に公開することで、マスターは CAPSENSE™ボタンの状態を容易に読み出せます。

EzI2Cs は、EEPROM などの I²C ベース メモリに基づいて作られます。それは、公開された I²C データ構造内の特定データ ロケーションに書き込んだり読み出すためにサブアドレス指定機能を使用します。例えば、次のような I²C データ構造を考えます。

```
struct MyI2C_Regs {
      BYTE bStat;
      BYTE bCmd;
      int  iVolts;
      char cStr[6];
} MyI2C_Regs
```

上記データは次のメモリマップとともにマスターに公開されます。

0x00	MyI2C_Regs.bStat
0x01	MyI2C_Regs.bCmd
0x02	MyI2C_Regs.iVolts(MSB)
0x03	MyI2C_Regs.iVolts(LSB)
0x04	MyI2C_Regs.cStr[0]
0x05	MyI2C_Regs.cStr[1]
0x06	MyI2C_Regs.cStr[2]
0x07	MyI2C_Regs.cStr[3]
0x08	MyI2C_Regs.cStr[4]
0x09	MyI2C_Regs.cStr[5]

マスターが iVolts に書き込みたい場合、まずスレーブ アドレスを、その後に構造内の iVolts のオフセットを示すサブアドレス (0x02) を送信します。オフセットを計算するためには、iVolts の前のバイト数を数えます。この例では、サブアドレスは「2」になります。この例をさらにみてみると、サブアドレス「0」の場合は bStat に、サブアドレス「1」の場合は bCmd に、サブアドレス「4」の場合は cStr アレイの最初の要素に書き込みます。**Figure 14** に、マスターが、スレーブ アドレスが 0x04 の EzI2Cs ユーザー モジュール内の iVolts へデータを書き込む例を示します。

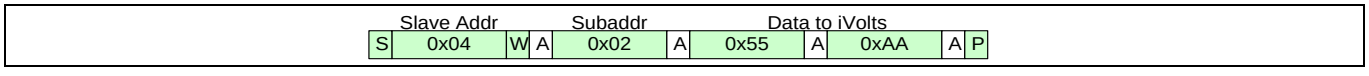

Figure 14　　EzI2Cs の iVolts への書き込み例

マスターが iVolts を読み出したい場合、最初に PSoC™スレーブのアドレスを指定してからサブアドレス「2」を書き込む必要があります。それから再びスレーブのアドレスを指定し、2 バイトを読み出す必要があります。新しいサブアドレスが書き込まれるまで、後続の各読み出しは iVolts から開始します。**Figure 15** に、マスターが iVolts のデータを読み出す例を示します。

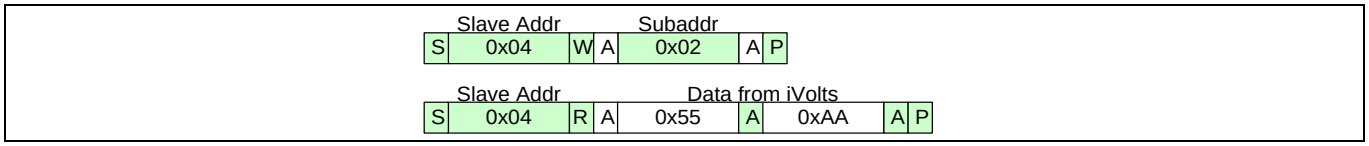

Figure 15 EzI2Cs の iVolts からの読み出し例

EzI2Cs がどのようにプロジェクトに実装されるのか見てみましょう。**Figure 16** に、プロジェクトのパラメーターを示します。

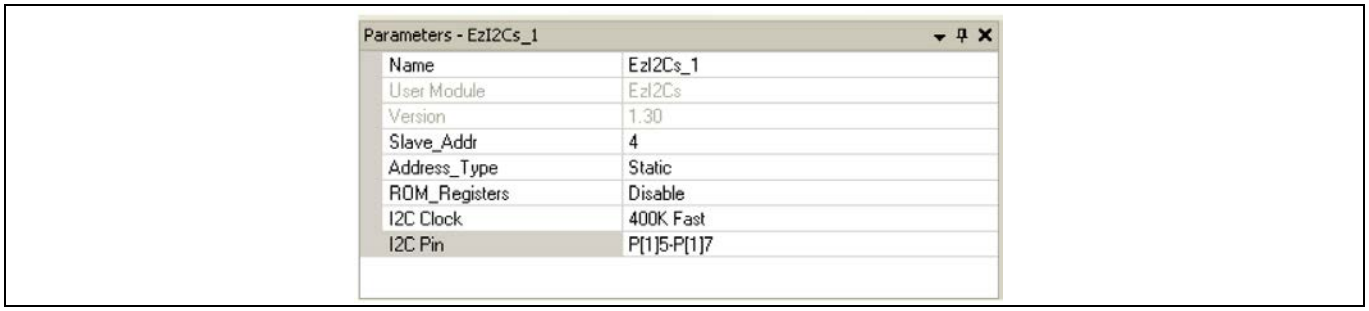

Figure 16 EzI2Cs のユーザー モジュールのパラメーター

Slave_Addr: このパラメーターは EzI2Cs スレーブのアドレスを設定します。ROM_Registers パラメーターが無効の場合、Slave_Addr は 0～127 の範囲内にある 7 ビット アドレスです。ROM_Registers パラメーターが有効の場合、Slave_Addr は 0～63 の範囲内にある 6 ビット アドレスです。

Address_Type: アドレス タイプがスタティックに設定されている場合、EzI2Cs スレーブのアドレスは Slave_Addr パラメーターに設定されている値に固定されます。アドレス タイプがダイナミックに設定される場合、EzI2Cs_SetAddr 関数を使用して、ファームウェアでアドレスを変更できます。これは複数の PSoC™ 1 EzI2Cs スレーブが 1 つのバスに共存するアプリケーションで役に立ちます。また、スレーブのアドレスは GPIO ピンをコンフィギュレーションすることで設定できます。

ROM_Registers: ROM_Registers パラメーターを有効化することで、ROM 中のアレイに保存されるデータを I²C マスターに公開できます。このパラメーターが有効化されると、EzI2Cs は 2 つのアドレスを伴ってマスターに公開されます。マスターが ROM メモリ空間にアクセスする場合、7 ビット目がセットされた 7 ビット アドレスを使用します。RAM メモリ空間にアクセスする場合、7 ビット目がクリアされた 7 ビット アドレスを使用します。例えば、Slave_Addr に 0x04 が設定された場合、マスターは ROM レジスタにアクセスするためにアドレス 0x44 でスレーブのアドレスを指定し、RAM レジスタにアクセスするためにアドレス 0x04 でスレーブのアドレスを指定します。これは、ROM レジスタが有効にされた時には Slave_Addr が 6 ビットアドレスでなくてはならない理由です。

I2C_Clock: このパラメーターによりスレーブが動作可能な最大速度を設定します。最大速度は 24MHz の SYSCLK を基にしていることに注意してください。グローバル リソース内で SYSCLK を 6MHz または 12MHz (SLIMO 有効) に設定した場合、最大クロック速度は同じ比率で低減されます。例えば、I2C_Clock パラメーターに 400kHz、 SYSCLK に 6MHz を設定した場合、スレーブの実際の速度は 100kHz になります。詳しくは、**クロック速度**を参照してください。

I2C_Pin: このパラメーターは I²C クロックおよびデータ用に使用されるピンを選択します。

ユーザー モジュール パラメーターをコンフィギュレーションした後、ファームウェア内で次の手順に従って EzI2Cs を動作させます。

1. マスターに公開するデータ構造を作成してください。例えば以下です。

I2C ユーザー モジュール

```
structMyI2C_Regs {
    BYTE bStat;
    BYTE bCmd;
int  iVolts;
char cStr[6];
} MyI2C_Regs
```

2. EzI2Cs_Start 関数を呼び出して EzI2Cs を起動してください。
3. EzI2Cs_EnableInt 関数を呼び出して割り込みを有効にしてください。
4. EzI2Cs_RamSetBuffer 関数を使用してデータ構造をマスターに公開してください。

```
EzI2Cs_SetRamBuffer(sizeof(MyI2C_Regs), 2, (char*)&MyI2C_Regs);
```

最初のパラメーターでバッファのサイズを設定します。2 番目のパラメーターは、書き込み境界を設定し、3 番目のパラメーターはデータ構造へのポインタを初期化します。

EzI2Cs は、SetRamBuffer 関数を通してデータ構造の読み出し／書き込み許可を定義することを可能にします。例えば、ステップ 1 で示したコードでは、アプリケーションは書き込み境界を「2」に設定することで bStat と bCmd 変数に読み出し／書き込み許可をし、他のすべての変数を読み出し専用にします。この場合、マスターが読み出し専用変数に書き込もうとした場合、スレーブが NAK 信号を生成してマスターによって書き込まれたデータを無視します。

5. メイン ループ内では、マスターが読み出すためのプロセス データでデータ構造を更新し続け、またマスターからの新しいデータを待ちつづけます。

データの一貫性は、EzI2Cs のユーザー モジュールを使用する際の重要な考慮事項です。この例において、iVolts は 2 バイトの変数です。CPU がこの変数の更新を開始したが、I²C マスターが両バイトを読み出す前に、1 バイトしか書き込む時間がなかったということがありえます。その結果として、マスターは誤データを読み出しました。例えば、iVlots 変数が 0x01FF 値を持つことを想定します。CPU は、この変数を 0x0200 の新しい値に更新しなければなりませんでした。CPU は、まず最上位ビット (MSB) に 0x02 を、最下位ビット (LSB) に 0x00 を書き込みます。CPU が MSB に書き込んだ直後、LSB への書き込みの前に I²C の読み出しが行われた場合、マスターは 0x2FF を読み出すことになります。

これに対抗するためには、データの読み出しや書き込みの準備ができたことを示すために、マスターとスレーブの間でフラグやセマフォを使用するのが最善です。**EzI2Cs データシート**には、データの一貫性を保つための他の方法が詳しく記載されます。

本アプリケーション ノートには、EzI2Cs のユーザー モジュールの使用方法をデモする簡単なプロジェクトが添付されています。プロジェクトの使用方法の詳細については、**付録 B** を参照してください。

このユーザー モジュールの仕組みやその使用法の詳細については、**EzI2Cs データシート**を参照してください。

EzI2Cs のまとめと重要注意事項
EzI2Cs は、実装しやすいスレーブ専用ユーザー モジュールです。
EzI2Cs は、読み出し／書き込み許可が明確に定義されたレジスタ構造をマスターに公開します。
ROM レジスタが有効な場合、スレーブは、RAM 空間用 (7 ビット目がクリアされる) と ROM 空間用 (7 ビット目がセットされる) の 2 つのアドレスを持ちます。このため、ROM レジスタが有効な場合、スレーブ アドレスは 6 ビット値に制限されます。
EzI2Cs 内の複数バイト値を処理する場合、マスターとスレーブ間でフラグまたはセマフォを使用してデータの一貫性を保ってください。

4.2 I²CHW

このユーザー モジュールは、I²C ハードウェア ブロック上にあるファームウェア層であり、付録 A に述べているファームウェア タスクをすべて実装します。この柔軟なユーザー モジュールはスレーブ、マスター、またはマルチマスターのスレーブとして使用できます。

EzI2Cs とは異なり、I²CHW はより多くのコード操作を必要とします。ステータス ビットをチェックして、I²C トランザクションが発生したかどうかを確認します。トランザクションが完了した時、バッファを再初期化し、ステータス ビットをクリアします。また、トランザクションでエラー状態が発生したかどうかファームウェアをチェックします。

以下は、プロジェクトで I²CHW を実装する方法です。ダブルクリックすると、I²CHW トポロジ選択ウィンドウが開きます。Figure 17 を参照してください。

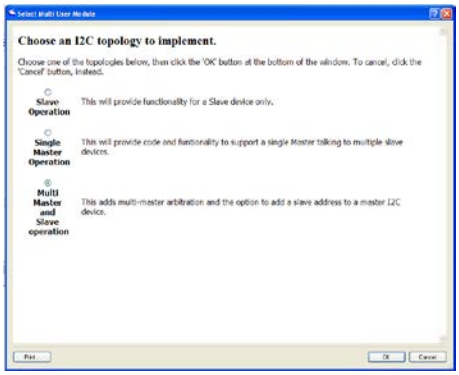

Figure 17 I²CHW トポロジ ウィンドウ

スレーブ動作: スレーブ動作が必要な場合、このオプションを選択してください。

シングル マスター動作: シングル マスター環境でシンプルな動作が必要な場合、このオプションを選択してください。

マスター動作では、Read_Buffer_Type、I2C_Clock、I2C_Pin という 3 つのパラメーターが使用できます。これらのパラメーターのコンフィギュレーションは、以下のスレーブ動作の場合のパラメーターのコンフィギュレーションと同様です。

マルチマスターおよびスレーブ動作: マルチマスター環境で I²CHW をマスターおよびスレーブとして必要な場合、このオプションを選択してください。

選択したオプションに応じて、適切なユーザー モジュール パラメーター ウィンドウが表示されます。

Slave_Addr: これは 7 ビット スレーブ アドレスです。EzI2Cs と異なり、I²CHW は動的アドレス指定または ROM バッファ用の別のアドレスを提供しません。マルチマスターおよびスレーブ動作において、Slave_Addr はバス上の別のマスターによってスレーブとしてアドレス指定された時のデバイスのアドレスとなります。

Read_Buffer_Type:「RAM only」 (RAM のみ) を選択した場合、アプリケーションは、RAM でのみ読み出しバッファをセットアップします。「RAM or flash」 (RAM またはフラッシュ) を選択した場合、アプリケーションはマスターに RAM バッファかフラッシュ バッファを公開します。

通信サービス タイプ:「Interrupt」 (割り込み) を選択した場合、すべての I²C トランザクションが自動的に割り込み内で処理されます。「Polled」 (ポーリング) を選択した場合、I²C トランザクションは、フォアグラウンド アプリケーションが I2CHW_Poll 関数を呼び出した時のみ処理されます。ハードウェア

は、アプリケーションが I2CHW_Poll を呼び出すまで I²C クロックストレッチを継続します。サイプレスは「Interrupt」 (割込み) オプションを推奨します。

I2C_Clock: このパラメーターはスレーブのクロック速度を選択します。EzI2Cs スレーブのクロック パラメーターで説明したように、スレーブのクロック速度は 24MHz の SYSCLK を基準にします。

I2C_Pin: このパラメーターは、I²C 用のピンを選択します。CY8C28xxx を除いてすべてのデバイスでは P1[0]-P1[1]および P1[5]-P1[7] から選択します。CY8C28xxx の場合、P1[2]-P1[6]および P3[0]-P3[2]も選択可能です。

5 ファームウェア

トポロジとパラメーターをコンフィギュレーションした後、次の手順に従って I²CHW をファームウェアで動作させてください。

5.1 スレーブ動作

1. マスターが読み出せるように RAM (またはフラッシュ) 内に読み出しバッファを宣言します。例:

   ```
   BYTE ReadBuffer[16];
   ```

   バッファは次のようにも構築できます。

   ```
   struct ReadBuffer {
       BYTE bStatus;
       int iVolts;
   }ReadBuffer;
   ```

2. マスターが書き込めるように RAM 内に書き込みバッファを宣言します。例:

   ```
   BYTE WriteBuffer[16];
   ```

   バッファは次のようにも構築できます。

   ```
   struct WriteBuffer {
       BYTE bCmd;
       int iDACCounts;
   }WriteBuffer;
   ```

3. M8C_EnableGInt マクロを呼び出してグローバル割り込みを有効にします。

   ```
   M8C_EnableGInt ;
   ```

   すべての I²C が I²C ISR 内のバックグラウンドで動作するため、グローバル割り込みを有効にするのは重要です。

4. I2CHW_Start、I2CHW_EnableSlave、および I2CHW_EnableInt 関数を呼び出してスレーブを開始します。

   ```
   I2CHW_Start();
   I2CHW_EnableSlave();
   I2CHW_EnableInt();
   ```

5. I2CHW_InitRamRead 関数を使用して読み出しバッファを初期化します。

   ```
   I2CHW_InitRamRead(ReadBuffer, sizeof(ReadBuffer));
   ```

   この関数は、読み出しバッファへのポインタを初期化し、バッファサイズを設定します。マスターがスレーブからデータを読み出そうとする時は常に、データがバッファからマスターに自動的に送信されます。マスターがバッファ サイズを超えるバイト数を読み出そうとすると、バッファの最終バイトが繰り返し送信されます。

6. I2CHW_InitWrite 関数を使用して書き込みバッファを初期化します。

   ```
   I2CHW_InitWrite(WriteBuffer, sizeof(WriteBuffer));
   ```

   この関数は書き込みバッファへのポインタおよび、バッファ サイズを初期化します。マスターがスレーブにデータを書き込む時、データはバッファに自動的に蓄積されます。マスターがバッファ サ

イズを超えるバイト数を書き込もうとすると、スレーブから余分のバイトに対するアクノリッジ信号が発行されず、それら余分のバイトは破棄されます。

7. I2CHW_bReadI2CStatus 関数を呼び出して I2CHW_RD_COMPLETE フラグを確認します。フラグがセットされている場合、マスターの読み出しトランザクションが完了したことを意味します。ステータス フラグをクリアし、読み出しバッファを再初期化してください。

```
// Check if a read operation is over
if (I2CHW_bReadI2CStatus()
I2CHW_RD_COMPLETE)
{
// Prepare fresh data for Master


// Clear the flag
I2CHW_ClrRdStatus();


// Re-initialize the buffer
I2CHW_InitRamRead(ReadBuffer,
sizeof(ReadBuffer));
}
```

RD_COMPLETE フラグをチェックし、バッファを再初期化することは重要です。そうしなければ、さらにマスターからの読み出しを続けた時に、バッファの最終バイトが繰り返し送信されるようになります。

8. I2CHW_bReadI2CStatus 関数を呼び出し I2CHW_WR_COMPLETE フラグをチェックします。フラグがセットされることは、マスターの書き込みトランザクションが完了したことを意味します。データを処理し、フラグをクリアし、書き込みバッファを再初期化します。

```
// Check if a write operation is over
if (I2CHW_bReadI2CStatus() & I2CHW_WR_COMPLETE)
{
// Process data from Master
// Clear the flag
I2CHW_ClrWrStatus();


// Re-initialize the buffer
I2CHW_InitWrite(WriteBuffer,
sizeof(WriteBuffer));
}
```

書き込みバッファの再初期化は重要です。そうしなければ、マスターからの以降のデータのすべてに対してスレーブはアクノリッジ信号を返しません。

本アプリケーション ノート内のシンプルなプロジェクトは、I²CHW を I²C マスターにより書き込まれるデータをエコーするスレーブとして使用する方法をデモします。プロジェクトの詳細については、**付録 B** をご参照ください。

5.2 マスター動作

1. スレーブから読み出されるデータが保存される RAM (またはフラッシュ) 内に読み出しバッファを宣言してください。例:

   ```
   BYTE ReadBuffer[16];
   ```

   このバッファも上記の「スレーブ動作」節に説明したものと類似した構造にすることもできます。

2. I²C スレーブに書き込まれるデータ用に RAM 内に書き込みバッファを宣言してください。例:

   ```
   BYTE WriteBuffer[16];
   ```

3. M8C_EnableGInt マクロを呼び出してグローバル割り込みを有効にしてください。

   ```
   M8C_EnableGInt;
   ```

   すべての I²C が I²C ISR 内のバックグラウンドで動作するため、グローバル割り込みを有効にするのは重要です。

4. I2CHW_Start、I2CHW_EnableMstr、および I2CHW_ EnableInt 関数を呼び出してマスターを開始してください。

   ```
   I2CHW_Start();
   I2CHW_EnableMstr();
   I2CHW_EnableInt();
   ```

5. スレーブに書き込むためには、I2CHW_bWriteBytes 関数を使用してください。

   ```
   I2CHW_bWriteBytes(0x50, WriteBuffer, 16, I2CHW_CompleteXfer);
   ```

   第 1 のパラメーターはスレーブ アドレスです。7 ビット スレーブ アドレスを使用してください。bWriteBytes 関数は、I²C バス上にアドレスを送信する前に、読み出し／書き込みビットを 7 ビット アドレスに自動的に追加します。第 2 のパラメーターは、スレーブ用のデータを保持しているバッファへのポインタです。第 3 のパラメーターはスレーブに書き込まれるバイト数です。第 4 のパラメーターは、トランザクション タイプで、I2CHW_CompleteXfer、I2CHW_NoStop、および I2CHW_RepStart という 3 つの異なる値があります。

   I2CHW_CompleteXfer を使用すると、スタート ビット、アドレス バイト、データ バイト、およびストップ ビットを含む、1 つの I²C のトランザクションが完全に行われます。I2CHW_NoStop を使用すると、データの書き込み後 STOP が生成されません。NoStop によるトランザクションの後には、I2CHW_RepStart によるトランザクションを続けられます。一般に、I2CHW_CompleteXfer を大部分の I²C トランザクションに適用することを推奨します。

   I2CHW_bWriteByte 関数は、バッファへのポインタを初期化し、カウント値を設定し、I²C ハードウェアのスタート ビットを初期化します。その後、すべての動作が ISR 内部で行われます。I²C ハードウェアは、各バイト完了時に割込みを生成し、ISR は、バッファへのポインタをインクリメントし、次のバイトをスレーブに送信します。すべてのバイトが送信されると ISR は STOP を生成します。

6. I2CHW_bReadI2CStatus 関数を使用して、書き込み処理が完了したかどうかを確認し、その後フラグをクリアしてください。例:

   ```
   while(!(I2CHW_bReadI2CStatus() &
       I2CHW_WR_COMPLETE));
       I2CHW_ClrWrStatus();
   ```

   上記のコードは WR_COMPLETE フラグがセットされるまで待機します。「while」の代わりに「if」条件文を使用できます。I²C トランザクションがバックグラウンドで動作している間に、プロセッサは別の処理を行えます。

7. I2CHW_fReadBytes 関数を使用してスレーブからの読み出しを開始してください。I2CHW_bReadI2CStatus 関数を呼び出して、I2CHW_RD_COMPLETE フラグをチェックしてから、読み出し状態をクリアしてください。

```
I2CHW_fReadBytes(0x50, ReadBuffer, 16,     I2CHW_CompleteXfer);
while(!(I2CHW_bReadI2CStatus() &
    I2CHW_RD_COMPLETE));
    I2CHW_ClrRdStatus();
```

5.3 マルチマスター スレーブ動作

マルチマスター スレーブ モードでは、I²CHW は、マルチマスター環境ではマスターとスレーブの両方として機能できます。

1. I2CHW_Start 関数を呼び出してください。
2. I2CHW_EnableMstr 関数と I2CHW_EnableSlave 関数を呼び出してマスター モードとスレーブ モードの両方を有効にし、I2CHW_EnableInt 関数を呼び出して割込みを有効にしてください。

```
I2CHW_Start();
I2CHW_EnableMstr();
I2CHW_EnableSlave();
I2CHW_EnableInt();
```

3. スレーブ モード動作は単一スレーブ動作と類似していますが、関数名は異なります。

- 読み出しバッファと書き込みバッファを割り当てます；I2CHW_InitSlaveRamRead および I2CHW_InitSlaveWrite 関数を呼び出してこれらのバッファを初期化します。
- I2CHW_bReadSlaveStatus を呼び出します。
- I2CHW_RD_COMPLETE および I2CHW_WR_COMPLETE フラグを確認します。フラグがセットされる場合は、バッファを再初期化します。

4. 動作のマスター側用ファームウェアは特殊なケースに対処しなくてはなりません。マルチマスター環境では、マスター (仮にマスター1 と呼びます) がスレーブへのトランザクションを開始すると、他のマスターの動作に応じて 3 つの異なるシナリオが存在します。(バスに別のマスターがあると仮定し、それをマスター2 と呼びます。)
   a) バスはフリーで、マスター1 が読み出しまたは書き込みトランザクションを開始し完了する場合。
   b) マスター2 がバス上の別のスレーブで既にトランザクションを開始しており、そのためバスがビジー状態である場合。この場合、マスター1 はバスがフリーになるまで待ってからトランザクションを開始しなくてはならない。
   c) マスター1 とマスター2 が同時に読み出しまたは書き込みトランザクションを開始する場合。この場合、アービトレーションが実行される。マスター1 がアービトレーションを取得すると、トランザクションを完了する。マスター1 がアービトレーションに負けると、マスター2 がトランザクションを完了した後にトランザクションを再試行しなくてはならない。アービトレーションの詳細については付録 A を参照。

   I²CHW は、マスターがマルチマスター環境で動作するための別の関数を備えています。

5. スレーブに書き込むには、I2CHW_bWriteBytesNoStall を呼び出してください。スレーブから読み出すには、I2CHW_fReadBytesNoStall を呼び出します。これらの関数は、トランザクションを開始する前に、バスがビジー状態かどうかを確認します。バスがビジー状態の場合、これらの関数は 0xFF を返します。ファームウェアは返り値をチェックして、その値が 0xFF の場合はトランザクションを再実

行しなくてはなりません。例えば、次のコードは I2CHW_bWriteBytesNoStall が 0xFF 以外の値を返すまでループし続けます。

```
while(I2CHW_bWriteBytesNoStall(0x50, WriteBuffer, 16, I2CHW_CompleteXfer)
== 0xFF);
```

ブロックする「while」ループ文の代わりに、「if」条件文を使用して返り値を確認することもできます。返り値が 0xFF の場合、一定時間後にトランザクションを再実行できます。

6. 読み出し動作または書き込み動作がバスのビジーエラーもなく開始した場合、ファームウェアは次に動作が完了したかどうか、またはマスターがアービトレーションに負けたかどうかをチェックしなくてはなりません。これを実行するためのコードの例を以下に示します。

```
while((!(I2CHW_bReadMasterStatus() & I2CHW_WR_COMPLETE)) &&
(!(I2CHW_bReadBusStatus() & I2CHW_LOST_ARB)));
if (I2CHW_bReadMasterStatus() & I2CHW_WR_COMPLETE)
{
    // Write is completed successfully.
    // Clear flag
    I2CHW_ClrMasterWrStatus();
}
if (I2CHW_bReadBusStatus() & I2CHW_LOST_ARB)
{
   // Master lost arbitration. Retry later
}
```

先頭行は、書き込みが完了するか、LOST_ARB エラーフラグがセットされるまでループします。「while」ループが完了した場合、コードはどの条件でループが完了したかをチェックします。WR_COMPLETE フラグがセットされている場合は、フラグをクリアします。LOST_ARB フラグがセットされている場合、後でトランザクションを再実行します。

本アプリケーション ノートには、標準 EEPROM を読み出すためのI²CHW のユーザー モジュールの使用方法をデモする簡単なプロジェクトが添付されています。プロジェクトの詳細については、**付録 B** を参照してください。

詳細については、I²CHW のユーザー モジュールとテクニカル リファレンス マニュアルの I²C 節を参照してください。

I²CHW のまとめと重要注意事項

I²CHW は、シングル マスター モード、シングル スレーブ モード、またはマルチマスター スレーブ モードで動作可能です。

スレーブ モードでは、マスターが読み出しまたは書き込みを完了すると、次のトランザクション用にバッファを再初期化しなくてはなりません。

マルチマスター環境のマスター モードでは、ファームウェアが bWriteBytesNoStall 関数と fReadBytesNoStall 関数の返り値をチェックして、バスがビジー状態であるかどうかを確認し、トランザクションを再実行しなくてはなりません。

マルチマスター環境のマスター モードでは、読み出しまたは書き込みトランザクションを開始した時、ファームウェアはトランザクションが正常に完了したか、またはマスターが別のマスターに対しアービトレーションに負けたかどうかをチェックします。マスターがアービトレーションに負けた場合、トランザクションを再実行しなくてはなりません。

5.4 I²Cm

PSoC™ 1 内の別のユーザー モジュールである I²Cm は、GPIO ポート ピンのソフトウェア操作によって I²C マスターを実装します。このユーザー モジュールは、I²C ハードウェア ブロックを使用しません。I²C ソフトウェア スレーブ ユーザー モジュールはありません。

I²CHW に対するこのユーザー モジュールの優れた点は、P1.5、P1.7、または P1.0、P1.1 だけでなくすべてのピン ペアで使用できることです。しかし、1 つの欠点は、より大きな CPU オーバーヘッドを必要とし、マルチマスターの動作に対応しないことです。I²C トランザクションの間 CPU 能率を 100 パーセント使用してしまいます。2 つ目の欠点は、バス周波数が≤100kHz に限定されることです。

I²Cm は、1 つのチップに複数のマスターが必要なアプリケーションにとって、または I²C ハードウェアに接続するピンが利用できない時に最適です。マスターが 1 つだけ必要で、P1.5、P1.7 または P1.0、P1.1 が利用可能な場合、I²CHW をマスターとして使用することをお勧めします。

プロジェクト内でこのユーザー モジュールを機能させるステップを以下に示します。

1. I²Cm_Start 関数を使用してユーザー モジュールを開始してください。
2. データを RAM バッファ (または ROM バッファ) からスレーブへ書き込むには、I2Cm_bWriteBytes 関数を使用してください。この関数は、スレーブ アドレス、RAM (または ROM) 内のソース データへのポインタ、転送するバイト数を入力として受け入れます。
3. スレーブからデータを読み出すには、 I2Cm_fReadBytes 関数を使用してください。この関数は、スレーブ アドレス、読み出しデータが保存されるべき保存先としてのバッファへのポインタ、読み出すバイト数を入力として受け入れます。

このユーザー モジュールは、ユーザー モジュール自身が動作しているポートの駆動モード レジスタ (PRTxDMx) とデータ レジスタ (PRTxDR) を操作することで動作します。このため、ユーザー コードでデータ レジスタを操作する時は慎重に行ってください。そうしなければ、I²C トラフィックが悪影響を受けます。問題を避ける最善の方法は、I²Cm ピンに関連する駆動モードとデータ レジスタに書き込んだ時にシャドー レジスタを使うことです。

I²Cm ユーザー モジュールをプロジェクトに設定すると、PSoC™ Designer は自動的にデータ レジスタと 2 個の駆動モード レジスタ (PRTxDM0 と PRTxDM1) 用のシャドー レジスタを生成します。ユーザー モジュールは PRTxDM2 に影響を与えないため、このレジスタにはシャドー レジスタは生成されません。

これらのシャドー レジスタは、*psocconfig.asm* および *psocgpioint.h* ファイル中で定義されています。たとえば、I²Cm がポート 0 に配置されている場合、次の変数は *psocconfig.asm* ファイルで定義されます。

```
; write only register shadows
_Port_0_Data_SHADE:
Port_0_Data_SHADE:        BLK    1
_Port_0_DriveMode_0_SHADE:
Port_0_DriveMode_0_SHADE:  BLK      1
_Port_0_DriveMode_1_SHADE:
Port_0_DriveMode_1_SHADE:  BLK      1
```

次に示す定義は、*psocgpioint.h* ファイルに記述されています。

```
extern BYTE Port_0_Data_SHADE;
extern BYTE Port_0_DriveMode_0_SHADE;
extern BYTE Port_0_DriveMode_1_SHADE;
```

ファームウェア

アプリケーション コードは、これらのシャドー レジスタを使用して、データ レジスタと駆動モード レジスタに書き込みます。たとえば、アプリケーションが P0[0]をセットしたい場合には、次のコードを使用してください。

```
// Write to PRT0DR through shadow register
Port_0_Data_SHADE |= 0x01;
PRT0DR = Port_0_Data_SHADE;
```

PRT0DMx レジスタを使用して P0[0]をストロング モードに設定するには、次のようなコードを使用します。

```
// Write to PRT0DM0 through shadow register
Port_0_DriveMode_0_SHADE |= 0x01;
PRT0DM0 = Port_0_DriveMode_0_SHADE;

// Write to PRT0DM1 through shadow register
Port_0_DriveMode_1_SHADE &= ~0x01;
PRT0DM1 = Port_0_DriveMode_1_SHADE;

// Write to PRT0DM2 directly
PRT0DM2 &= ~0x01;
```

シャドー レジスタ使用の必要性に関する詳細は、**シャドー レジスタ データベース**を参照してください。このユーザー モジュールの詳細情報については、**I²Cm データシート**を参照してください。

I²Cm のまとめと重要注意事項

I²Cm は、I²C マスターのソフトウェア実装であり、I²C ハードウェア リソースを占有しません。
スレーブから／へ読み出すまたは書き込む時に I²Cm は CPU リソースをすべて使用します。
アプリケーション コードで I²Cm が配置されているポートのデータ レジスタまたは駆動モード レジスタに書き込まなくてはならない場合、シャドー レジスタを使用して I²C インターフェースの問題を回避するよう注意してください。

6 特別な I²C 注意事項

I²C はデバイス同士が通信する簡単な方法を提供します。他のすべてのデザインの場合と同じように、設計中には問題が発生することもあります。ここでは、一般的な質問に回答し、I²C で設計する際に問題の発生を防止するためにヘルプとしての情報を提供します。以下のトピックが含まれます。

- 7 ビットと 10 ビットのアドレス指定
- プルアップ抵抗の注意事項
- I²C と ISSP ピンの共有
- 電源投入中のグリッチ
- SYSCLK 対 I²C クロック速度
- クロック ストレッチと割込みレイテンシ
- ホット スワッピング
- グリッチ フィルタ処理
- I²C およびスリープ モード
- I²C およびダイナミック リコンフィギュレーション
- I²CHW でのダイナミック アドレス指定

6.1 I²C アドレス指定

I²C スレーブ アドレスを記述するためによく使われる方法が 2 つあります。1 つは読み出し／書き込みビットを使用せず、7 ビット アドレスを使用する方法です (例えば、0x42 アドレス)。これは、インフィニオンのユーザー モジュールが I²C スレーブ アドレスを扱う方法です。

もう 1 つは、アドレスの一部として読み出し／書き込みビットを含める方法です (例えば、0x84/0x85) 。

作業中のデバイスがどのアドレス指定方法を使用するかを必ず確認してください。

現時点でインフィニオンは、10 ビットのスレーブ アドレスをサポートしません。

6.2 プルアップ抵抗

I²C における設計上の別の一般的考慮事項は、プルアップ抵抗の値です。選択される抵抗値は、通信周波数とバス静電容量に依存します。バス静電容量やプルアップ抵抗器サイズが大きくなると、クロックとデータ ラインの立ち上り時間がより長くなります。I²C 仕様には最大の立ち上り時間が指定されます。

バス上の立ち上り時間が最大値を超えると、I²C 通信は正常に行われません。I²C 仕様は、プルアップ抵抗の値を決定するために、**Figure 18** に示されるグラフを提供します。R_S は直列抵抗で、R_S の最大値は I²C 仕様で定義します。

ほとんどのシステムでは、2.2k～4k のプルアップ抵抗が使われます。

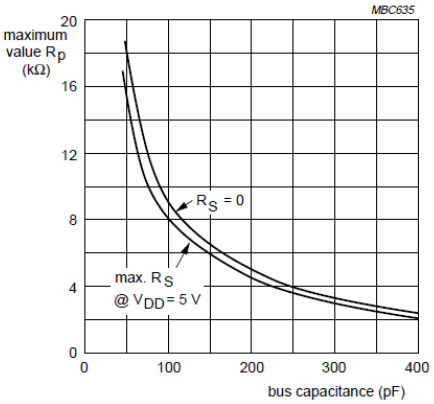

Figure 18　　プルアップ抵抗値

6.3　　I²C および ISSP プログラミングの競合

I²C を PSoC™で使用する際の 1 つの一般的な考慮事項は、使用対象とするピンの選択です。ハードウェア ブロックを使用する場合、I²C で使用可能な固定ピンは 2 ペアあります: P1.5、P1.7 および P1.0、P1.1 です。 (28xxx ファミリでは、P1.2 と P1.6、または P3.0 と P3.2 を使用することも可能) 。PSoC™をプログラムするために、P1.0、P1.1 ピンを使用できます。これらのラインのプルアップ抵抗は、インシステムプログラミングに障害を引き起こす可能性があります。

プログラミング中、P1.0 は抵抗プルダウン駆動モードを使用して、信号線の論理レベルを強制的に LOW にします。プルダウン抵抗はおよそ 5.6kΩです。この内部プルダウン抵抗と外部 I²C プルアップ抵抗が分圧器を作成します。分圧器はライン上で不定または HIGH 論理レベルを適用し、プログラマは所望の LOW 論理レベルとして認識しません。これはプログラミングが失敗する原因となります。適切な電圧レベルを決定するには、デバイス固有のデータシート内の DC プログラミング仕様と GPIO 論理レベルを参照してください。

最善なソリューションは、P1.5、P1.7 を使用することです。P1.0 と P1.1 は必要な場合だけ使用してください。もう 1 つのオプションは、外部プルアップ抵抗を使用する代わりに、PSoC™内の P1.0、P1.1 の駆動モードを変更して P1.0、P1.1 内部にプルアップすることです。内部プルアップ抵抗はおよそ 5.6kΩです。

6.4　　電源投入時のピンのグリッチ

前述したとおりに、P1.0 と P1.1 はプログラミングに使用されます。すなわち、電源投入時にはこれらピンが他のピンと違って動作します。

デバイスがリセットから開放された後、P1.0 はストロング HIGH に駆動されます。このことは、バス上の他の I²C デバイスのいずれかがラインを LOW に駆動する場合、問題を発生させます。P1.1 はレジスティブ LOW に駆動されます。前述のように、これは分圧器を形成しバスの中間電圧を引き起こし、他のデバイスはこれを認識しないかもしれません。設定された時間が経過した後、P1.0 はレジスティブ LOW に移行し、そのラインに中間電圧になってしまいます。

PSoC™が起動中に、他の I²C デバイスに電源が供給され、P1.0 と P1.1 に接続される場合、上記の動作に注意し、それがバス上の I²C デバイスにどう影響するか確認する必要があります。

この問題を回避するには、I²C 通信に P1.5 と P1.7 を使用します。もしくはリセット中の P1.0 と P1.1 のピン動作の影響を最小限にする手順に従います。簡単な方法としては、PSoC™が起動されるまで他のすべてのデバイスをリセットに維持します。または、PSoC™の出力をトランジスタや論理ゲートでゲーティングするといった、複雑なソリューションもあります。**Figure 19** をご参照ください。

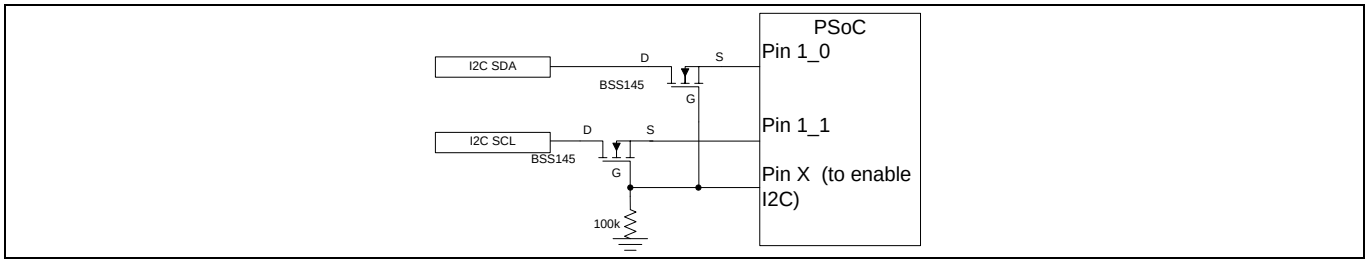

Figure 19 PSoC™の分離

Note:

- このブロックで使用される N チャネル FET を、FET のソースが PSoC™ GPIO ピンに接続されるように配置することが重要です。
- FET は N チャネルのもので、その VGS_{TH} が PSoC™の V_{dd} 以下でなければなりません。例えば、PSoC™ V_{dd} = 3.3V の場合、VGS_{TH} が 3.3V 以下の N チャネル FET を選択します。
- ピン X をストロングにセットし、それに 1 を書き込んで I^2C バスに接続します。

6.5 クロック速度

前述のように、マスター モードの時は、I^2C ハードウェアは SCL にクロックを生成します。PSoC™ 1 で使用可能な I^2C クロック周波数は、50kHz、100kHz、400kHz です。これらの周波数は、システム クロック (SYSCLK) の固定接続されたクロック分周器に基づきます。

これらのクロック分周器は I^2C ラインを 16 倍または 32 倍オーバーサンプリングするサンプリング クロックを生成します。**Table 4** に内部サンプル レートを一覧表示します。

Table 4 内部サンプル レート

クロック レート	SYSCLK プリスケーラ	内部サンプリング クロック周波数 (SYSCLK=24MHz)	ビットごとのサンプル数
50kHz	/16	1.5MHz	32
100kHz	/4	1.5MHz	16
400kHz	/16	6MHz	16

内部サンプリング クロックは SYSCLK が 24MHz であることを前提とします。SYSCLK が 24MHz より遅い場合、I^2C クロックは遅くなります。例えば、SYSCLK が 12MHz なら、使用可能速度は 25kHz、50kHz、200kHz です。SYSCLK が 6MHz なら、使用可能な I^2C 速度は 12.5kHz、25kHz、100kHz です。(**Table 5** を参照してください)。

Table 5 実際の周波数対 IMO と UM 設定

UM 設定	IMO (SYSCLK) 設定		
	24MHz	12MHz	6MHz
400kHz	400kHz	200kHz	100kHz
100kHz	100kHz	50kHz	25kHz
50kHz	50kHz	25kHz	12.5kHz

Note: SYSCLK は CPU_Clock からは独立しています。

特別な I2C 注意事項

スレーブ モードで動作している時は、I²C ブロックが読み出せる最高クロック速度に対して同じルールが適用されます。オーバーサンプリング クロックが I²C ラインを監視するために使用されます。スレーブ モードで速度を設定することは、クロックとデータ ラインがハードウェアによりオーバーサンプリングされる頻度を示します。

I²C クロック周波数が 400kHz で、かつハードウェアが 100kHz にコンフィギュレーションされている場合、ハードウェアは適正にデータを受信しません。よくある間違いの 1 つは、PSoC™ Designer の I²C スレーブ クロック周波数を 100kHz にし、SYSCLK を 6MHz に設定することです。スレーブが 100kHz バス上で動作するという前提が間違っています。SYSCLK が 6MHz であるため、スレーブは 25kHz バスでのみ動作できます。したがって、6MHz の SYSCLK で 100kHz のバスで動作するためには、ユーザー モジュールのプロパティで I²C 速度を 400kHz に選択する必要があります。

SYSCLK の周波数は、PSoC™の電源電圧に依存します。電源電圧が 3V 以上の場合、ほとんどのデバイスの SYSCLK は 24MHz です。電源電圧が 3V 以下の場合、多くのデバイスの SYSCLK は 6MHz です。いくつかのデバイスには、12MHz の SYSCLK のオプションがあります。さらに、SYSCLK 値は PSoC™ Designer 内で変更できます。I²C を使用する際、SYSCLK または CPU_CLK の周波数を変更しないでください。変更すると、I²C ライン上にグリッチを発生させることがあるからです。クロックに関する詳細情報については、テクニカル リファレンス マニュアルのクロック関連セクションを参照してください。

外部クロックを使用する時は、正しい I²C クロック速度が使用されているか確認してください。

6.6 クロック ストレッチおよび割込みレイテンシ

クロック ストレッチはスレーブ デバイスがクロック ラインを LOW に保持するプロセスであり、したがってマスターによるバス上でのさらなる通信を一時停止させます。マスターから受信した情報を処理したり、マスターに送信する追加のデータを準備するために、スレーブ デバイスは通常、クロック ストレッチを行います。クロック ストレッチはトランザクションのどの時点でも行えます。PSoC™は、バイト完了割込みの後にクロック ストレッチを行います。**Figure 20** を参照してください。

I²C 仕様では、これは任意の機能であり、すべての I²C デバイスがクロック ストレッチ機能をサポートする必要はないとされます。しかし、インフィニオンのすべての PSoC™ 1 の I²C スレーブ ユーザー モジュールでは、クロック ストレッチを使用します。クロック ストレッチ機能をサポートしないマスターを使用している場合、バスがロックアップしてリセットに失敗する可能性があります。

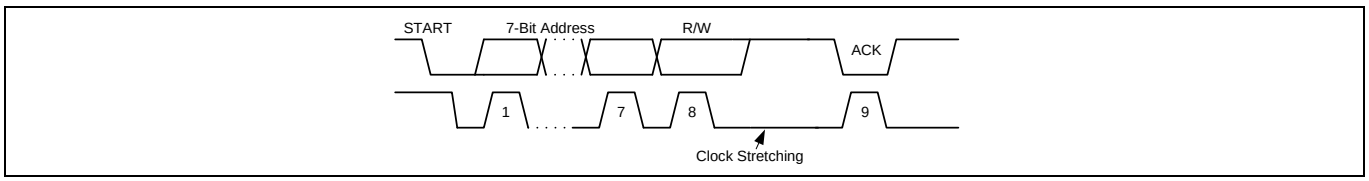

Figure 20 クロック ストレッチ例

PSoC™スレーブがクロック ストレッチに使用する時間は、マスターがスレーブに対して読み出しと書き込みのどちらを行っているのか、またシステム内で他の割り込みが発生するか否かによって異なります。また、CPU 速度にも依存します。

PSoC™ 1 デバイスは、100kHz 以上の I²C バス速度の場合、ほとんどの場合、クロック ストレッチをします。EzI2Cs および I2CHW のユーザー モジュール内の ISR コードは 150～300 の CPU 命令サイクルが含まれています。24MHz の CPU クロックを使用すると、ISR は実行に 6～13µs かかります。SCL ラインは ISR コードの終了時に開放されます。

ISR は、SCL の立ち上りエッジから 3～4 内部サンプル クロック後にトリガされます。**Figure 21** をご参照ください。これは、SCL の内部グリッチ フィルタによるものです。**Figure 22** を参照してください。**Table 4** は、サンプリング クロックの周波数を示したものです。100kHz の場合、サンプル クロックは 1.6MHz

です。これは、ISR は SCL の立ち上りエッジから最大 2.5µs 後に起動されることを意味します。100kHz クロックの公称周期は 10µs です。ISR のトリガに 2.5µs かかり、ISR に 6〜13µs かかる場合、ほとんどの場合においてクロック ストレッチが行われます。

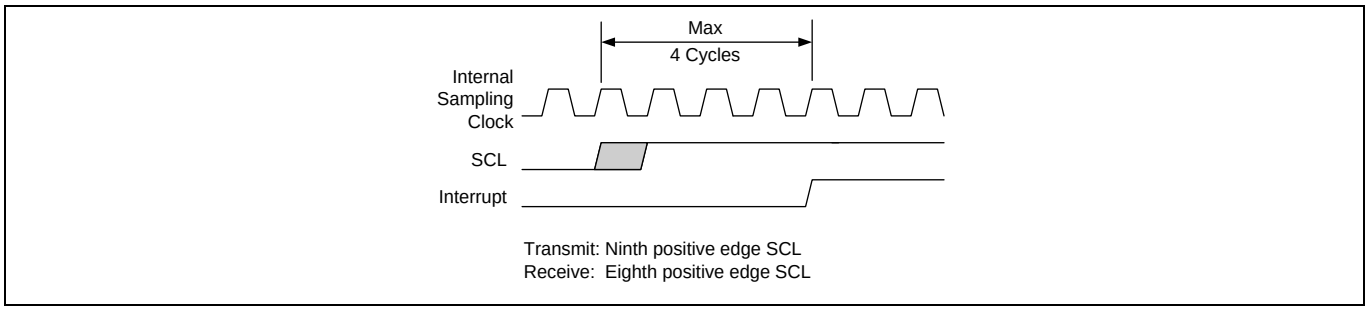

Figure 21　　バイト完了割込みタイミング

この情報を利用して、クロック ストレッチされるかどうか、またその長さはどのくらいであるかを判断できます。しかし、システム内で他の割込みがある場合、その割込みに費やされる時間を考慮しなくてはいけません。

PSoC™内のクロック ストレッチを最小限に抑えるために最初のステップは、CPU を 24MHz で実行させることです。次に、I²C クロック速度を 100kHz 以下にしてください。最後に、割込みを最小限にするとクロック ストレッチは少なくなります。

6.7　　ホット スワッピング

ホット スワッピングは、電源が供給されていない PSoC™に電源が供給されているデバイスを接続するプロセスです。PSoC™内の I²C は、ホット スワッピングが可能な設計になっていません。PSoC™のホット スワッピング時には考慮すべき要素がいくつかあります。最初の問題は電源のバック パワーリングです。PSoC™に電源が供給されておらず、外部ピンのいずれかが HIGH 電圧レベルである場合、PSoC™デバイスにバック パワーリングを起こす可能性があります。PSoC™が予期されない方法で実行され、I²C ラインが悪影響を受けることがあるため、これは望ましくない状況です。

6.8　　グリッチ フィルタ処理

SCL の入力には、**Figure 22** に示すグリッチ フィルタが含まれます。

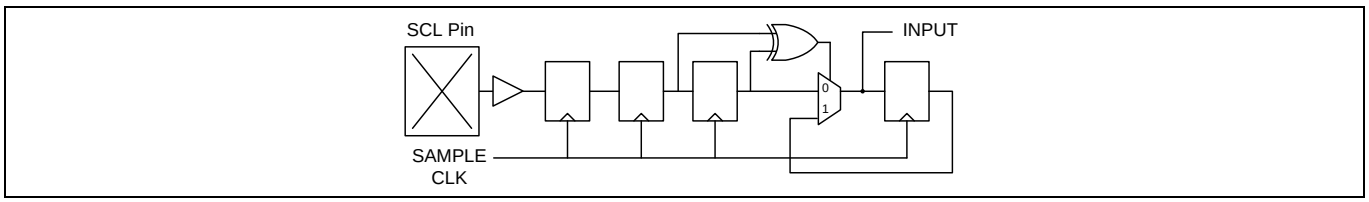

Figure 22　　SCL グリッチ フィルタ

入力はサンプル クロックに二重同期され、その後、グリッチ フィルタ処理されます。信号が通過するためには、生入力と遅延入力が一致する必要があります。サンプル クロックは 1.5MHz または 6MHz であるため、666ns 未満のグリッチと 166ns 未満のグリッチが抑制されます。

6.9 I²C とスリープ

スリープ モードに移行したり復帰したりする設計で I²C を使用する場合、設計上の特別の注意事項を考慮する必要があります。

まず、I²C がスリープ モードに移行する前にすべての I²C トランザクションを完了することが重要です。そうでない場合、デバイスがスリープ モードから復帰した時、I²C ブロックが誤ってデータをアドレスとして、またはアドレスをデータとして解釈する可能性があります。すべての I²C トラフィックが停止したことを確認した後、以下の手順に従ってください。

- I²C ピンを HIGHZ 駆動モードに設定してください。
- I²C ブロックを無効にしてください。一般的には、ユーザー モジュールの Stop() API を読み出すことによってこれが行われます。
- すべての保留中の I²C 割込みをクリアしてください。

上記の条件が満たされた後、PSoC™をスリープ状態に移行できます。デバイスがスリープ状態から復帰した時に、I²C が正常に動作するために、次の手順を実行してください。

- バスに I²C の動作がないことを確認してください。
- 該当する Start API を呼び出してください。
- I²C ピンをオープン ドレインで LOW 駆動になるように設定してください。
- 割り込みを有効にしてください。

上記の手順を実施すると、I²C をスリープ モードと併用する場合にほとんどのエラーを回避できます。

6.10 I²C とダイナミック リコンフィギュレーション

I²C ユーザー モジュールは、絶対にダイナミック リコンフィギュレーションをとおしてロードおよびアンロードを行わないでください。ユーザー モジュールは常時存在する必要があります。ユーザー モジュールのロードとアンロードを行った場合、I²C エラーが発生します。

I²C ユーザー モジュールは、ベース コンフィギュレーション内、または常時ロードされる別のオーバーレイに配置する必要があります。

6.11 I²CHW ユーザー モジュールでのダイナミック スレーブ アドレス指定

EzI2Cs ユーザー モジュールを使用することにより、I²C のスレーブ アドレスをプログラムによりその場で変更できます。ただし、この機能は I²CHW ユーザー モジュールでは使用不可です。同一の機能を実現するには、以下の手順に従って行ってください。

I²CHW ユーザー モジュールでは、マスターからのアドレスは *I2CHW_1int.asm* ファイル内で処理されます。次のコードによりアドレス マッチングを行います。

```
mov A, reg[I2CHW_1_DR]
and F, 0xF9
rrc A
xor A, I²CHW_1_SLAVE_ADDR
```

マスターからアドレスを受信した時、アドレスには 7 ビット アドレスと読み出し／書き込みビットが含まれます。「rrc A」は、読み出し／書き込みビットを除去してから 7 ビット アドレスを I2CHW_1_SLAVE_ADDR 定数と比較します。I2CHW_1_SLAVE_ADDR は I²CHW 用のユーザー モジュール パラメーターで設定された Slave_Addr に応じてデバイス エディタにより作成されるものです。この定数は、*I2CHW_1.inc* ファイルに記述されています。

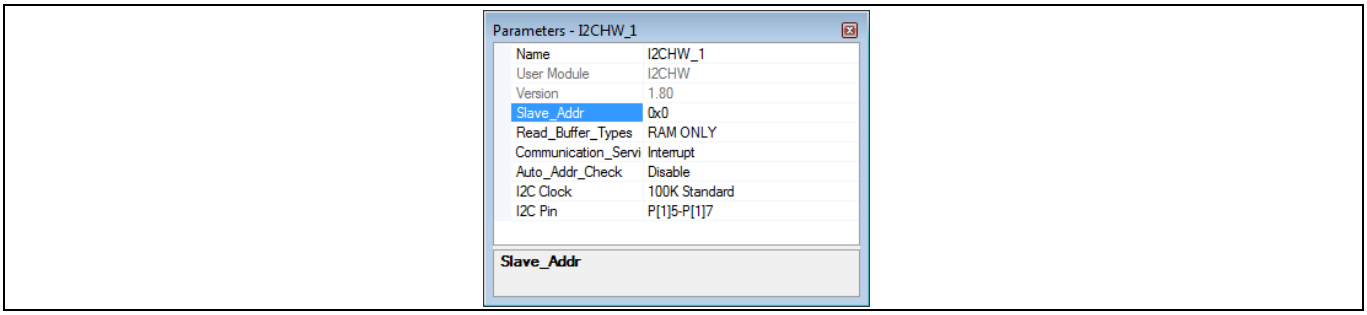

Figure 23 I²CHW ユーザー モジュール スレーブ アドレス

以下の手順に従って定数を RAM の変数と置き換えてください。

1. ダイナミック スレーブ アドレスを保持するための RAM 変数を作成します。*I2CHW_1int.asm* ファイルの変数割り当ての真下にカスタム ユーザー コード エリアがあります。カスタム宣言エリアに、次のコードを追加します。

   ```
   export _I2CSlaveAddress
   export I2CSlaveAddress
   ```

   変数割り当てエリアに、次を追加します。

   ```
   Area InterruptRAM(ram)
   _I2CSlaveAddress:
   I2CSlaveAddress:   BLK   1
   ```

   「_」を含む変数を使用することによって、この変数を C コードで変更できます。

2. アドレス比較を行うコードを修正します。

   ```
   mov A, reg[I2CHW_1_DR]
   and F, 0xF9
   rrc A
   xor A, [I2CSlaveAddress]
   ```

   このコードがカスタム ユーザー コード エリア内にあるため、アプリケーション生成時にユーザー モジュール ライブラリ ファイルへの変更が保存されます。ユーザー モジュールの名前を変更すると、それまでの変更が失われ、変更を再度行う必要があります。

3. *main.c* ファイル (または他の C ファイル) に次のコードを追加して、*I2CHW_1int.asm* ファイルで定義された I² スレーブ アドレスの変数への参照を追加します。

   ```
   extern BYTE I2CSlaveAddress;
   ```

6.12 SCL ラインが LOW のままになる

I²C で発生する一般的な問題は、SCL が LOW のままになってしまうことです。PSoC™ 1 デバイスでこの問題をデバッグ処理して対応するために以下の手順に従ってください。

1. EzI2Cs を PSoC™ Designer 5.0 SP5 でお使いですか？お使いの場合、このバージョンのユーザー モジュールで SCL ラインが LOW になってしまう原因となる既知のバグがあります。最新版の PSoC™ Designer に更新してこの問題を修正します。「**I2C クロックが永久に論理 LOW のまま**」の記事を参照してください。

特別な I2C 注意事項

2. PD5.0 バージョン以前の PSoC™ Designer をお使いですか？お使いの場合、起動時に P1.5、P1.7 にグリッチが生じます。この問題は最新版の PSoC™ Designer では解決されています。詳細については、「**電源投入時の I2C ライン上のグリッチ**」記事を参照してください。
3. マスター デバイスがクロック ストレッチをサポートしますか？サポートしない場合、スレーブとマスター間の通信が非同期になり、SCL が LOW のままになるなど、望ましくない動作が生じる可能性があります。マスターがクロック ストレッチをサポートしない場合、PSoC™がバス上で正常に機能することは保証されません。
4. コードの実行時に CPU クロックか SYSCLK が動的に変更されますか？変更される場合、これは SCL および SDA にグリッチが生じる原因となる可能性があります。これを防ぐには、クロック速度をコードで変更しないでください。
5. コード内で I^2C 割込みのイネーブルとディセーブルを行っていますか？行う場合は、必ず ResumeInt API を使用するようにし、割込みをクリアする EnableInt API を使用しないでください。保留中の I^2C 割込みがあり、その割込みをクリアすると、SCL がいつまでも LOW のままとなります。
6. PSoC™の起動時に、他の I^2C トラフィックがバスに生じる可能性がありますか？そういう可能性がある場合、最新版の PSoC™ Designer で修正されている既知のバグがあります。最新版を持っていなくて更新できない場合、次の回避方法を使用してください: チップ エディタで、I^2C ピンの駆動モードをアナログ High-Z に設定します。メイン コードで、I^2C ユーザー モジュールを有効にしてください。I^2C ピンの駆動モードをオープンドレインのドライブ LOW に設定してください。
7. プロジェクトでスリープ モードが使用されていますか？使用されている場合は、問題を防ぐため、スリープのセクションで説明した手順を実行します。

最新版の PSoC™ Designer では、上記のほとんどの問題について修正と回避策が準備されています。このため、常に最新版の PSoC™ Designer を使用することを推奨します。

7 まとめ

I²C はシンプルな 2 線式の、チップ間のデジタル通信プロトコルです。プロトコルはマスター指向ですが、2 本の通信回線だけで双方向通信が可能です。

インフィニオンの PSoC™では、スレーブ、マスター、およびマルチマスターそれぞれのコンフィギュレーションを含む設計で I²C を実装するためのユーザー モジュールがいくつか提供されます。

PSoC™での I²C 通信は、インフィニオンが推奨する考慮事項に従えば、簡単で信頼性があります。

8 付録 A

この付録は、PSoC™での I²C ハードウェア ブロックの詳細な動作に関心をお持ちのユーザーを対象としたものです。

8.1 ハードウェア レジスタ

レジスタの一部は、I²C ハードウェア ブロックを制御し、その状態を報告します。

I2C_CFG レジスタ: I²C ハードウェア ブロックのコンフィギュレーションを制御します。このレジスタは使用されるピンやクロック速度を制御し、スレーブ機能とマスター機能のどちらが有効かを示します。また、このレジスタは、STOP 条件での割込みやバス エラー時の割込みのサポートを有効にします。

I2C_SCR レジスタ: I²C ハードウェア ブロックからステータス フラグを返します。このレジスタは、フルバイト (完全なバイト) が送信または受信されたかどうかを報告します。このレジスタは、バス エラーが発生したかどうか、およびアービトレーションに負けたかどうかを示します。また、このレジスタは最後のトランザクションがアドレスであるかどうかを判断します。

I2C_DR レジスタ: このレジスタは送受信データの値を保持します。このレジスタは、データがアドレスである場合や、ハードウェア ブロックがスレーブとしてコンフィギュレーションされアドレス指定された場合、またはマスターが読み出しを開始した場合にのみ、データをシフトインします。

I2C_MSCR レジスタ: このレジスタは I²C トランザクションのマスター部分を制御します。このレジスタは START 条件の生成を許可するビットを保持します。I2C_CFG レジスタが利用できるようにするために、そのマスター イネーブル ビットをセットする必要があります。セットしない場合、このレジスタはリセット状態で保持されます。

8.2 ファームウェア要件

ここでは、さまざまなコンフィギュレーションに応じたハードウェア ブロックの動作を説明します。テキストにファームウェアの要件が記載される場合は、すべての PSoC™ I²C ユーザー モジュールがこのコードを実行する点に注意してください。

8.2.1 Start 条件の生成

マスター ステータスと制御レジスタで Start 条件生成ビットがセットされる場合、ハードウェア ブロックが START 条件を生成してデータ レジスタ内のアドレスを送信します。ただし、ハードウェアは、別のデバイスがバスの制御権を獲得したかどうかを認識します。外部の Start 条件が検出された場合、ハードウェアはバスがフリーになるまで現時点の Start 条件をキューに入れます。スタート ビットは、ハードウェアが Start 条件を正常に送信するか、またはファームウェアがそれをクリアするまで、クリアされません。

一度にキューに入れられる Start 条件は 1 つだけです。バスがビジーの時にマスターが 2 つの Start 条件を送信しようとすると、最後のアドレスのみがハードウェアによって送信されます。前述のように、ユーザー コードはこの状況が発生しないようにする必要があります。

8.2.2 マスター動作

マスター モードで、Start 条件とアドレスが送信された後、ハードウェアはスレーブからの ACK/NAK ビットを受信するまで待機します。ACK/NAK ビットを受信すると、ハードウェアが CPU に割込みをかけます。続いて、ファームウェアは、スレーブがアドレスに対して アクノリッジ信号を発行したか否かを判定します。I2C_SCR レジスタでの最後に受信されたビット (LRB) を読み出すことにより、判定します。このビットが 0 なら、スレーブはアクノリッジです。このビットが 1 なら、スレーブはアクノリッジではありません。この条件に応じて、適切なアクションを取る必要があります。

また、ファームウェアは通信方向を設定する役割も担います。SCR レジスタでの送信ビットをセットすることによってこれを行います。ビットに 0 が書き込まれた場合、ブロックは受信モードに入ります。このモードでは、ハードウェアは 8 データ ビットを受信すると CPU に割込みをかけます。この割込み後、ACK/NAK 信号の送信が必要かどうかをファームウェアで判定する必要があります。

送信ビットに「1」が書き込まれた場合、ブロックは送信モードになります。このモードでは、ハードウェアは ACK/NAK 信号がスレーブ デバイスから受信された後に割り込みをかけます。ファームウェアは再度これらのケースを処理します。

SCR レジスタが書き込まれると、マスターはクロックの生成を開始して、さらにデータの送信または受信を行います。

8.2.3 スレーブ動作

スレーブ モードでは、ハードウェアは、START 条件を検出すると、次の 8 データ ビットを I2C_DR レジスタにシフト インします。8 ビット目を受信すると、ハードウェア ブロックは CPU に割込みをかけ、I2C_SCR レジスタのアドレス ビットを HIGH にします。

割込みが発生すると、ハードウェアはクロック ラインを LOW に保持します。受信アドレスを読み出し、アドレスが自分のアドレスであるかどうかを確認するのはファームウェアの役割です。ファームウェアは SCR レジスタの ACK ビットを適切にセットする必要があります。また、ファームウェアはアドレスの読み出し／書き込みビットを読み出す必要があります。ビットが「1」の場合、SCR レジスタの送信ビットは「1」に変更されます。CPU が SCR レジスタに書き込むと、ハードウェアがクロック ラインを解放し、トランザクションの続行が可能となります。

スレーブがトランスミッタとしてコンフィギュレーションされた場合、ファームウェアは有効なデータを DR レジスタへ書き込んでから、SCR レジスタへの書き込みとバス解放を行う必要があります。バスが解放された後、ハードウェアはマスターによって提供されるクロック エッジに合わせて SDA ラインにデータをシフトアウトします。ハードウェアはマスターから ACK/NAK 信号を受信するまで待機して、その後は CPU に割り込みます。その後、ファームウェアは新しいデータをロードするか、または何もしない必要があります。

スレーブがレシーバとしてコンフィギュレーションされた場合、ハードウェアは 8 番目のデータビットが受信された後に割込みを発生させます。次に、ファームウェアはさらにデータを受信できるかどうかを判定します。その際、ファームウェアは ACK ビットを適切に設定する必要があります。

8.2.4 Stop 条件

マスター モードでは、トランザクションが完了した時、ハードウェア ブロックは Stop 条件を生成してバスがフリーであることを示します。

スレーブ モードでは、Stop 条件を受信すると、ハードウェア ブロックは新規の Start 条件を受信するまでアイドル モードになります。

8.2.5 割込みソース

上記のそれぞれの例では、ハードウェア ブロックはバイト完了条件で割込みをかけます。この割込みの発生は通信方向によって異なります。

- 送信: バイト完了割り込み = 9 ビット目
- 受信: バイト完了割り込み = 8 ビット目

ハードウェア ブロックではさらに 2 つの割込み源を使用できます。I2C_CFG レジスタで適切なビットをセットすることにより、この割込みが有効になります。ハードウェアは STOP 条件で割込みを行えます。これはスレーブ モードでの動作時に有用です。この割込みによって、ファームウェアに現時点のト

ランザクションが完了したことを通知します。バスにおいて誤 Start 条件、または誤 Stop 条件のバス エラーが検出されると、次の割込みが発生します。ハードウェアが割込みを検出すると、現時点の動作を停止して割込みを通知します。次いで、ファームウェアがこの状況に対する対処法を決定します。

8.3 アービトレーション

ハードウェア ブロックは、マスターがアービトレーションに負けたかどうかを示すこともできます。これはマルチマスター モードでの動作時に必要です。アービトレーションは、2 つのマスターが同時に書き込みを開始した場合に発生します。ハードウェアは SDA ラインを監視します。マスターがバスを HIGH にしようとした時にもう一方のマスターがバスを LOW に引き下げた場合、マスターがアービトレーションに負けたことを通知します。アービトレーション条件の後は、ハードウェアは SDA ラインの制御を行いませんが、SCL ラインのクロック処理を続行します。次のバイト完了割込み時に、ロスト アービトレーション ビットを確認し、前回の転送の間アービトレーションに負けたかどうかを確認するのはファームウェアの役割です。

8.3.1 I²C の基本フロー

Figure 24 と **Figure 25** に I²C トランザクションの基本フローを示します。これは、PSoC™ 1 に I²C を正しく実装するために必要な基本フローです。提供されるユーザー モジュールでは、このフローに従いますが追加のオーバーヘッドを必要とします。

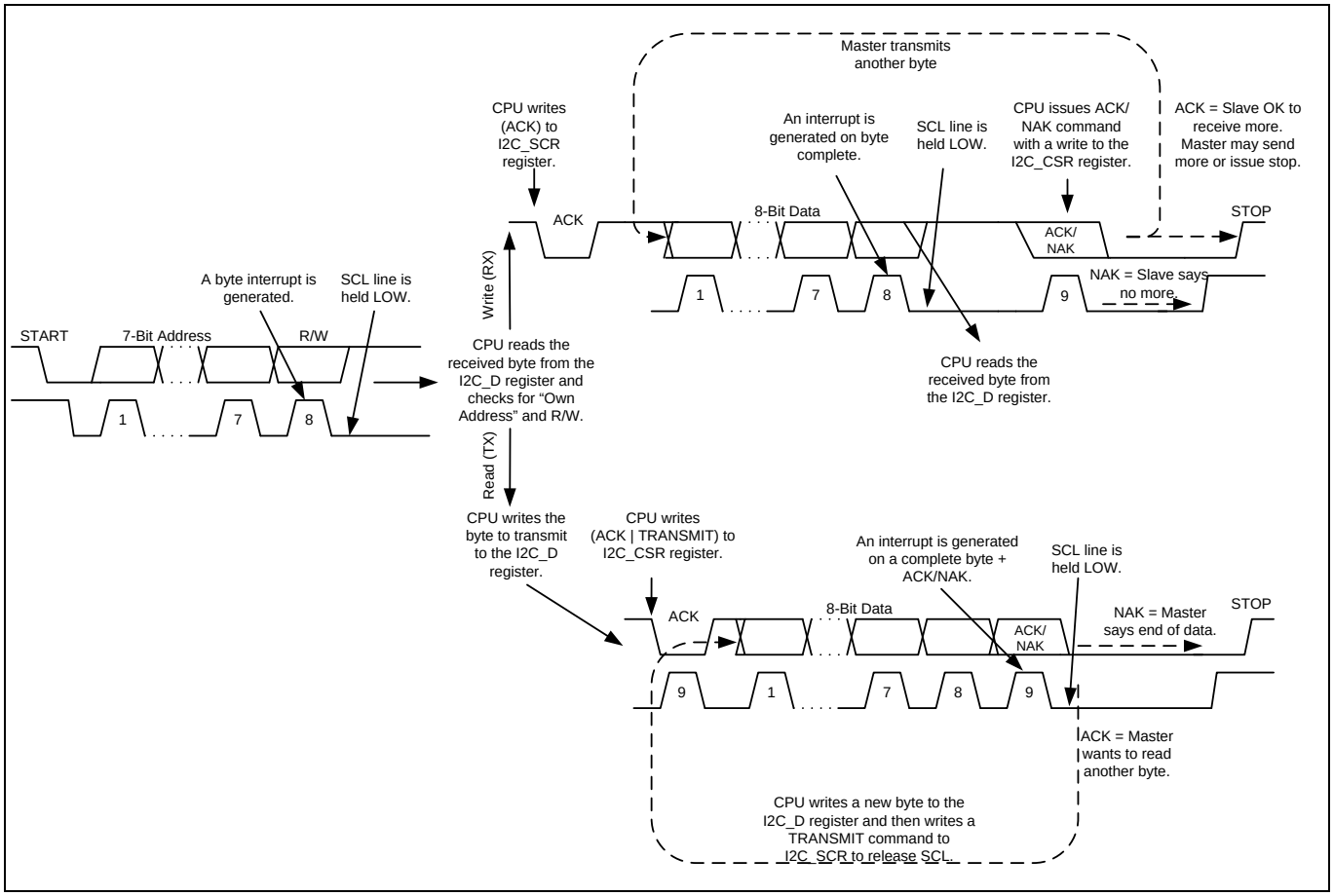

Figure 24 成功したスレーブ トランスミッタ／レシーバのフロー図

付録 A

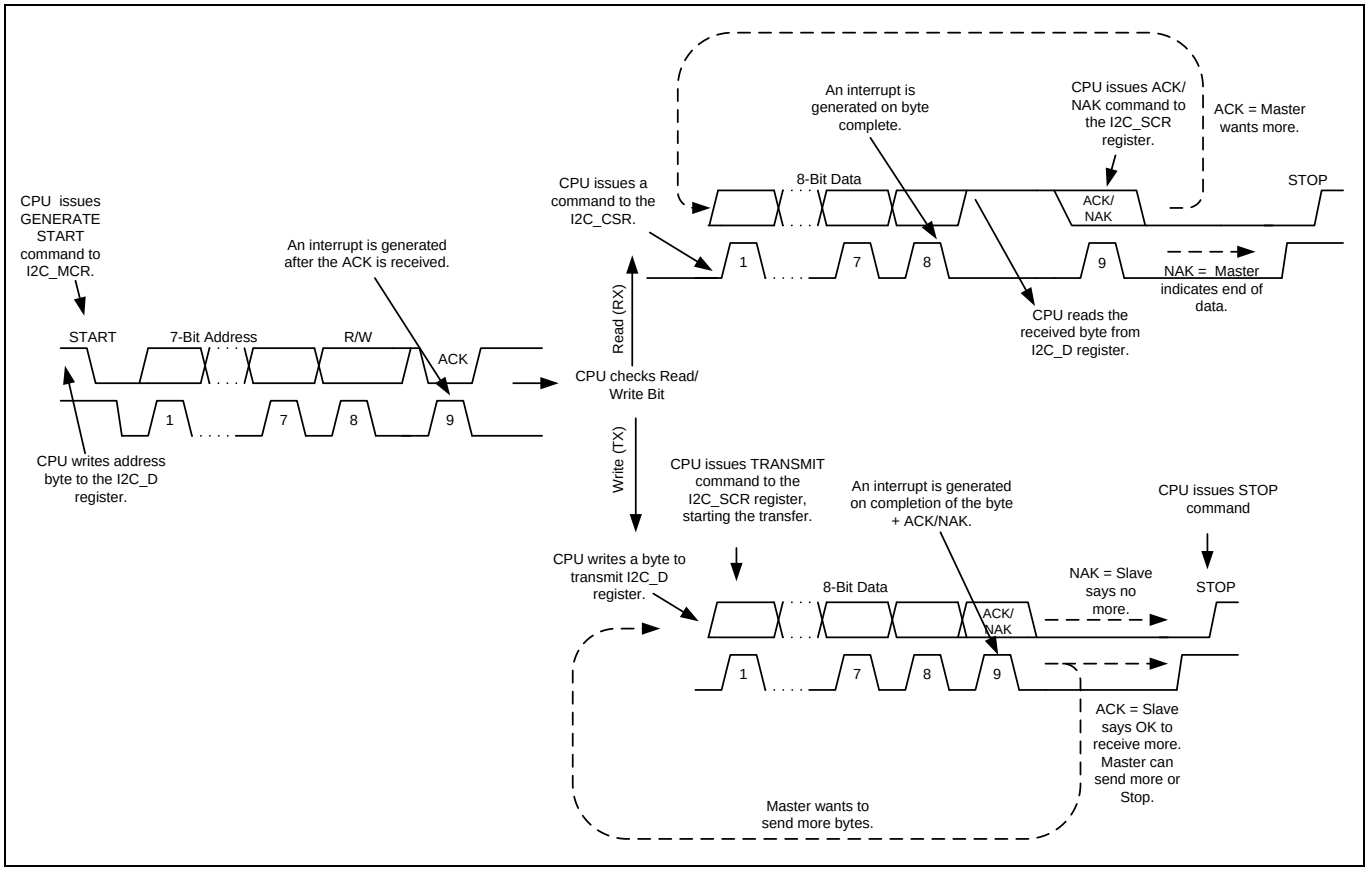

Figure 25　　成功したマスター トランスミッタ／レシーバのフロー図

9 付録 B

ここでは、I²C ユーザー モジュールの使用をデモするさまざまなサンプル プロジェクトを示します。

9.1 EzI2Cs_ADC_LED_DAC サンプル プロジェクト

このプロジェクトは、EzI2Cs ユーザー モジュールをコンフィギュレーションする方法をデモします。プロジェクトは、PSoC™を I²C スレーブとしてコンフィギュレーションし、I²C により制御されるアナログ ペリフェラル デバイスとポート エクスパンダの実装として扱われます。

このプロジェクトは、次のデータ構造を作成します。

```
struct I2CRegs
{
      BYTE LEDValue;/* Updates LEDs */
      BYTE DACValue;/* Updates DAC */
      BYTE ADCValue;/* Reads ADC value */
}I2CRegs;
```

この構造は次の API 呼び出しによって I²C マスターに公開されます。

```
EzI2Cs_1_SetRamBuffer(sizeof(I2CRegs), 2, (BYTE *) &I2CRegs);
```

最初のパラメーターで構造体のサイズを設定します。2 番目のパラメーターは読み書き可能なパラメーターの数を設定します。これ以降の他のパラメーターは読み出し専用です。この構造では、LEDValue と DACValue は読み書き可能で、ADCValue は読み出し専用です。最後のパラメーターは構造体自身へのポインタです。

Figure 26 に示すように EzI2Cs ユーザー モジュールを Chip View でコンフィギュレーションします。

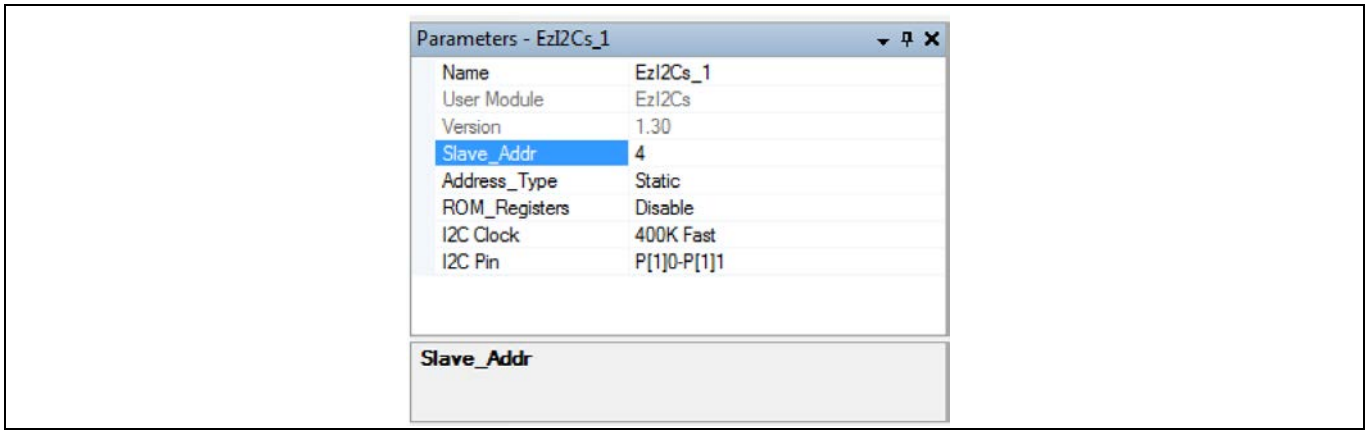

Figure 26 EzI2Cs ユーザー モジュールのコンフィギュレーション

スレーブ アドレスを 0x04 に、アドレス タイプを「static」に設定し、ROM レジスタを無効にし、クロック速度を 400kHz に、高速モードに設定し、P1[0]と P1[1]を I²C ピンとして使用します。P1[0]と P1[1]は、CY3210 PSoC™ 1 評価用基板上に搭載された ISSP ポートを I²C マスターの接続にも使用できるように使用されます。

付録 B

ファームウェアでは、ADC は P0.7 上の電圧を読み出すために使用されます。ユーザーは、ポテンショメーターをこのピンに接続して異なった電圧をシミュレートできます。ADC により読み出された値は、ADCValue 変数に書き込まれます。マスター デバイスは、この変数を読み出せます。

ファームウェアでは、LEDValue と DACValue のローカル コピーが作成されます。I²C レジスタ構成内の LEDValue と DACValue は、これらパラメーターのローカル コピーと連続比較されます。マスターが別の DAC 値か LED 値を書き込んだ場合、LED ポート (P0[0]～P0[3]) と DAC は新しい値に更新されます。DAC 出力は P0[5]に出力されます。

9.1.1 プロジェクトのテスト

Figure 27 に、プロジェクトのテスト用セットアップを示します。CY3210 PSoC™ 1 評価用基板を使ってセットアップを行います。

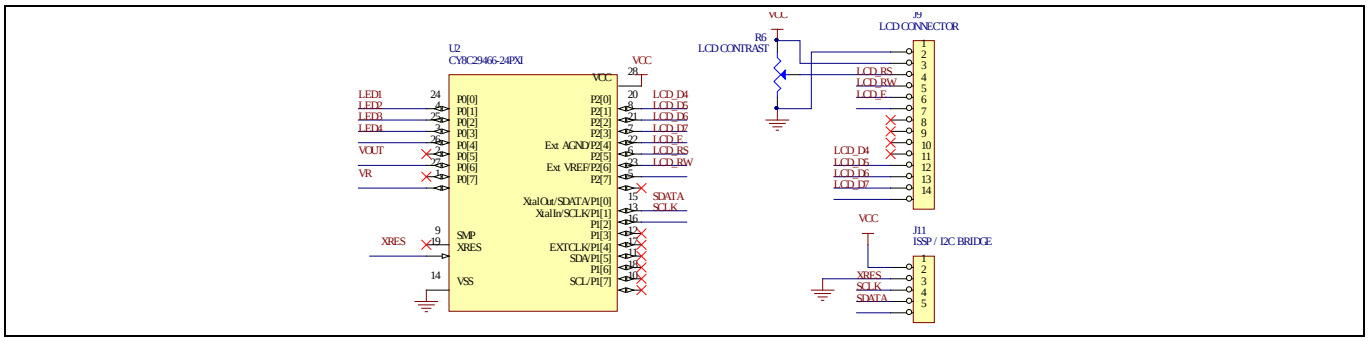

Figure 27 EzI2Cs_ADC_LED_DAC_Project の回路図

P0[0]～P0[3] は、J5 上の LED1～LED4 信号に接続されます。P0[7]は可変抵抗器 (VR) に接続されます。デジタル マルチメーターは、DAC 出力を監視するために P0[5]に接続されます。LCD は LCD コネクターJ9 に接続されます。

CY3217 MiniProg1 または **CY8CKIT-002 MiniProg3** は、CY3210 基板上の ISSP ヘッダーを使ってデバイスをプログラムするために使用されます。**Table 6** をご参照ください。

CY3240 USB-I2C ブリッジ PSoC™開発キットまたは MiniProg3 は、I²C マスターとして使用されることがあります。**Figure 28** にプロジェクトが実行中の CY3210 基板を示します。ここでは CY3240 I²C-USB ブリッジが使用されます。

Table 6 CY3210 評価用基板上のセットアップ

PSoC™ 1 ピン	CY3210 接続	説明
P0[0]～P0[3]	LED1～LED4	P0[3:0]を 4 個の LED に接続
P0[7]	VR	ポテンショメーター入力
P0[5]	-	マルチメーターを接続
-	ISSP ヘッダー (J11)	プログラミング用に MiniProg1 または MiniProg3 を接続
-	ISSP ヘッダー (J11)	I²C 通信のために PC へ CY3240／MiniProg3 を接続

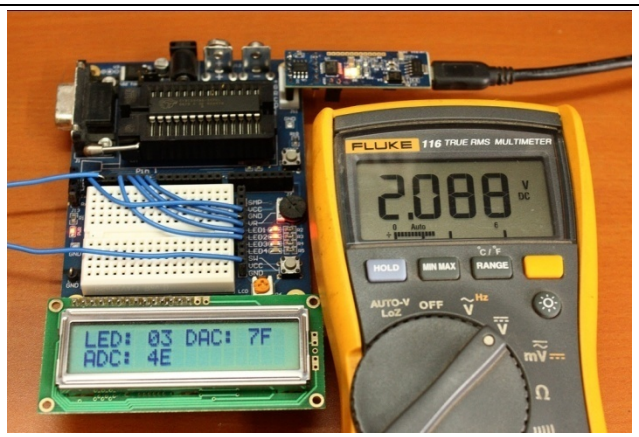

Figure 28　　EzI2Cs サンプル プロジェクトのテスト セットアップ

プロジェクトをテストする 2 つの方法があります。Bridge Control Panel ソフトウェアまたは外部 MCU を使用できます。

9.1.2　　Bridge Control Panel ソフトウェアを使用したテスト

サイプレスの Bridge Control Panel ソフトウェアは、PSoC™ I²C スレーブ デバイスと通信するためにグラフィカル フロントエンドとして機能します。I²C スレーブ インターフェースを持っているプログラムのテスト、チューニング、およびデバッグ処理に有用です。Bridge Control Panel を PSoC™ Designer と PSoC™ Programmer™とともにインストールできます。

デバイスをプログラミングした後、CY3240 または MiniProg3 を ISSP コネクターに接続します。CY3240 と MiniProg3 は、I²C の SDA ラインと SCL ラインに必要なプルアップ抵抗を持ちます。

以下のステップでは、Bridge Control Panel ソフトウェアを使用して ADC の値を読み出し、EzI2Cs_ADC_LED_DAC プロジェクトの LED と DAC 出力を制御する方法を示します。

1. Windows のスタート メニューから Bridge Control Panel プログラムを起動してください。このプログラムは、インフィニオンのフォルダに保存されています。
2. デバイス一覧から MiniProg3 または CY3240 を選択して、接続ボタンをクリックしてください。
3. 次に、電源ボタンをクリックして CY3210 のテスト セットアップに電源を供給してください。(**Figure 29** を参照。)

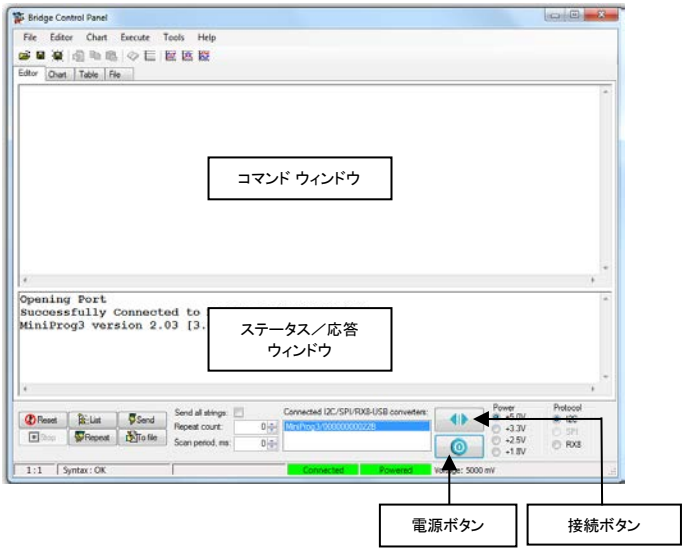

Figure 29 Bridge Control Panel (ブリッジ コントロール パネル)

4. LED と DAC パラメーターに書き込むには、Bridge Control Panel のコマンド ウィンドウに以下のコマンドを入力してください。

```
w 04 00 03 80 p
```

「w」が書き込みコマンド、「04」がスレーブ アドレス、「00」が値が書き込まれるサブアドレス、「03」が LED の値を意味します。LED 値が 03 の場合、LED1 と LED2 は点灯します。「80」は DAC 出力です。この値に応じる DAC 出力は、約 2.09V です。

このコマンドをコマンド ウィンドウに入力して Enter キーを押すと、I²C マスターはコマンドを EzI2Cs スレーブに送信します。結果ウィンドウで LED の状態と P0[5]上の出力を観察します。(**Figure 30** を参照。)

```
w 04+ 00+ 03+ 80+ p
```

各バイトの終わりに付いている「+」符号は、EzI2Cs スレーブがバイトをアクノリッジしたことを示します。「–」符号は、バイトがアクノリッジされなかったことを示します。

例えば、以下のコマンドを試してみてください。

```
w 04 00 03 80 55 p
```

応答は以下のとおりです。

```
w 04+ 00+ 03+ 80+ 55- p
```

EzI2Cs レジスタ構造内の第 3 バイトを読み出し専用バイトとして設定したため、スレーブはこのレジスタへの書き込みをアクノリッジしません (NAK)。

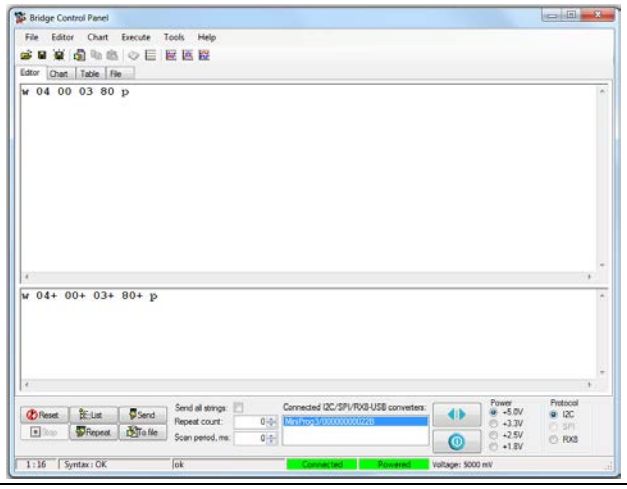

Figure 30　　LED と DAC の値の書き込み

5. ADC 結果を読み出すために、コマンド ウィンドウに以下のコマンドを入力してください。

```
r 04 x x x p
```

「r」が読み出しコマンド、「04」がスレーブ アドレスを意味します。3 つの「*x*」記号は、スレーブから読み出されるバイト数を示します。

ここでは、Bridge Control Panel 内の Repeat (繰り返し) ボタンをクリックして、結果ウィンドウで表示される結果を観察します。(**Figure 31** を参照)

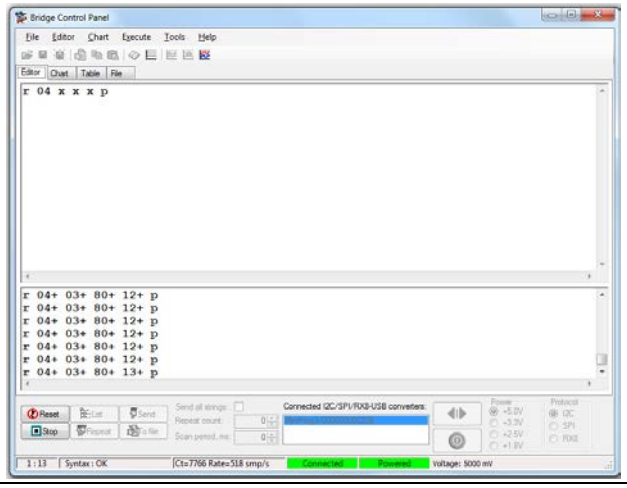

Figure 31　　I²C レジスタをすべて読み出す

応答時の最後のバイトは、ADC 結果です。第 2 と第 3 バイトは、それぞれ LED と DAC パラメーターの値を示します。

6. LED と DAC の値ではなく ADC 値のみ読み出したい場合 (**Figure 32** を参照)、まず、以下のコマンドを実行してください。

```
w 04 02 p
```

この書き込みコマンドは、EzI2Cs 内のサブアドレスを設定します。それ以降の読み出し処理は、ADC の値である 0x0 サブアドレスで行われます。

次に、以下のコマンドを入力して、Repeat (繰り返し) ボタンを選択してください。ADC 値のみが読み出されていることを観察してください。

```
r 04 x p
```

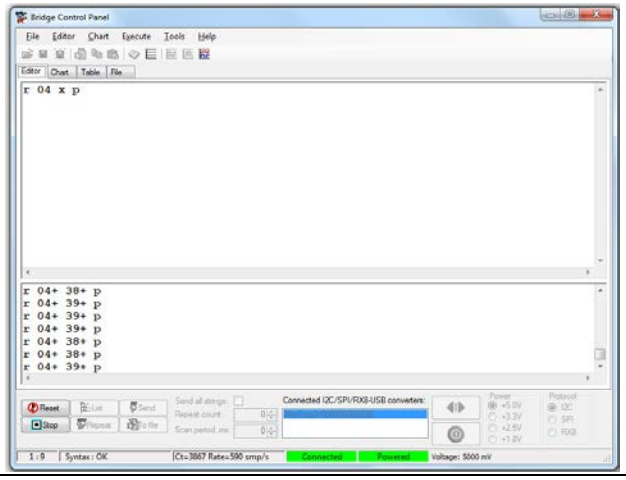

Figure 32 ADC レジスタのみ読み出す

9.1.3 外部 MCU を使ったテスト実施

I²C マスターとして機能する別のマイクロコントローラーを使ってプロジェクトのテストを行うこともできます。ここでは、I²C マスターとしてコンフィギュレーションされた **Arduino Duemilanove** 基板を使った I²C インターフェースをデモし、またユーザーは Arduino プロトタイピング プラットフォームに精通していることを前提とします。この基板と Arduino プラットフォームの詳細情報については、**www.arduino.cc** にアクセスして情報をご参照ください。Arduino 基板を PC に接続し、プログラムをアップロードする方法については、**Getting Started with Arduino** を参照してください。

Arduino デバイスは、専用ピン (AIN4、AIN5) 上に I²C マスター インターフェース、ピン 11 上に 1 つの PWM 出力が存在するようにコンフィギュレーションされます。

CY3210 が I²C スレーブとしてコンフィギュレーションされ、4 ビットのデジタル ディスプレイ (4 個の LED)、1 つのアナログ出力 (DAC)、1 つのアナログ入力 (アナログ入力ピンに接続されるポテンショメーター)、および 1 つの LCD ディスプレイを持ちます。

1. サンプル プロジェクトを CY3210 基板にプログラミングしてください。
2. **Arduino ソフトウェア**を使って付属している Arduino プロジェクト ファイル Arduino_I2C_Master_PSoC1_Slave.ino を開いてください。
3. Tool (ツール) メニューから、「Board」 (基板)、その後、「Arduino Duemilanove」オプションを選択してください。
4. Tool (ツール) メニューから、「Serial Port」 (シリアル ポート) 、その後、基板が接続されているポートを選択してください。
5. USB A-B ケーブルを使ってプログラムを基板にダウンロードしてください。プログラミングが完了した後に USB ケーブルを取り外してください。
6. **Table 7** に従ってハードウェア接続を行ってください。

付録 B

Table 7　　ハードウェア接続

Arduino Duemilanove 上の接続		
ピン	**接続**	**説明**
ピン 11 (PWM)	LED 接続	Arduino 上の PWM 出力
CY3210-PSoC 評価用基板上の接続		
ピン	**接続**	**説明**
P00	J5 上の LED1	デジタル出力ビット 0
P01	J5 上の LED2	デジタル出力ビット 1
P02	J5 上の LED3	デジタル出力ビット 2
P03	J5 上の LED4	デジタル出力ビット 3
P05	マルチメーター／スコープ	アナログ出力 (DAC)
P07	J5 上の VR	アナログ入力 (ADC)
CY3210 と Arduino の接続		
ピン	**接続**	**説明**
Analog in 4 (A4)	J7 上の P10	I^2C 用の SDA
Analog in 5 (A5)	J7 上の P11	I^2C 用の SCL
5V	J5 上の VCC	Arduino 上の 5V ピンを介して PSoC™ 1 基板に電源供給
GND	J5 上の GND	グランド

7. **CY3217 MiniProg1** または **CY8CKIT-002 MiniProg3** を CY3210 基板に搭載される ISSP ヘッダーに接続してください。プログラミングが終わった後、ISSP ヘッダーからプログラマを取り外してください。
8. USB ケーブルを Arduino 基板に再接続してください。これにより、Arduino 基板と CY3210 基板が電源供給されます。**Figure 33** は、セットアップのスナップショットです。

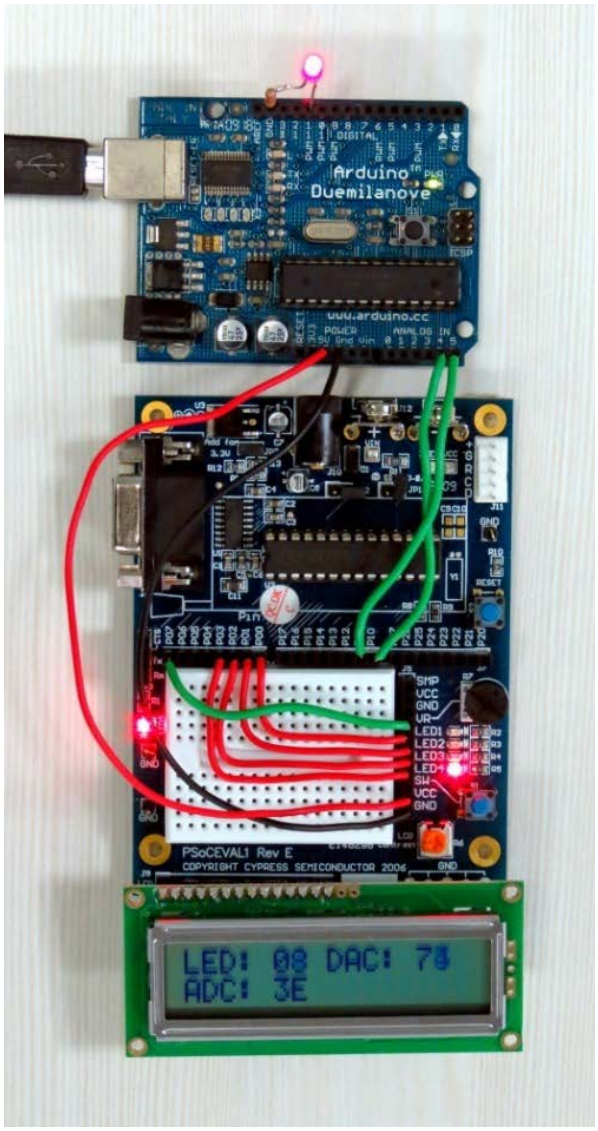

Figure 33　　Arduino と PSoC™間のインターフェース

PSoC™は、(ポテンショメーターに接続される) P07 での電圧を読み出し、値を ADCValue に格納します。Arduino は、I²C を使ってこの値を読み込んで、ピン 11 上の PWM 出力を制御します。CY3210 に搭載されるポテンショメーターを回すと、Arduino 基板上の LED の輝度が変化します。

Arduino は、I²C インターフェースを介して 4 ビット パターンを送信し続けます。PSoC™は、この値を読み出して、それを 4 個の LED に接続される Port0 ピンに書き込みます。Arduino は、PSoC™ DAC によりアナログ電圧に変換される 8 ビット デジタル値を連続して送信します。送信された DAC コードは、0 から 255 までインクリメントされる値です。**Figure 34** に、オシロスコープで表示される時の PSoC™ピン (P05) 上の DAC 出力波形を示します。

そのため、この例は、外部 MCU が I²C を使って PSoC™デバイスとインターフェースする方法を示します。

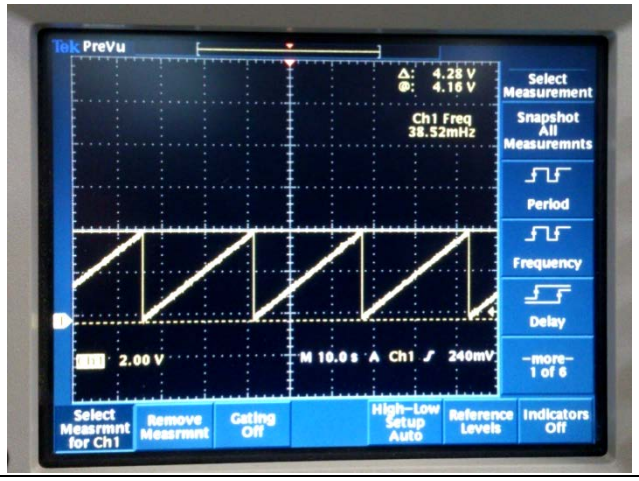

Figure 34 P05 上の DAC 出力波形

9.2 I²CHW スレーブのサンプル プロジェクト

このプロジェクトでは、I²CHW をスレーブとしてコンフィギュレーションする方法をデモします。

プロジェクトでは、I²CHW スレーブに書き込まれたデータがマスターにエコーバックされます。読み出しバッファと書き込みバッファを同じバッファとしてコンフィギュレーションすることにより、これを行います。これを行うには、以下の API を呼び出します。

```
/*When master writes data it will write to rxtxBuffer*/ I2CHW_1_InitWrite(rxtxBuffer,
10);
/*When master reads data it will read from rxtxBuffer*/

I2CHW_1_InitRamRead(rxtxBuffer, 10);
```

メイン コードは、マスター デバイスが I²CHW スレーブに読み出し／書き込みをしたかどうかを確認します。マスター デバイスが読み出し／書き込みをした場合、コードはバッファをリセットして該当するステータス フラグをクリアします。

Figure 35 に、I²CHW ユーザー モジュールのコンフィギュレーションを示します。デバイスのスレーブ アドレスは 0x04 です。

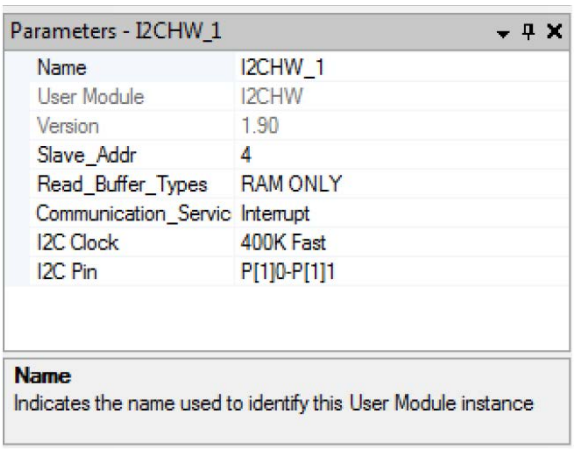

Figure 35 I²CHW ユーザー モジュール スレーブ コンフィギュレーション

9.2.1 プロジェクトのテスト

Figure 36 に、プロジェクトのテストのセットアップを示します。

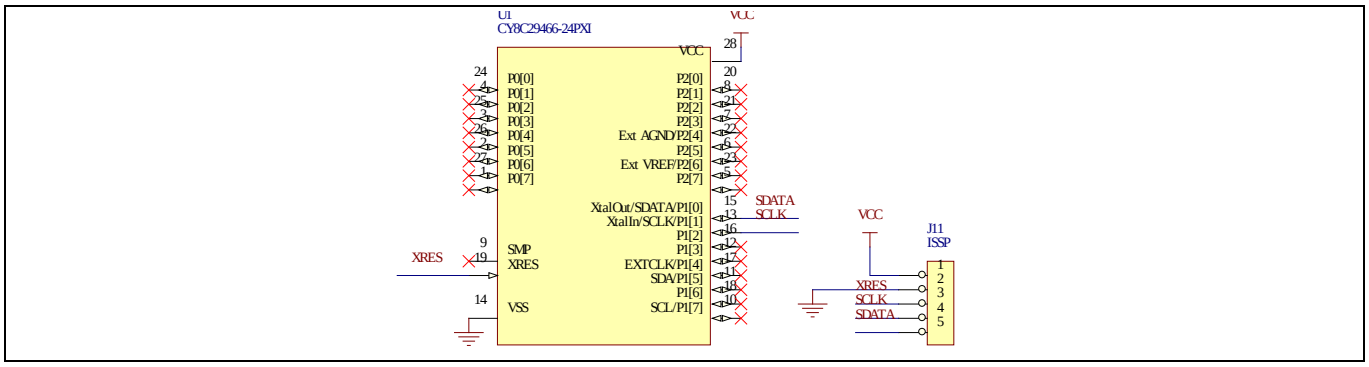

Figure 36 I²CHW_Slave プロジェクトの回路図

プロジェクトのテストに必要な唯一の接続は ISSP 接続です。その ISSP コネクターは、I²C マスターを接続するためにも使用されます。(**Table 8** を参照)

Table 8 プロジェクトのテスト時の接続

PSoC™ 1 ピン	CY3210 接続	説明
-	ISSP ヘッダー (J11)	プログラミング用に MiniProg1 または MiniProg3 を接続
-	ISSP ヘッダー (J11)	I²C 通信のために PC へ Cy3240 または MiniProg3 を接続

CY3210 PSoC™ 1 評価用基板を使って I²CHW スレーブ プロジェクトをテストできます。

CY3210 基板に搭載されている ISSP ヘッダーを使うことにより **CY3217 MiniProg1** または **CY8CKIT-002 PSoC™ MiniProg3** を使ってデバイスをプログラムしてください。

CY3240 USB-I2C ブリッジまたは MiniProg3 を I²C マスターとして使用してください。

Figure 37 に、Bridge Control Panel を使って I²CHW スレーブから／へ値を読み書きする方法を示します。**Bridge Control Panel** のセットアップ方法に関する簡単な説明については、EzI2Cs_ADC_LED_DAC サンプル プロジェクトを参照してください。

1. Bridge Control Panel のコマンド ウィンドウに以下のコマンドを入力してください。

```
w 04 00 01 02 03 04 05 06 07 08 09 p

r 04 x x x x x x x x x x p

w 04 0a 0b 0c 0d 0e 0f 10 11 12 13

r 04 x x x x x x x x x x p
```

2. 「Send all string」 (すべての文字列を送信) オプションを選択し、送信ボタンをクリックしてすべてのコマンドを順序どおりに実行してください。

結果表示ウィンドウで、書き込みコマンドを実行している時に書き込まれる値が、読み出しコマンドを実行している時に読み出される値と同じであることを確認してください。

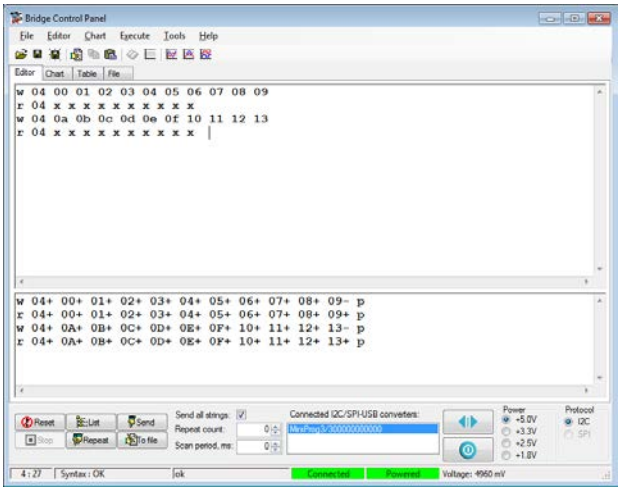

Figure 37 I²CHW スレーブの書き込みと読み出し

9.3 I²CHW マスターのサンプル プロジェクト

このプロジェクトでは、I²CHW ユーザー モジュールをマスター モードで使用する方法をデモします。具体的に、このプロジェクトは、I²CHW ユーザー モジュールを使って外部 I²C EEPROM から／へ読み書きする方法を説明します。

Figure 38 に、I²CHW ユーザー モジュールをコンフィギュレーションする方法を示します。

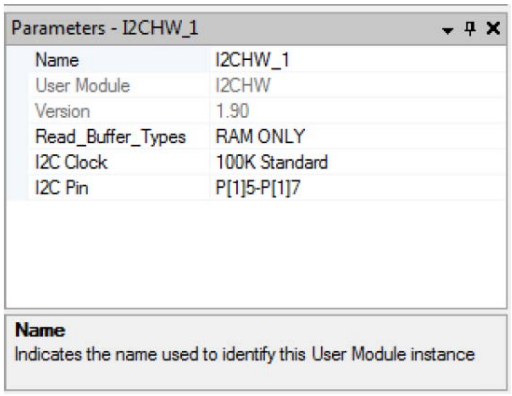

Figure 38 I²CHW マスター コンフィギュレーション

メイン コードで、I²CHW ユーザー モジュールが初期化されます。その後、66 バイトを RAMBuffer アレイから EEPROM へ書き込みます。最初の 2 バイトは、データが EEPROM へ書き込まれた場所を示すサブアドレスです。このプロジェクトは、ページ サイズが 64 バイトの 32kB EEPROM でテストされます。より小容量の EEPROM に書き込む場合は、当該 EEPROM のページ サイズに書き込み量を制限してください。

EEPROM にデータが書き込まれると、EEPROM は書き込みサイクルに入ります。この間、スレーブは、どの I²C トランザクションに対しても ACK 信号を生成しません。EEPROM が書き込みサイクルを完了するまで、他の処理はできません。これを検出するために、マスターは、継続的にスタートを送信し、ACK ステータスを確認する while ループに入ります。

```
while(!(I2CHW_fSendStart(0x50, I2CHW_READ)))
{
```

付録 B

```
I2CHW_SendStop();
}
```

スレーブが ACK 信号を生成すると、マスターは while ループを抜けます。

次に、マスターは EEPROM に書き込んだばかりのデータを読み出します。最初にサブアドレス用の 2 バイトを書き込み、次いで EEPROM から 64 バイトのデータを読み出すことによってこれを行います。次に、マスターは読み出しデータを書き込みデータと比較します。すべての 64 バイトが一致すると、LED は点灯します。

このプロジェクトをテストするには、P1.5 と P1.7 を外部 I²C EEPROM の SDA ラインと SCL ラインに接続します。これらのラインに外部プルアップ抵抗器が取り付けられていることを確認してください。直列に抵抗を付けて LED を P0_7 に接続してください。CY3210 PSoC™評価用基板を使ってプロジェクトをテストできます。(**Table 9** を参照)

Table 9　　プロジェクトのテスト時の接続

PSoC™ 1 ピン	CY3210 接続	説明
P1[5]	V_{DD} に接続する 2.2kΩ プルアップ抵抗	24C256 IC の SDA ラインに接続
P1[7]	V_{DD} に接続する 2.2kΩ プルアップ抵抗	24C256 IC の SCL ラインに接続
P0[7]	LED1	LED インジケータ

Figure 39 に、プロジェクトのテストのセットアップを示します。

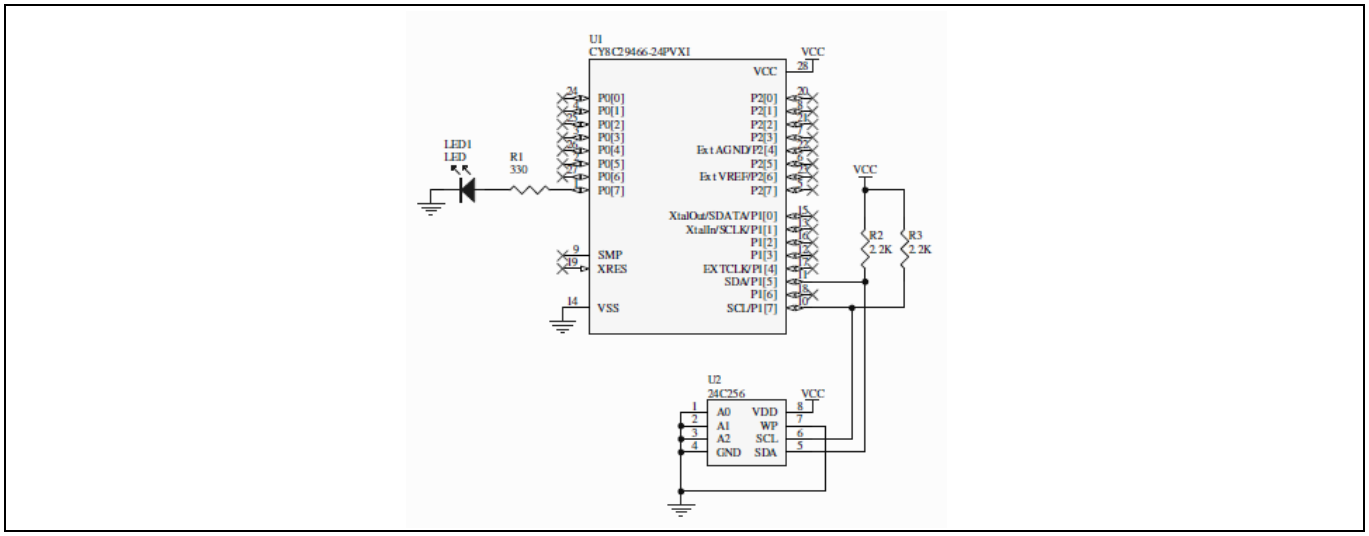

Figure 39　　I2CHW_Master のテスト回路図

9.4　サンプル プロジェクトの移行

このプロジェクトは、PSoC™ Designer に内蔵されたクローン機能を使用して簡単に他のデバイスへ移行できます。例えば、EzI2Cs_ADC_LED_DAC プロジェクトを移行するには、以下の手順を行ってください。

1. PSoC™ Designer を開いて新規プロジェクトを作成してください。
2. Project Creation (プロジェクト作成) パラメーターで「Clone」 (複製) オプションを選択してください。

付録 B

3. 「Clone from Project」 (プロジェクトから複製) パラメーターでは、EzI2Cs_ADC_LED_DAC プロジェクトのプロジェクト フォルダ内の EzI2Cs_ADC_LED_DAC.cmx ファイルを選択してください。「Device Catalog」 (デバイス カタログ) ボタンをクリックしてこのプロジェクトを実行したいデバイスを選択してください。
4. デバイスを選択した後、OK をクリックしてください。PSoC™ Designer は、プロジェクトを新しいデバイスへ移行します。(**Figure 40** を参照。)

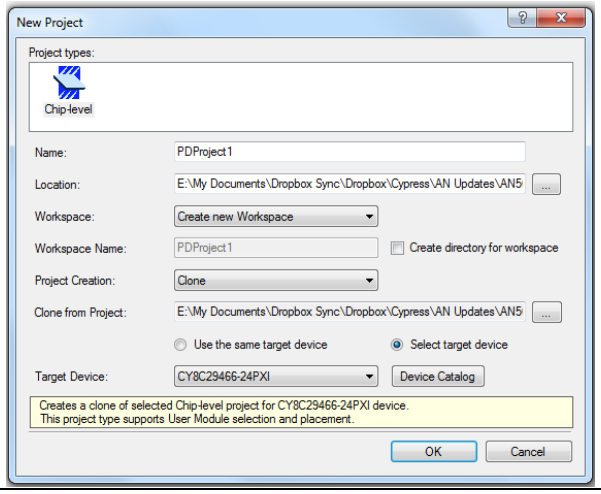

Figure 40　　プロジェクトの移行

改訂履歴

Document version	Date of release	Description of changes
**	2012-09-03	これは英語版 001-50987 Rev. *C からを翻訳した日本語 001-82520 Rev. **です。
*A	2014-11-24	これは英語版 001-50987 Rev. *D からを翻訳した日本語 001-82520 Rev. *A です。
*B	2015-10-23	これは英語版 001-50987 Rev. *E を翻訳した日本語版 001-82520 Rev. *B です。
*C	2021-07-19	これは英語版 001-50987 Rev. *F を翻訳した日本語版 001-82520 Rev. *C です。

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

Edition 2021-07-19
Published by
Infineon Technologies AG
81726 Munich, Germany

Do you have a question about this document?
Go to: www.cypress.com/support

Document reference
001-82520 Rev. *C

重要事項

本文書に記載された情報は、いかなる場合も、条件または特性の保証とみなされるものではありません（「品質の保証」）。本文に記された一切の事例、手引き、もしくは一般的価値、および／または本製品の用途に関する一切の情報に関し、インフィニオンテクノロジーズ（以下、「インフィニオン」）はここに、第三者の知的所有権の不侵害の保証を含むがこれに限らず、あらゆる種類の一切の保証および責任を否定いたします。

さらに、本文書に記載された一切の情報は、お客様の用途におけるお客様の製品およびインフィニオン製品の一切の使用に関し、本文書に記載された義務ならびに一切の関連する法的要件、規範、および基準をお客様が遵守することを条件としています。

本文書に含まれるデータは、技術的訓練を受けた従業員のみを対象としています。本製品の対象用途への適合性、およびこれら用途に関連して本文書に記載された製品情報の完全性についての評価は、お客様の技術部門の責任にて実施してください。

本製品、技術、納品条件、および価格についての詳しい情報は、インフィニオンの最寄りの営業所までお問い合わせください（**www.infineon.com**）。

警告事項

技術的要件に伴い、製品には危険物質が含まれる可能性があります。当該種別の詳細については、インフィニオンの最寄りの営業所までお問い合わせください。

インフィニオンの正式代表者が署名した書面を通じ、インフィニオンによる明示の承認が存在する場合を除き、インフィニオンの製品は、当該製品の障害またはその使用に関する一切の結果が、合理的に人的傷害を招く恐れのある一切の用途に使用することはできないこと予めご了承ください。