I²C ハードウェア ブロックのデータシート I2CHW V 1.1

リソース	PSoC® ブロック				API メモリ（バイト）		センサー当たりのピン数
	CapSense®	I2C/SPI	タイマ	コンパレータ	Flash	RAM	
CY8C20x66、CY8C20x36、CY8C20336AN、CY8C20436AN、CY8C20636AN、CY8C20x46、CY8C20x96、CY7C60413、CY7C645xx、CY7C643/4/5xx、CY7C60424、CY7C6053x、CYONS2010、CYONS2011、CYONSFN2051、CYONSFN2053、CYONSFN2061、CYONSFN2151、CYONSFN2161、CYONSFN2162、CY8CTST200、CY8CTMG2xx、CY8CTMG30xx							
スレーブのみ		x			279-440	8	2
拡張スレーブ (CY8C20x66、CYONS2xxx、CYONSFN2xxx、CYONSTB2xxx)		x			170-200	1	2

特徴と概要

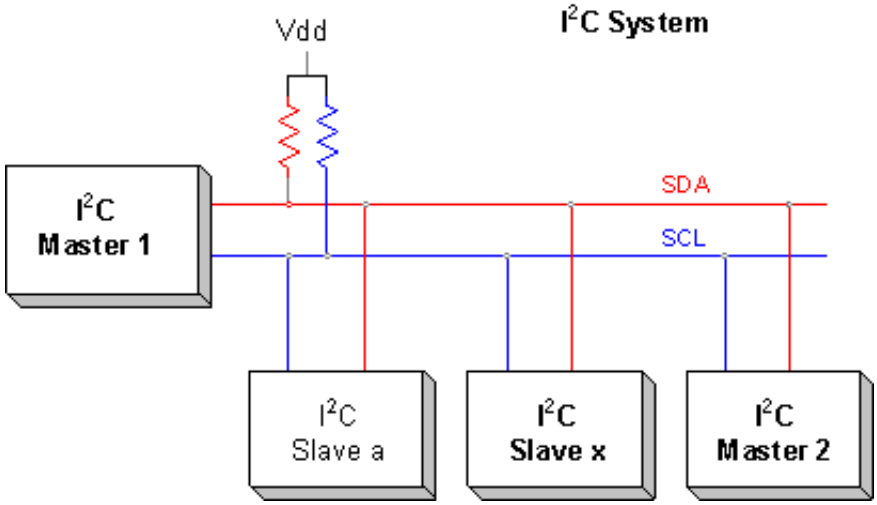
- 業界標準の Philips I²C バス互換インターフェイス
- スレーブ専用の動作
- 2 つのピン (SDA および SCL) のみで I²C バスへのインターフェイスを実現
- 100/400 kbits/s 標準のデータ伝送速度に加え、50 kbits/s をサポート
- ユーザ プログラミングを最小限に抑える高レベル API
- 7-bit アドレッシングモード、10-bit サポートされるアドレス指定

拡張スレーブの特徴：

- 7-bit ハードウェア アドレス比較のサポート
- 柔軟なデータ バッファ方式
- 「バス ストールなし」の動作モード
- 低消費電力のバス監視モード

I²C ハードウェアユーザーのモジュールは、ファームウェアの I²C スレーブのデバイスを実装しています。I²C バスは、Philips® によって開発された業界標準の 2 つの 配線のハードウェアインタフェースです。マスタは、I²C バスですべての通信を開始し、すべてのスレーブのデバイスのためにクロックを提供します。I2CHW ユーザ モジュールは、最高 400 kbits/s までの速度を持つ標準モードをサポートし、同じバス上の他のスレーブ デバイスと互換性があります。I2CHW ユーザ モジュールは、リソース表にあるデバイスで利用可能な拡張スレーブ関数とその他の PSoC デバイスでサポートされているスレーブ関数をサポートします。

Figure 1. I²C ブロック図



機能説明

このユーザ モジュールは、I²C ハードウェアリソースのサポートを提供します。CPU クロックを 12 MHz に構成する場合は、データを 50/ 100/400 kbits/s の速度で転送できます。より遅い CPU クロックを使用することもできますが、拡張スレーブ機能を使用しない場合に、アドレスとデータの処理中にバス ストールが発生する場合があります。拡張スレーブ機能を使用すると、ストールは発生しません。

I²C 仕様により、マスタが 100 kHz のクロック速度で DC で駆動できるようになります。ハードウェアリソースへの直接アクセスを提供するため、SDA (シリアルデータ) と SCL (シリアルクロック) という異なる選択肢があります。提供されている API では、7 ビットのアドレス モードがサポートされています。10-bit アドレス モードは、API セットへのユーザ拡張でサポートされます。

I²C バスについての詳細な説明と、リソースの実装方法は、デバイスのデータシートで参照できます。また、インターネットでも検索可能です。

I²C スレーブ

I²C リソースは、バイト単位のレベルでのデータ転送機能をサポートしています。各アドレスやデータの送受信が終了した時点での状態が報告され、専用の割り込みをトリガすることがあります。状態の報告と割り込み信号の生成は、ハードウェアで検出されたデータの転送方向と I²C バスの条件によって異なります。割り込みは、バイトの完了時およびバスの エラー検出時に発生するよう構成できます。

すべての I²C 処理作業は、開始、アドレス、R/W の方向、データ、および終了で構成されています。このユーザ モジュールで使用されている I²C リソースは、I²C スレーブのみで動作可能です。このユーザ モジュールは、バッファリングされた転送メカニズムに基づいて割り込みを提供します。通信は、最優先関数呼び出し機能から開始されます。個々のメッセージのバイトが終了した時点で、割り込みがトリガされて I²C バスがストールされ、提供された割り込みサービス ルーチン (ISR) が通信を継続できるよう、実行される初期化により異なる、バス上の適切なアクションを取ります。アドレスを認識しないスレーブのデバイスは、次のアドレスが受信されるまで、再割り込みされることはありません。スレーブ デバイスは、応答 (ACK) または応答しない (NAK) のいずれかにより、各アドレスに応答する必要があります。

マスタとスレーブの間の違いを無視する場合は、2 つの一般的なケース、つまり、受信機のケースと送信機のケースが存在します。I²C 受信機の場合は、受信データの 8th ビットの後で割り込みが発生しま

す。この時点で、受信デバイスは、アドレスまたはデータであるかにより、受信バイトの ACK または NAK を決定する必要があります。次に、受信のデバイスの適切な制御ビットを I2C_SCR レジスタに作成して、I²C リソースに ACK または NAK 状態を通知します。バスをアンインストールして ACK または NAK 状態をバスに配置し、次のデータバイトにシフトすることで、I2C_SCR レジスタへの書き込み速度がバス上のデータ フロー速度と一致されます。2 番目の送信機の場合には、外部の受信デバイスが ACK または NAK を提供した後で割り込みが発生します。このビットの状態を判別するために I2C_SCR を読み取ることができます。送受信機の場合、データが I2C_DR レジスタにロードされ、I2C_SCR レジスタが、次の転送の部分をトリガするために再度書き込まれます。

バッファされた読み取りおよび書き込みルーチン (bWriteBytes()、bWriteCBytes()、fReadBytes()) を使用する場合、バッファの初期化関数 (InitWrite(...)、InitRamRead(...)、InitFlashRead(...)) を使用する必要はありません。これらの関数は、バッファされる読み取りまたは書き込み (bWrite バイト (...)、bWriteC バイト (...)、fRead バイト (...)) を開始する関数の呼び出しの一環として呼び出されます。

I²C 拡張スレーブ

拡張スレーブは、バスをストールせずにハードウェアのアドレスを比較できるよう、ハードウェアの機能に API のサポートを追加します。I²C との直接的な読み取りと書き込みを可能にするよう、32-byte RAM バッファも含まれます。このバッファは、I²C ブロックの一部で、従来の SRAM には依存しませんが、I2CHW ブロックまたは CPU のいずれかからアクセスできます。低消費電力のバス監視機能を使うと、PSoC がスリープ状態の場合に、拡張 I²C スレーブ ブロックがバスを監視できるようになります。これらの機能を合わせると、バスのストールを発生しない柔軟な I²C 通信が可能になり、I²C バスを使用している際の低消費電力性能を高めることができます。

I²C 拡張スレーブ機能は、32-byte RAM バッファ インターフェースを備えたスレーブを提供します。この外部マスタは、RAM バッファとのアドレスの読み取りと書き込みを制御します。CPU の読み取りと書き込みは、RAM バッファがデュアルポートの RAM バッファのように機能するよう、マスタの読み取りと書き込みには依存しません。データが常に利用可能であるため、バスがストールすることはありません。

バッファへのアドレス ポインタとして使用されるレジスタは 4 つあります。このうち 2 つは外部マスタで制御されます。また、1 つは、ベースポインタ (I2C_BP) で、後の 1 つは現在のポインタ (I2C_CP). です。この 2 つのポインタは、マスタが 1 またはそれ以上のバイトを書き込む際に、セットを受け取ります。マスタが書き込む際、送信される最初のバイトは、常にベース ポインタのアドレスと現在のポインタが最初に設定されるアドレスです。2 つ目のバイト (データの最初のバイト) は、現在のポインタのアドレスにロードされ、現在のポインタが自動的に増分され、3 つ目のバイト (データの 2 つ目のバイト) のアドレスを設定します。バッファにバイトが書き込まれるたびに、現在のポインタが増分されます。マスタが読み取り動作を実行すると、現在のポインタがベース ポインタと同じ値に設定され、それに続くバイトがバッファから読み取られるたびに、増分されます。ベース ポインタは、マスタの書き込みが発生する際にのみ設定されます。マスタが許可された範囲外でデータを読み取ろうとすると、無効なデータが返されます。マスタが、許可された範囲外でデータを書き込もうとすると、データが破棄されるため、RAM 領域外には影響を与えません。

他の 2 つのレジスタはファームウェアによって制御され、同じ 32-byte RAM バッファへのアドレス ポインタとしても使用されます。1 つはベース ポインタ (CPU_BP) で、もう 1 つは現在のポインタ (CPU_CP). です。これらの 2 つのポインタは、前述のアドレス ポインタ (I2C_BP および I2C_CP). と同様に機能します。CPU の読み取りと書き込みを行う際に使用されるもう 1 つのレジスタ (I2C_BUF) があります。このレジスタが書き込まれると、データが現在のポインタ (CPU_CP). によってポイントされる場所でバッファに転送されます。このレジスタが使用されると、現在のポインタにあるデータが返されます。RAM の内容が初期化されない場合は、有効なデータは返されません。

設計上の考慮事項

スレーブのデバイスは、マスタによってアクセスされるように残されたバッファ領域の内部カウントを保持します。変数は I2CHW_Read_Count および I2CHW_Write_Count という名前になります (ここで、I2CHW は、PSoC Designer のユーザ モジュールのインスタンス名で置き換えられます)。変数はグローバルで、適切な「extern」宣言文を含めることにより、C からアクセスされます。カウント変数の初期値からカウント変数の現在の値を差し引いて、マスタによる読み取りまたは書き込みのバイト数を決定します。スレーブ (のみ) ユーザ モジュールの場合は、カウント変数の初期サイズが、関数 I2CHW_InitWrite、I2CHW_InitRamRead、I2CHW_InitFlashRead を使用して設定されます。MultiMasterSlave ユーザ モジュールで使用されるオプション スレーブのカウント値を設定する関数の呼び出しは、I2CHW_InitSlaveWrite、I2CHW_InitSlaveRamRead、I2CHW_InitSlaveFlashRead. です。

I^2C スレーブ API 内のデータを読み取る、または書き込むにはバッファが使用されます。I^2C スレーブを有効にする前に、適切なバッファを初期化する必要があります。I^2C マスタによって読み取り、または書き込みが開始されると、適切な状態ビットが I2CHW_Status バイトに設定されます。スレーブ内の最優先的なプロセスは、書き込みバッファに保管されるか、読み取りバッファから抽出されます。ISR を使用する場合、スレーブのデータ転送ルーチン (ISR) は、指定された長さを超えるバッファのアクセスを許可していません。バッファの読み取りおよび書き込みは、以下の方法で処理されます。

- I^2C マスタがバッファに含まれているよりも多くのデータを読み取ろうとする場合、I^2C マスタが読み取りを中止するまで、最終のバイトが再送信されます。(I^2C プロトコルは、マスタが読み取りを停止するように I^2C スレーブの方法を指定しません。)
- I^2C マスタがデータを受信して、I^2C スレーブに書き込み、利用可能なストレージ領域がないと判断すると、最終のバイトを受信するときにスレーブが NAK を生成します。I^2C マスタがデータの書き込みを継続すると、スレーブは NAK を継続します。最初の NAK が生成されると (データが最後の利用可能な位置に保存される)、データはそれ以上保存されません。
- バッファがゼロの長さで定義されている場合には、I^2C スレーブに書き込まれたデータは NAK されますが、保存されません。Flash でデータを直接読み取ることができる機能を有効にすると、RAM または Flash バッファのいずれかを使用できます。データ転送の ISR が Flash/ROM や RAM にある読み取りバッファを使用する場合は、提供されている API を使用して、これを構成します。

動的再構成

I2CHW リソースを動的にロード / アンロードされたオーバーレイに統合することは推奨できません。I2CHW リソースは、ベース設定の一部として配置してください。動作条件に示されているように、I2CHW ブロックの操作を変更してください。ただし、動的再構成の一環として、リソースを削除しようとする場合には、外部の I^2C デバイスに好ましくない影響を与える可能性があります。

I^2C アドレス指定

I^2C アドレスは、読み取りまたは書き込み処理に対する最初のバイトの上位 7 ビットに含まれています。このバイトは、スレーブをアドレスする I^2C マスタによって使用されます。有効な選択肢は、0-127(dec) からです。バイトの LSB には R/~W ビットが含まれます。このビットが 0 の場合はアドレスが書き込まれ、LSB が 1 である場合はアドレスされたスレーブがそれから読み取ったデータを持ちます。

内部的に、ユーザ モジュールは、入力アドレスを使用し、これをシフトして読み取り / 書き込みビットと結合させ、完全なアドレス バイトを形成します。

例

0x48 アドレスがパラメータとして渡されるか、スレーブ アドレスとして指定されます。個々のパラメータが渡されます。このパラメータには、読み取り / 書き込み情報が含まれます。I²C マスタは 0x90 のバイト (8-bits) を送信してスレーブにデータを書き込み、バイト 0x91 を送信してスレーブからデータを読み取ります。

スレーブのモジュールは、アドレス パラメータの数値入力に基づく 10 進数を受け入れるため、7-bit アドレスを 10 進数 (10 進数の 72) で入力してください。

DC および AC の電気的特徴

I²C インターフェースの電気的特性については、PSoC デバイスのデバイス データシートを参照してください。

回路図で示されているように、I²C バスには外部プルアップ レジスタが必要です。プルアップ レジスタ (R_P) は、供給電圧、クロック速度、バス静電容量によって決定されます。特定のデバイス (マスタまたはスレーブ) の最小シンク電流は、出力電圧が V_{OLmax} = 0.4V の場合 3 mA 以上です。これは、5V システムの最小プルアップ レジスタ値をおよそ 1.5 kΩ に制限します。R_P の最大値は、バスの容量、およびクロック速度に依存します。150 pF のバス静電容量を持つ 5V のシステムでは、プルアップ レジスタは、6 kΩ を超えません。I²C バスの仕様について詳しくは、NXP のウェブサイト (www.nxp.com) をご覧ください。

Note サイプレスまたはサブライセンスを持つ関連企業から I²C コンポーネントをご購入いただく場合は、Philips I²C 特許権を譲渡し、これらのコンポーネントを Philips によって定義されている I²C 標準仕様準拠の I²C システムで使用してください。

配置

I2CHW ユーザ モジュールを使用すると、SCL と SDA の 2 つの選択肢を持つことができます。P1[7] 上の SCL と P1[5] 上の SDA です。その他の選択肢は、P1[1] 上の SCL と P1[0] 上の SDA です。P1[0]/P1[1] は、ISSP および ECO と共有されるため、可能な限り、P1[5]/P1[7] ピンを使用してください。配置に関して制限はありません。I²C モジュールは、専用の PSoC リソース ブロックと割り込みを使用するため、I²C モジュールを複数配置することはできません。

パラメータおよびリソース

すべてのバッファは、I²C マスタによる使用に関する名前が付けられます。たとえば、「Read」という名前のバッファは、I²C マスタによる書き込みを行います。

Slave_Addr

Slave_Addr は、スレーブをアドレスする I²C マスタによって使用される、7-bit スレーブアドレスを選択します。有効な選択肢は、0-127 (10 進数) からです。このパラメータは、拡張スレーブでは使用されません。

I2C_Clock

I²C インターフェースが実行される望ましいクロック速度を指定します。可能なクロック速度は、以下の 3 つです。

- 50K 標準

- 100K 標準
- 400K 高速

I2C_Pin

I²C 信号で使用するため、ポート 1 からピンを選択します。PSoC Designer がこれを自動的に実行するため、これらのピンで適切な駆動モードを選択する必要はありません。

Read_Buffer_Types

データ読み取りでサポートされているバッファのタイプを選択します。以下の 2 つのオプションを使用できます。RAM ONLY または RAM OR FLASH。RAM ONLY を選択すると、直接的な Flash-ROM 読み取りをサポートするのに必要なコードと変数が削除されます。RAM OR FLASH を選択すると、マスタに送信されるデータに関して、RAM バッファまたは Flash-ROM バッファの読み取りをサポートするコートと変数が提供されます。RAM OF FLASH を選択する場合は、バッファのタイプを選択するため、I2CHW_InitRamRead() または I2CHW_InitFlashRead() API を使用してください。このパラメータは、拡張スレーブでは使用されません。

Communication_Service_Type

このパラメータを使用すると、割り込みベースのデータ処理方式とポーリング方式のいずれかを選択できます。このパラメータは、拡張スレーブでは使用されません。割り込みベースの方式では、事前に定義したバッファに対して転送が開始されます。それからバックグラウンドで、可能な限り迅速にデータがバッファに入力されたりバッファから出力されたりします。データの移動を処理する ISR が含まれます。ポーリング方式を選択すると、データの移動を制御できるようになります。ポーリング方式を実装するには、関数 I2CHW_Poll() を定期的に呼び出します (正確なインスタンス名については I2CHWslave.h ファイルを参照してください)。ポーリング関数を呼び出すたびに 1 つのバイトが送信されます。他の I²C 関数は同じように使用されます。割り込みレイテンシーが重要な場合 (および同期通信の割り込みが原因で問題が発生する場合) は、ポーリング方式を使用してください 。ポーリング方式は、データをいつ転送するかを確実に制御する必要があるときにも使用できます。ポーリングの欠点は、I²C 状態のデバイスがが有効になっていると、ポーリング関数が呼び出されるまで、個々のバイト後バスが自動的にストールすることです。

アプリケーション プログラミング インタフェース (API)

アプリケーションプログラミングインターフェイス (API) のファームウェアは、複数のバイトの伝送記号を送受信するための作業をサポートする高レベルの命令を提供します。RAM または Flash メモリからの読み取りバッファを生成します。書き込みバッファは RAM メモリでのみ設定されます。

Note すべてのユーザ モジュール API の場合と同じように、API 関数を呼び出すことで A と X レジスタの値が変更されます。呼び出し後これらの値が必要な場合は、呼び出す関数が、呼び出し前に A と X の値を維持する必要があります。「registers are volatile」(レジスタは揮発性である) ポリシーは、PSoC Designer のバージョン 1.0 より有効となっています。C コンパイラは、これらの要求条件を自動的に処理します。アセンブリ言語のプログラマは、コードがこのポリシーを遵守していることも確認する必要があります。一部のユーザ モジュール API 関数では A と X は変更されないこともありますが、将来も変更されないという保証はありません。

共通の関数

以下の関数は、ユーザ モジュールのスレーブと拡張スレーブ バージョンの両方に共通です。

I2CHW_Start

説明：

なし。インターフェースの一貫性を保つためにのみ提供されます。スレーブを開始するには I2CHW_EnableSlave()、割り込みを有効にするには I2CHW_EnableInt() を使用してください。

C プロトタイプ：

```
void  I2CHW_Start(void);
```

アセンブラ：

```
lcall  I2CHW_Start
```

パラメータ：

なし

戻り値：

なし

特殊作用：

この関数、または今後この関数の実装によって、A および X レジスタが変更される場合があります。これは大容量メモリ モデルで、すべての RAM ページ ポインタ レジスタについて同じです。必要に応じて、fastcall16 関数を呼び出す値を維持するのは、呼び出す関数が行います。

I2CHW_Stop

説明：

I^2C 割り込みを無効にすることで、I2CHW を無効にします。

C プロトタイプ：

```
void   I2CHW_Stop(void);
```

アセンブラ：

```
lcall  I2CHW_Stop
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。

I2CHW_EnableInt

説明：

開始状態を検出できるよう、I²C 割り込み機能を有効にします。マクロを使用して、グローバルな割り込みイネーブル関数を呼び出す必要があります。M8C_EnableGInt.

C プロトタイプ：

```
void   I2CHW_EnableInt(void);
```

アセンブラ：

```
lcall  I2CHW_EnableInt
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。

I2CHW_DisableInt

説明：

SDA 割り込みを無効にして、I²C スレーブを無効にします。I2CHW_Stop. を同じ動作を実行します。

C プロトタイプ：

```
void   I2CHW_DisableInt(void);
```

アセンブラ：

```
lcall I2CHW_DisableInt
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。

I2CHW_EnableSlave

説明：

I2C_CFG レジスタでイネーブル スレーブ ビットを設定することで、I2C HW ブロックの I2C スレーブ関数を開始します。

C プロトタイプ：

```
void   I2CHW_EnableSlave(void);
```

アセンブラ：

```
lcall  I2CHW_EnableSlave
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。

I2CHW_DisableSlave

説明：

I2C_CFG レジスタでイネーブル スレーブ ビットをクリアすることで、I2C スレーブ関数を無効にします。

C プロトタイプ：

```
void   I2CHW_DisableSlave(void);
```

アセンブラ：

```
lcall  I2CHW_DisableSlave
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。

スレーブ関数

これらの関数は、拡張スレーブでは利用できません。

I2CHW_Poll

説明：

Communication_ Service_Type パラメータが 「Polled」(ポーリング) に設定されている場合に使用します。この関数は、ユーザが制御する入力を I/O 処理ルーチンに提供します。Communication_Service_Type パラメータが 「Interrupt」(割り込み) に設定されている場合、この関数は何も実行しません。

注意事項：I2CHW_Poll の呼び出しにより、I2C バスがリリースされます。これで、スレーブのファームウェアが、以前のリクエストの処理を終了する前に、マスタがデータを読み出したり、書き込んだりできるようになります。

C プロトタイプ：

```
void   I2CHW_Poll(void);
```

アセンブラ：

```
lcall I2CHW_Poll
```

パラメータ：

なし

戻り値：

なし

特殊作用：

1 つの I^2C のイベントは、これらのルーチンが呼び出され、状態変数が更新されるたびに処理されます。イベントには、エラー状態、I/O バイト数、または特定の場合には停止条件のいずれかが含まれます。これらのルーチンを呼び出すと、以下の 3 つの結果が発生します。

- イベントが存在しない場合は、何も実行されません。
- 利用できる場合は、アドレスまたはデータ バイトの受信または送信が行われます。
- 外部マスタが書き込み動作を完了すると、停止イベントが処理されます。

書き込み動作の終了時に停止状態が処理されると、状態変数のみが更新されます。停止状態が処理されるときに I^2C バイトが保留中である場合は、I2CHW_Poll 関数を再度呼び出して、バイトを処理します。Communication_Service_Type を Interrupt (割り込み) に設定すると、I2CHW_Poll() 関数は何の影響も与えません。バスで起動 / 再起動の条件とアドレスが検出された場合は、I2CHW_Poll() 関数が呼び出されるまでバスがストールします。アドレスが NAK された場合は、別の開始 / 再起動およびアドレスが検出されるまで、続いて送信されがバイトが無視されます。それ以外の場合は、I2CHW_Poll() 関数が呼び出されるまで、各データ バイトで I^2C バスがストールします。

Communication_Service_Type が 「Polled」(ポーリング) に設定されている場合は、I^2C バスが I^2C ハードウェアによってストールされます。受信データの場合、SCL (クロック) 行を低 (Low) に保つことで、ACK/NAK が生成される前にバスがバイトの終わりでストールします。送信データの場合、ACK または NAK のビットが外部的に生成された直後にバスがストールします。

I2CHW_bReadI2CStatus

説明：

状態ビットを制御 / 状態レジスタに返します。

C プロトタイプ：

```
BYTE  I2CHW_bReadI2CStatus(void);
```

アセンブラ：

```
lcall  I2CHW_bReadI2CStatus ; Accumulator contains status
```

パラメータ：

なし

戻り値：

```
BYTE  I2CHW_RsrcStatus
```

表を参照してください。

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

定数	値	説明
I2CHW_RD_NOERR	01h	マスタによってデータが読み取られ、標準の ISR 終了が発生した。
I2CHW_RD_ オーバーフロー	02h	利用可能なデータより多くのデータ バイトが読み取られた。
I2CHW_RD_ 完了	04h	読み取りが開始され、完了した。
I2CHW_READFLASH	08h	Flash から次の読み取り操作が実行される。
I2CHW_WR_NOERR	10h	マスタによってデータが書き込まれた。
I2CHW_WR_OVERFLOW	20h	マスタが書き込みバッファでバイトを書き込み過ぎた。
I2CHW_WR_COMPLETE	40h	マスタの書き込みが新しいアドレスまたは停止信号により完了した。
I2CHW_ISR_ACTIVE	80h	I^2C ISR がまだ終了しておらず、アクティブである。

I2CHW_ClrRdStatus

説明：

I2CHW_RsrcStatus レジスタで読み取り状態ビットをクリアします。他のビットは影響を受けません。

C プロトタイプ：

```
void  I2CHW_ClrRdStatus (void);
```

アセンブラ：

```
lcall  I2CHW_ClrRdStatus
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。

I2CHW_ClrWrStatus

説明：

I2CHW_RsrcStatus レジスタで書き込み状態ビットをクリアします。他のビットは影響を受けません。

C プロトタイプ：

```
void  I2CHW_ClrWrStatus (void);
```

アセンブラ：

```
lcall  I2CHW_ClrWrStatus
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

I2CHW_InitWrite

説明：

データを保存するために使用するスレーブのデータバッファへのポインタを初期化して、同じバッファのカウントバイトの値を初期化します。カウントは、供給される最大バッファの長さに初期化されます。マスタ書き込みの次のインスタンスでは、この関数によって定義されたアドレスにデータが配置されます。

C プロトタイプ：

```
void   I2CHW_InitWrite(BYTE * pWriteBuf, BYTE bBufLen);
```

アセンブラ：

```
AREA    bss (RAM, REL)
abWriteBuf    blk 10h

AREA    text (ROM,REL)
    push X                          ; save registers
    push A
    add  SP, 3
    mov  X, SP
    dec  X                          ; X points at data SP points at next
                                    ; empty stack location
    mov  [X], abWriteBuf            ; place the buffer address
                                    ; (page 0)on the stack at [X]
    mov  [X-2], 10                  ; place the count at [x-2]
                                    ; don't care what [X-1] is
                                    ; the compiler would assign 0 as the
                                    ; MSB of the Ramtbl addr
    lcall  I2CHW_InitWrite
    add   SP, -3                    ; restore the stack
    pop   A                         ; restore registers
    pop   X
```

パラメータ：

pWriteBuf: RAM バッファの位置へのポインタ。

buf_len：書き込みバッファの長さ。

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

I2CHW_InitRamRead

説明：

マスタがデータを取得するための RAM データ バッファを初期化して、同じバッファのカウント バイトの値を初期化します。I2CHW_SlaveStatus フラグ I2CHW_READFLASH から 0 までクリアすると、RAM でそれまでに設定されたバッファ位置から次の読み取りが実行されます。

C プロトタイプ：

```
void   I2CHW_InitRamRead(BYTE * pReadBuf, BYTE bBufLen);
```

アセンブラ：

```
AREA bss (RAM,REL)
abReadBuf:    blk 10h

AREA text (ROM,REL)
    push  X                        ; save registers
    push  A
    add   SP, 3
    mov   X, SP
    dec   X                        ; X points at data SP points at next
                                   ; empty stack location
    mov   [X], abReadBuf           ; place the read buffer address
                                   ; (page0)on the stack at [X]
    mov   [X-2], 10                ; place the count at [x-2]
                                   ; don't care what [X-1] is
                                   ; the compiler would assign 0 as
                                   ; the MSB of the Ramtbl addr
    lcall  I2CHW_InitRamRead
    add   SP, -3                   ; back up the stack (subtract 3)
    pop   A                        ; restore registers
    pop   X
```

パラメータ：

_ReadBuf: RAM バッファの位置側のポインタ。bBufLen: 読み取りバッファの長さ。

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

I2CHW_InitFlashRead

説明：

データを取得するために、Flash のデータ バッファを初期化します。I2CHW_SlaveStatus フラグ I2CHW_READFLASH から 1 までクリアすると、Flash でそれまでに設定されたバッファの位置から次の読み取りが実行されます。

C プロトタイプ：

```
void   I2CHW_InitFlashRead(const BYTE * pFlashBuf, WORD wBufLen);
```

アセンブラ：

```
area table(ROM,ABS)
org 0x1015

abFlashBuf:

      db 0x11, 0x22, 0x33, 0x44, 0x55, 0x66, 0x77, 0x88
      db 0x99, 0xaa, 0xbb, 0xcc, 0xdd, 0xee, 0xff


area text(ROM,REL)


    push  X                          ; save registers
    push  A
    add   SP, 4
    mov   X, SP
    dec   X                          ; X points at data SP points at next
                                     ; empty stack location
    mov   [X], <abFlashBuf           ; place the LSB of rom
                                     ; address on the stack at [X]
    mov   [X-1], >abFlashBuf         ; place the MSB of the rom address
                                     ; at [x-1] variable
    mov   [X-2], 0x0F                ; place the LSB of length
                                     ; at [x-2]
    mov   [X-3], 0x00                ; place the MSB of length
                                     ; at [x-3]
    lcall  I2CHW_InitFlashRead
    add   SP, -4                     ; adjust the stack (subtract 4)
    pop   A                          ; restore registers
    pop   X
```

パラメータ：

pFlashBuf: Flash/ROM バッファの位置側のポインタ。wBufLen: バッファの長さ。

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

拡張スレーブ関数

以下の関数は、ユーザ モジュールの拡張スレーブ バージョンのみで利用できます。

I2CHW_StartBufferedSlave

説明：

I2C バッファ モードを有効にします。拡張スレーブ API は、32-byte ハードウェア バッファを使用して通信をサポートするため、後方互換性がなくなります。

C プロトタイプ：

```
void  I2CHW_StartBufferedSlave(void);
```

アセンブラ：

```
lcall  I2CHW_StartBufferedSlave
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

I2CHW_StopBufferedSlave

説明：

I2C バッファ モードを無効にします。これにより、拡張スレーブと I^2CHW スレーブ API の互換性がサポートされます。

C プロトタイプ：

```
void  I2CHW_StopBufferedSlave (void);
```

アセンブラ：

```
lcall  I2CHW_StopBufferedSlave
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

I2CHW_EnableNoBCInt

説明：

バイト完了割り込みをオフにします。この関数が呼び出される前に I2CHW_StartBufferedSlave() と I2CHW_SetHWAddr() を呼び出す必要があります。

C プロトタイプ：

```
void  I2CHW_EnableNoBCInt(void);
```

アセンブラ：

```
lcall  I2CHW_EnableNoBCInt
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

I2CHW_DisableNoBCInt

説明：

バイト完了割り込みをオンにします。この関数が呼び出される前に I2CHW_StartBufferedSlave() と I2CHW_SetHWAddr() を呼び出す必要があります。

C プロトタイプ：

```
void  I2CHW_DisableNoBCInt (void);
```

アセンブラ：

```
lcall  I2CHW_DisableNoBCInt
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

I2CHW_EnableStopIE

説明：

停止割り込みをオンにします。この関数が呼び出される前に I2CHW_StartBufferedSlave() と I2CHW_SetHWAddr() を呼び出す必要があります。

C プロトタイプ：

```
void  I2CHW_EnableStopIE(void);
```

アセンブラ：

```
lcall  I2CHW_EnableStopIE
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

I2CHW_DisableStopIE

説明：

停止割り込みをオフにします。この関数が呼び出される前に I2CHW_StartBufferedSlave() と I2CHW_SetHWAddr() を呼び出す必要があります。

C プロトタイプ：

```
void  I2CHW_DisableStopIE (void);
```

アセンブラ：

```
lcall  I2CHW_DisableStopIE
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

I2CHW_ GetMasterWriteData

説明：

I²C ベースと現在のポインタ アドレス (I2C_BP および I2C_CP レジスタ) を取得することで書き込まれた最新データを取得します。バッファ カウントは、CPU ベースと現在のポインタを使用して決定されます。データが取得され、指定されたバッファに配置されます。

C プロトタイプ：

```
void   I2CHW_GetMasterWriteData(BYTE * pbReadMasterBuf);
```

アセンブラ：

```
   mov  A,> pbReadMasterBuf ; Load MSB part of pointer to RAM based null
                            ; terminated string.
   mov  X,< pbReadMasterBuf ; Load LSB part of pointer to RAM based null
                            ; terminated string.
   lcall I2CHW_GetMasterWriteData ; lcall function to send string out TX8 port
```

パラメータ：

```
_pbReadMasterBuf: Pointer to a RAM buffer location.
```

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

I2CHW _SetHWAddr

説明：

HW スレーブのアドレスを設定します。

C プロトタイプ：

```
void   I2CHW SetHWAddr (BYTE bHWAddr);
```

アセンブラ：

```
mov  A, 0x07         ; Load address of 0x07
lcall I2CHW_SetHWAddr ; The address will be set to 0x07
```

パラメータ：

_ bHWAddr：設定される HW スレーブのアドレス。(0 - 7Fh)

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

I2CHW _bMonitorBufAccess

説明：

I2CHW_XSTAT レジスタの SLAVE_BUSY ビットのステータスを返します。これは、スレーブがビジー状態になる場合とバッファ モジュールがアクセスされる場合を決定します。

C プロトタイプ：

```
BYTE   I2CHW bMonitorBufAccess (void);
```

アセンブラ：

```
lcall I2CHW bMonitorBufAccess ;lcall function to get status of buffer module
                              ;Status is returned in the Accumulator
```

パラメータ：

なし

戻り値：

_ status：SLAVE_BUSY ビットのステータス

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

I2CHW _WriteI2CBuf

説明：

この関数は、**pbWriteBuf** によって指定された SRAM バッファの bWriteCount バイトを **bStartAddr** で始まる 32-byte I2C バッファにコピーします。

C プロトタイプ：

```
void   I2CHW_WriteI2CBuf(BYTE bStartAddr, BYTE * pbWriteBuf, BYTE bWriteCount);
```

アセンブラ：

```
mov A, 32          ;Load buffer count
push A
mov A, >pbWriteBuf ;Load MSB part of pointer to RAM array
push A
mov A, <pbWriteBuf ;Load LSB part of pointer to RAM array
push A
mov A, 0           ;Load write start address (0-1Fh)
push A
lcall I2CHW_WriteI2CBuf
```

パラメータ：

_ bStartAddr：CPU_BP アドレスの始まり (ベース ポインタ)。

_ pbWriteBuf：書き込まれるデータの RAM バッファ位置へのポインタ。

_ bWriteCount：書き込むバイト数。

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

I2CHW _ReadI2CBuf

説明：

この関数は、**pbReadBuf** によって指定された SRAM バッファの bReadCount バイトを **bStartAddr** で始まる 32-byte I2C バッファにコピーします。

C プロトタイプ：

```
void   I2CHW ReadI2CBuf (BYTE bStartAddr, BYTE * pbReadBuf, BYTE bReadCount);
```

アセンブラ：

```
mov A, 32          ;Load # bytes to read
push A
mov A, >pbReadBuf ;Load MSB part of pointer to RAM array
push A
mov A, <pbReadBuf ;Load LSB part of pointer to RAM array
push A
mov A, 0           ;Load read start address (0-1Fh)
push A
lcall I2CHW_ReadI2CBuf
```

パラメータ：

_ bStartAddr：CPU_BP アドレスの始まり (ベース ポインタ)。

_ pbReadBuf：読み取られるデータを保存する RAM バッファ位置へのポインタ。

_ bReadCount：読み取るバイト数。

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

I2CHW _ wGetI2CPointers

説明：

I^2C ベース ポインタ (I2C_BP) と現在のポインタ (I2C_CP) レジスタの値を読み取ります。I2C_BP は、返される単語の MSB に含まれます。I2C_CP は、返される単語の LSB に含まれます。

C プロトタイプ：

```
WORD   I2CHW wGetI2CPointers (void);
```

アセンブラ：

```
lcall I2CHW wGetI2CPointers ;Returns LSB in A and MSB in X
; X contains I2C_BP and A contains I2C_CP
```

パラメータ：

なし

戻り値：

WORD wI2Cptrs (X contains I2C_BP および A contains I2C_CP)

特殊作用：

API セクションの冒頭にある注意事項を参照してください。現時点では、CUR_PP ページ ポインタ レジスタだけが変更されています。

I2CHW _Sleep

説明：

I2C 動作のライト スリープを管理します。この API 関数は、CY8CTMA3xx、CY8CTMG3xx、CY8CTST3xx デバイスのみで利用できます。

C プロトタイプ：

```
void   I2CHW_Sleep (void);
```

アセンブラ：

```
call I2CHW_Sleep
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項を参照してください。

ファームウェア ソースコードの例

I2CHW ユーザ モジュールの実装例を示します。

```
/*-----------------------------------------------------------*/
/* This sample code echoes bytes received from a master */
/* The master sends and requests up to 64 bytes.         */
/*                                                         */
/* The instance name of the I2CHWs User Module is defined */
/* as I2CHW.                                              */
/*                                                         */
/*  NOTE I2CHW does not depend upon the CPU clock         */
/*-----------------------------------------------------------*/
#include <m8c.h>
#include <i2CHWCommon.h>

/* setup a 64 byte buffer */
BYTE    abBuffer[64];
BYTE    status;

void EchoData()
{
/* Start the slave and wait for the master */
    I2CHW_Start();
    I2CHW_EnableSlave();
/* Enable the global and local interrupts */
```

```
    M8C_EnableGInt;
    I2CHW_EnableInt();

    /* Setup the Read and Write Buffer - set to the same buffer */
    I2CHW_InitRamRead(abBuffer,64);
    I2CHW_InitWrite(abBuffer,64);


    /* Echo forever */
    while( 1 )
    {
        status = I2CHW_bReadI2CStatus();
        /* Wait to read data from the master */
        if( status & I2CHW_WR_COMPLETE )
        {
        /* Data received - clear the Write status */
        I2CHW_ClrWrStatus();
        /* Reset the pointer for the next read data */
        I2CHW_InitWrite(abBuffer,64);
        }
        /* wait until data is echoed */
        /* want to know if RD_NOERR is SET AND RD_COMPLETE is SET */
        if( status & I2CHW_RD_COMPLETE )
        {
        /* Data echoed - clear the read status */
        I2CHW_ClrRdStatus();
        /* Reset the pointer for the next data to echo */
        I2CHW_InitRamRead(abBuffer,64);
        }
    }
}
void main(void)
{
    EchoData();
}
```

アセンブリ コードで記述されたスレーブとして構成される I2CHW ユーザ モジュールの実装例を示します。

```
;-------------------------------------------------------------
; this assembly assumes small memory model
;-------------------------------------------------------------
include "m8c.inc"
include "I2CHW_1Common.inc"
export rec_cnt
export i2c_addr
;export master_state

BUFFERSIZE: equ 64

export abBuffer
export rec_cnt
export status
```

```
area bss(RAM)

rec_cnt: blk 1
i2c_addr: blk 1
abBuffer: blk BUFFERSIZE
status:   blk 1
;master_state: blk 1


area text(ROM, REL)

export _main

_main:
    ;enable the I2C slave as an ISR based process

    lcall I2CHW_1_EnableSlave
call I2CHW_1_EnableInt
M8C_EnableGInt

Init:
mov A, BUFFERSIZE
push A
push A
mov A, <abBuffer
push A
mov X, sp
dec X
call I2CHW_1_InitWrite
;
;
;everything should still be initialized on the stack to call initRamRead
call I2CHW_1_InitRamRead
add SP, -3

checkI2CStatus:
    call I2CHW_1_bReadI2CStatus
    and A, I2CHW_WR_NOERR
    jnz writeHappened
    and A, I2CHW_RD_NOERR
    jnz readHappened

    jmp checkI2CStatus



readHappened:
    ;call I2CHW_3_ClrRdStatus
    ;calculate bytes read
    mov A, BUFFERSIZE
    sub A, [I2CHW_1_Read_Count]
    inc A
    cmp A, BUFFERSIZE
    jnz checkI2CStatus
    jmp _main
```

```
writeHappened:
    ;call I2CHW_3_ClrWrStatus
    ;calculate bytes read
    mov A, BUFFERSIZE
    sub A, [I2CHW_1_Write_Count]
    inc A
    cmp A, BUFFERSIZE
    jnz checkI2CStatus
    jmp Init


;end _main
```

アセンブリ コードで記述されたマスタとして構成される I2CHW ユーザ モジュールの実装例を示します。

```
;-----------------------------------------------------------------------------
; Assembly main line
;-----------------------------------------------------------------------------

include "m8c.inc"       ; part specific constants and macros
include "memory.inc"    ; Constants & macros for SMM/LMM and Compiler
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export _main

area bss(RAM)
sTXData::   blk 10
sRXData::   blk 10


area text(ROM,REL)

_main:


     mov [sTXData+0], 00h
     mov [sTXData+1], 01h
     mov [sTXData+2], 02h
     mov [sTXData+3], 03h
     mov [sTXData+4], 04h
     mov [sTXData+5], 05h
     mov [sTXData+6], 06h
     mov [sTXData+7], 07h
     mov [sTXData+8], 08h
     mov [sTXData+9], 09h



     M8C_EnableGInt;Enable Global Interrupts
     call I2CHW_1_EnableInt;Enable I2C Interrupts
     call I2CHW_1_Start;Start the I2C Block
```

```
     call I2CHW_1_EnableMstr;Enable Master Mode for I2C

   .loop:

         ; Load data to send to slave at address 0x68
         mov A,I2CHW_1_CompleteXfer
         push A
         mov A, 0x05;Send 9 Bytes
         push A
         mov A, >sTXData
         push A
         mov A, <sTXData
         push A
         mov A, 0x68;Slave Address
         push A
         call _I2CHW_1_bWriteBytes
         add sp, -5

         ;Wait for the TX to complete
    .TxCompleteLoop:
         call I2CHW_1_bReadI2CStatus;Check the I2C Status
         and A, I2CHW_WR_COMPLETE;Check to see if the write is complete
         jz .TxCompleteLoop;If not try again
         call I2CHW_1_ClrWrStatus;If complete clear status

         ;Get read to read data from slave at address 0x68
         mov A, I2CHW_1_CompleteXfer
         push A
         mov A, 0x05;Read 9 Bytes
         push A
         mov A, >sRXData
         push A
         mov A, <sRXData
         push A
         mov A, 0x68;Slave Address
         push A
         call _I2CHW_1_fReadBytes
         add sp, -5

         ;Wait for the RX to Complete
    .RxCompleteLoop:
         call I2CHW_1_bReadI2CStatus;Check I2C Status
         and A, I2CHW_RD_COMPLETE;Check to see if read is complete
         jz .RxCompleteLoop;If not try again
         call I2CHW_1_ClrRdStatus;If complete clear status

         jmp .loop ;Send and Receive Again


.terminate:
    jmp .terminate
```

C で記述された I2CHW ユーザ モジュールの拡張スレーブ実装例を示します。

```
/*************************************************************
* This sample code does a CPU write and read to and from
* the buffer.  Then waits for the buffer to get accessed
* again  and checks to see what was last written by the
* master.
*
* The instance name of the I2CHWs User Module is defined
* as I2CHW.
*************************************************************/

#include <m8c.h>        // part specific constants and macros
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

void main(void)
{
    BYTE abWriteBuf[] = "Hello World! 32byte string here";
    BYTE abReadBuf [32];
    I2CHW_EnableSlave();
    I2CHW_StartBufferedSlave(); //Enable buffered mode

    I2CHW_SetHWAddr(5); //Set slave address to 5

    while (I2CHW_bMonitorBufAccess());  //Wait until buffer is free
    I2CHW_WriteI2CBuf(0, abWriteBuf, 32); //write data

    while (I2CHW_bMonitorBufAccess());
    I2CHW_ReadI2CBuf(0, abReadBuf, 32);  //read data

    while (1)
    {
    //wait for Master to write data
if (I2CHW_bMonitorBufAccess())
{
            while (I2CHW_bMonitorBufAccess());
            //see if master wrote data
            I2CHW_GetMasterWriteData(abReadBuf);
}
    }
}
```

設定レジスタ

ここでは、I2CHW ユーザ モジュールにより使用または変更される PSoC リソースのレジスタについて説明します。

Table 1. リソース I2C_CFG：バンク 0 reg[D6] 構成レジスタ

ビット	7	6	5	4	3	2	1	0
値	予約済み	PinSelect	予約済み	IE 停止	クロック Rate[1:0]		予約済み	Enable (イネーブル)

ピンの選択：クリアすると、SDA は P1[5]、SCL は P1[7] になります。設定すると、SDA は P1[0]、SCL は P1[1] になります。

停止エラー割り込みの有効化：I^2C 停止条件で I^2C 割り込みを有効にします。

クロックレート [1,0]: 3 つの有効なクロックレート 50-、100-、400 kbps から選択します。

00b = 100 kHz

01b = 400 kHz

10b = 50k

11b = 予約済み

有効化：I^2C HW ブロックを有効にします。

Table 2. リソース I2C_SCR：バンク 0 reg[D7] 状態制御レジスタ

ビット	7	6	5	4	3	2	1	0
値	バス エラー	予約済み	停止状態	ACK 出力	アドレス	送信	最終 Recd ビット (LRB)	バイト完了

バス エラー：バス エラー状態の検出を示します。

停止状態：I^2C 停止状態の検出を示します。

ACK 出力：I^2C ブロックに受信バイトを承認 (1) 、または承認しない (0) ように指示します。.

アドレス：受信または送信されるバイトは、アドレスです。

送信：このビットは、それに続くバイト転送でシフタの方向を設定します。シフタは、I^2C バスからのデータをシフトしますが、「1」の書き込みにより、SDA 出力ラインを駆動して、シフタの出力を可能にします。このレジスタへの書き込みは、次の転送を初期化するため、このビットを書き込む前に、データをデータレジスタに書き込む必要があります。受信モード (0) では、この書き込みを実行する前に、データ レジスタからこれまでに受信したデータを読み取る必要があります。ファームウェアは、受信したスレーブ アドレスで RW ビットからこの方向を取得します。この方向制御は、データ転送でのみ有効です。アドレス バイトの方向は、ハードウェアによって決定されます。

最後に受信したビット (LRB) ； 送信シーケンスで最後に受信したビット (ビット 9) の値、送信先デバイスからの Ack/Nak の状態。

バイト完了：8 データビットが受信されました。受信モードでは、バスが Ack/Nak を待機するため、ストールします。送信モードでは、Ack Nak も受信され (LRB を参照)、バスがストールされて次の動作を待機します。

Table 3. リソース I2C_DR：バンク 0 reg[D8] データレジスタ

ビット	7	6	5	4	3	2	1	0
値	データ							

受信または送信されたデータ。データを送信するには、I2C_SCR レジスタに書き込む前に、このレジスタをロードする必要があります。受信したデータは、このレジスタから読み取られます。アドレスやデータが含まれる場合があります。

Table 4. リソース I2C_XCFG：バンク 0 reg[C8] I2C 拡張構成レジスタ

ビット	7	6	5	4	3	2	1	0
値	予約済み	予約済み	予約済み	予約済み	No BC Int	予約済み	Buffer Mode (バッファ モード)	HW Addr EN

No BC Int：互換モードでは、受信または送信したすべてのバイトが、バイト完了割り込みを生成します。

Buffer Mode (バッファ モード) : 拡張バッファ モジュールの動作モードを決定します。

HW Addr EN：ハードウェアのアドレス比較を有効にします。異なる構成については、以下の表を参照してください。

HW Addr EN	Buffer Mode (バッファ モード)	NoBCInt	バイト完了割り込み	クロック (SCL) ストール
オン	オン	オン	割り込みなし	ストールなし
		オフ	各バイトで割り込みを生成。	
オン	オフ	オン	各バイトで割り込みを生成。[a]	各バイトで SCL がストール。
		オフ	各バイトで割り込みを生成。	
オフ	オン	オン	アドレス バイトでのみ生成。[b]	アドレス バイトでのみ SCL がストール。
		オフ	各バイトで割り込みを生成。	
オフ	オフ	オン	各バイトで割り込みを生成。[c]	各バイトで SCL がストール。
		オフ	各バイトで割り込みを生成。	

a. HWAddr が有効で、バッファ モードが無効の場合は、NoBCInt を有効にしても影響はありません。これは、アドレス バイトでも同じです。受信 / 送信ビットは CPU によって設定されるためです。送信動作の場合は、送信するバイトを I2C_Data レジスタにロードする必要があります。

b. この構成では、NoBCInt を有効にすると、アドレス バイトのみが影響を受けます。これは、CPU が、I2C_SCR レジスタの ACK ビットを ACK/NACK アドレス バイトに書き込む必要があるためです。

c. 互換モードでは、NoBCInt を有効にしても何の影響も与えません。

Table 5. リソース I2C_XSTAT：バンク 0 reg[C9] I2C 拡張状態レジスタ

ビット	7	6	5	4	3	2	1	0
値	予約済み	予約済み	予約済み	予約済み	予約済み	予約済み	Dir	Slave Busy (スレーブ ビジー)

Dir：現在のバッファ送信の方向を示します。「1」- マスタの読み取りを示します。「0」- マスタの書き込みを示します。Slave Busy ビットが「1」に設定されている場合にのみ有効です。

Slave Busy (スレーブ ビジー)：これは、ハードウェア比較で設定され、それに続く停止信号でリセットされます。このビットをポーリングし、スレーブがビジー状態になる場合とバッファ モジュールがアクセスされる場合を決定します。

Table 6. リソース I2C_ADDR：バンク 0 reg[CA] I2C アドレス レジスタ

ビット	7	6	5	4	3	2	1	0
値	予約済み	Slave Address (スレーブ アドレス)						

これらの 7 ビットは、スレーブ自体のデバイス アドレスを保持します。

Table 7. リソース I2C_BP：バンク 0 reg[CB] I2C ベース アドレス ポインタ レジスタ

ビット	7	6	5	4	3	2	1	0
値	予約済み	予約済み	予約済み	I^2C ベース ポインタ				

このレジスタの値は、各 I^2C 書き込み処理の始まりでのみ変更されます。I^2C マスタは、特定の書き込み処理スレーブでのアドレスの後に、データの第 1 バイトでこのレジスタの値を供給する必要があります。読み取りでは、マスタはこのレジスタの値を設定する必要がありません。このレジスタの現在の値は、読み取りから直接使用されます。

Table 8. リソース I2C_CP：バンク 0 reg[CC] I2C 現在のアドレス ポインタ レジスタ

ビット	7	6	5	4	3	2	1	0
値	予約済み	予約済み	予約済み	I^2C の現在のポインタ				

このレジスタは、I2C_BP レジスタが設定するのと同じ値で同時に設定されます。現在の I^2C 処理のデータ バイトが完了するたびに、このレジスタの値が 1 ずつ増分されます。このレジスタの値は、データの読み取りと書き込みが行われる位置を常に決定します。

Table 9. リソース CPU_BP：バンク 0 reg[CD] CPU ベース アドレス ポインタ レジスタ

ビット	7	6	5	4	3	2	1	0
値	予約済み	予約済み	予約済み	CPU ベース ポインタ				

このレジスタ値は、CPU によって書き込まれる IO で完全に制御されます。このレジスタが書き込まれると、現在のアドレス ポインタ CPU_CP も同じ値で更新されます。最初の読み取りや書き込み、または I2C_ BUF レジスタとの読み取りや書き込みは、このアドレスで開始されます。ファームウェアは、

スレーブ デバイスが有効な値を持っている、または上書きされる前にデータが読み取られたことを確認します。

Table 10. リソース CPU_CP：バンク 0 reg[CE] CPU 現在のアドレス ポインタ レジスタ

ビット	7	6	5	4	3	2	1	0
値	予約済み	予約済み	予約済み	CPU の現在のポインタ				

このレジスタは、CPU_BP レジスタと同じ値で同時に設定されます。I2C_BUF レジスタが書き込みまたは読み取られると、CPU_CP が自動的に増分されます。

Table 11. リソース I2C_BUF：バンク 0 reg[CF] I2C データ バッファ レジスタ

ビット	7	6	5	4	3	2	1	0
値	データ バッファ							

I^2C データ バッファ レジスタ (I2C_BUF) は、データ バッファへの CPU 読み取りおよび書き込みインターフェースです。このレジスタが読み取られると、CPU の現在のポインタ (CPU_CP) でポイントされている位置のデータが返されます。同様に、このレジスタが書き込まれると、データがバッファに転送され、CPU の現在のポインタ (CPU_CP). でポイントされている位置に書き込まれます。このレジスタが読み取られる場合は常に、I^2C または CPU インターフェースのいずれでも RAM の内容は初期化されないため、有効な値は返されません。

バージョン ヒストリー

バージョン	考案者	説明
1.1	DHA	他のユーザ モジュールで設定された該当ピンの駆動モードを破損させるため、「I2C ピン」パラメータのデフォルト ピン値を削除しました。 起動の機能が次のように変更されました。 1. UM ピンのオープンドレイン低駆動モードが HI-Z アナログに変更されました。 2. I^2C クロックが有効になりました。 3. 遅延 5 nop の指示が追加されました。 4. 初期の ^2C ピン駆動モードを復元しました。 シャドー レジスタを通して出力ピンを更新する機能が追加されました。

Note PSoC Designer 5.1 ではすべてのユーザ モジュール データシートにおいてバージョン ヒストリーを導入し、ユーザー モジュールの過去のバージョンと現在のバージョンとの違いに関して高度な解説を掲載しています。