I²C ハードウェアブロックのデータシート I2CHW V 1.5

Copyright © 2003-2011 Cypress Semiconductor Corporation. All Rights Reserved.

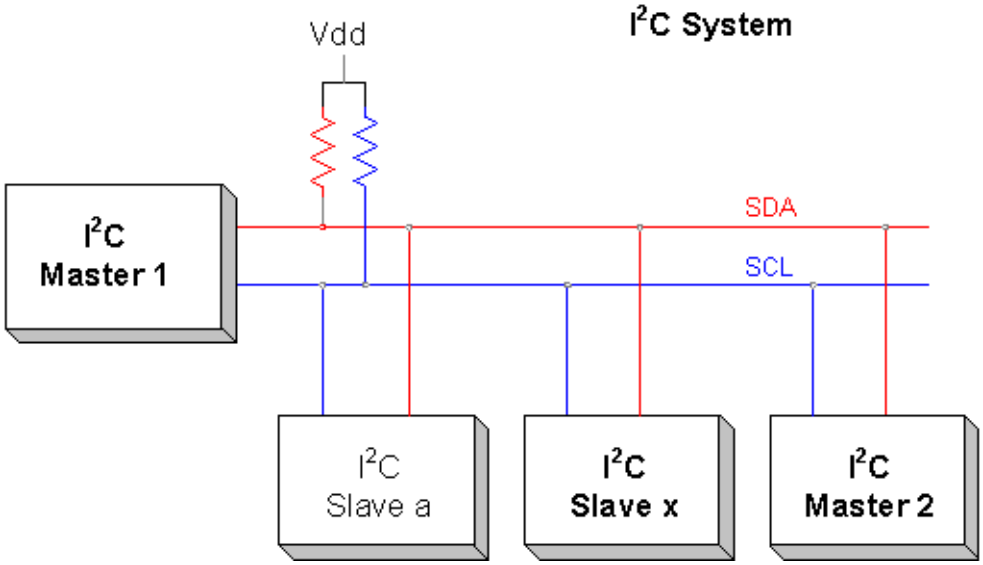
リソース	PSoC® Block			API メモリ (バイト)		ピン (外部 I/O)
	CapSense®	I²C/SPI	タイマー	フラッシュ	RAM	
CY8C20x34, CY8C20x24 (スレーブ専用)		X		279-440	8	

特性および概要

- 業界標準の Philips I²C バスコンパチブル • インターフェイス。
- スレーブ専用の作業、マルチマスターが有効。
- I²C バスでインタフェースするためには、2 つのピン (SDA と SCL) のみが必要です。
- 尚、100/400 kbits/s の標準スピードおよび 50 kbits/s をサポートします。
- 高いレベルの API には、最小のユーザープログラムが必要です。
- 7-bit アドレッシングモード、10-bit アドレッシングがサポートされています。

I2C ハードウェアユーザーモジュールは、ファームウェアで I²C スレーブデバイスを実装します。I²C バスは、Philips® が開発した 2 線式ハードウェアインタフェースの業界標準です。マスターは、I²C バスですべての通信を開始しますし、またすべてのスレーブデバイスのためクロックを提供します。I2CHW ユーザーモジュールは、400kbits/s までの標準モードをサポートします。このモジュールは、デジタルあるいはアナログユーザーブロックを使用しません。I2CHW ユーザーモジュールは同じバスで他のスレーブデバイスと互換しています。

Figure 1. I²C ブロック図



機能説明

このユーザーモジュールは、I²C ハードウェアリソースに対するサポート機能を提供します。CPU クロックが 12 MHz で駆動するように構成する場合、50/ 100/400 kbits/s でデータを転送することができます。より低い速度の CPU クロックを使用することもできますが、これを行う場合は、アドレスまたはデータ処理中に、バスストーリングが多少発生することがあります。I²C 仕様は、マスターが 100 kHz のクロックレートから DC まで駆動できるようにします。ハードウェアリソースへの直接的なアクセスを提供している SDA と SCL のため別の 2 つの選択肢があります。7 - ビットのアドレスモードは、指定された API でサポートされますが、10-bit アドレシングは API セットのためユーザー拡張機能でサポートします。

I²C のリソースは、バイトレベルのデータ転送機能をサポートします。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るか、専用の割り込みが発生します。ステータスレポートおよび割り込み信号の発生は、ハードウェアから取得し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。バイト完了、バスエラーを検出し、アービトレーション・ロスが発生するように割り込み信号を構成します。

すべての I²C 処理は、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。このユーザーモジュールに使用されている I²C リソースは、I²C スレーブとしてのみ動作することができます。このユーザーモジュールはバッファされた転送の構造に基づいて割り込み信号を提供します。最優先呼び出し機能からの通信が開始されます。メッセージの各バイトが終了した時点で、割り込み信号が生成され、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) が通信が継続させるためバスに対して適切な措置をしますが、ユーザーが実行する初期化機能によって異なります。アドレスを承認しないスレーブデバイスは、次のアドレスを受信するまで、再割り込みされていません。スレーブデバイスは、次のアドレスを受信するまで、承認または拒否して、それぞれのアドレスに応答する必要があります。

マスターとスレーブの違いを無視すれば、2 個の一般的な場合、受信機とトランスミッタが存在します。I²C 受信機は、入力データの 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の入力バイトを承認 (ack) するか承認しないか （ nak ）判断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 レジスタに書き込んで、I²C リソースに ack/nak 状態を通知します。バスのストールを解除して ack/nak 状態をバス上に配置し、次のデータバイトを変換することで、I2C_SCR レジスタの書き込み速度がバス内のデータの流入速度と一致されます。2 番目の送信機の場合、外部の受信デバイスは、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断するために I2C_SCR 読み取ることができます。送受信機の場合、データは I2C_DR レジスタにロードされ、I2C_SCR レジスタは、次の転送の部分をトリガするため再構築されます。

バッファの読み取りおよび書き込みルーチン (bWrite(...), bWriteBytes(...), fWriteBytes(...)) を使用する場合、特定のバッファの初期化機能 (InitWrite(...), InitRamRead(...), InitFlashRead(...)) を使用する必要はありません。これらの機能がバッファの読み取り、または書き込み (bWrite バイト (...), bWriteC バイト (...), fRead バイト (...)) 作業を開始する機能の呼び出しの一環として呼び出されます。

I²C バスについての詳細な説明と、リソースの実装方法は、インターネットで検索でき、PSoC Designer のデバイスのデータシートを参照してください。

設計上の考慮事項

I²C ソフトウェアの実装作業とは異なり、これらの実装作業は、内部的に生成されたクロックを使用して実行されます。主な発振器を、特定のクロック速度で実行します。データスループットは、プロセッサ速度の影響を受けますが、バイト単位のデータ転送機能は、指定された I²C 速度によって影響を受けます。

スレーブのデバイスは、マスター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、UMNAME_Read_Count と UMNAME_Write_Count. に名前が指定されます。変数は、適切な 'extern' 宣言文を含めることにより、ユーザーの C またはアセンブリコードでは、全体的にアクセスされます。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、マスタが読み出したり、作成されたバイト数を決定します。スレーブ (専用) モジュールの場合、カウント変数の初期サイズは、関連、カウント変数の初期サイズは、UMNAME_InitWrite, UMNAME_InitRamRead, UMNAME_InitFlashRead を使用して設定します。マルチマススレーブ モジュールに使用するオプションスレーブのカウント値を設定する機能の呼び出しは UMNAME_InitSlaveWrite, UMNAME_InitSlaveRamRead または UMNAME_InitSlaveFlashRead.

バッファを使用して、I²C スレーブ内のデータを読み取り、書き込んでください。I²C スレーブを活性化させる前に、適切なバッファを初期化する必要があります。I²C マスターによって読み取りまたは、書き込み作業が開始された後、適切な状態のビットが 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含まれているか、読み取りバッファから抽出されたデータを操作します。ISR を入力した場合、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ません。バッファの読み取りおよび書き込み作業は、このような方法で処理されます。

1. I²C マスターがバッファに含まれているより多くのデータを読み取ろうとする場合、I²C マスター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。(I²C プロトコルは、マスターが読み取りを停止するように I²C スレーブの方法を指定しません。)
2. I²C マスターがデータを受信して、I²C スレーブで書き込んで、より多くの使用保存領域が存在するか判断するときや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。I²C マスターがデータの作成作業を継続して行う場合、スレーブがこれを継続して NAK します。最初の NAK が生成された場合 (データが最終的に使用可能な場所に保存される), データはそれ以上保存されません。
3. バッファがゼロの長さで指定された場合に、I²C スレーブで書き込んだデータが NAK されますが、保存はされません。フラッシュでデータを直接読み取ることができる機能を活性化させる場合は、RAM またはフレッシュバッファのいずれかを使用することができます。データ転送の ISR は、フラッシュ /ROM や RAM にある読み取りバッファを使用する場合は、提供された 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を勧めます。I2CHW リソースをデフォルトの設定の一部だけとして置いてください。動作条件に示されたように、I2CHW ブロックの操作を変更しますが、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 I²C デバイスへの副作用がもたされる可能性があります。

I²C アドレッシング

I²C 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の 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 I²C マスターによって使用されます。有効なセクションは、0-127(dec) になります。バイトの LSB には R/~W ビットが含まれています。このビットが 0 の場合にはアドレスが作成されますし、LSB が 1 である場合に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、ユーザーモジュールは、入力アドレスをとって、これを変換し、これを読み取り / 書き込みビットと結合させ、完全なアドレスバイトを設定します。

例

0x48 アドレスがパラメータとして渡されるか、またはスレーブのアドレスに指定されます。各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ています。I²C マスターは (8-bits) /0x90 バイトを送信して、スレーブでデータを作成し、バイト 0x91 を転送して、スレーブでデータを読み取ります。

スレーブモジュールが、アドレスのパラメータの数値を入力値に基づく 10 進数を収容するため、7-bit アドレスを 10 進数 (10 進数の 72) で入力してください。

外部電気接続

ブロック図に示されているように、I²C バスは外部のプル抵抗が必要です。プル抵抗 (R_P) は、電源電圧、クロック速度、およびバスの容量に応じて決定されます。特定のデバイス (マスターまたはスレーブ) の最小のシンク電流は出力電圧の場合、V_{OLmax} = 0.4V で 3 mA 以上が必要です。これは、5-volt システムの最小プルアップ抵抗値を約 1.5 kOhms 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寄ります。150 pF バスの容量を持つ 5-volt システムの場合、プルアップ抵抗が 6 kOhm 以内になるようにする必要があります。I²C バス仕様に関する詳細情報は、Philips 社のウェブサイト (www.philips.com) を参照してください。

Note Cypress または下請ライセンスの関係会社から I²C 部品を購入し、Philips I²C 特許権に基づくライセンスを提供し、これらの部品を I²C システムとして使用しますが、これらのシステムは、Philips 社が指定した I²C 規格の仕様に準拠する必要があります。

DC および AC 電気特性

デバイスのデータシートの DC および AC 電気特性のセクションに記載されているデバイスの特性データを参照してください。

配置

I2CHW ユーザーモジュールを介して SCL および SDA P1[5]/P1[7] または P1[0]/P1[1] 2 つを選択することができます。配置の制約はありません。I²C モジュールを複数回交換することは許可されていないが、これは I²C モジュールが高密度 PSoC リソースブロックと割り込みの信号を使用しているからです。

パラメータ、およびリソース

すべてのバッファの名前は、I²C マスターで使用方法を説明するために作成されます。例えば、I2Cs_pRead_Buf は I²C マスターによって読み取りのデータが含まれている RAM の場所を意味します。

Slave_Addr

Slave_addr は、7-bit スレーブアドレスを選択し、このアドレスは、スレーブをアドレスする I²C マスターによって使用されています。有効なセレクションは 0-127(dec) です。

I2C_Clock

I²C インタフェースバスが駆動される望ましいクロック速度を指定します。3 個の可能なクロックレートが存在します。

I2C_Clock 速度
50K 標準
100K 標準
400K 迅速

I2C_Pin

I²C 信号を使用するために、ポート 1 からピンを選択します。これらのピンのために、適切なモードを選択する必要はないが、これは PSoC Designer が、これらの作業を自動的に実行するためです。

Read_Buffer_Types

データを読み取るためにどのようなタイプのバッファがサポートされるか選択します。2 個のオプションを使用することができます。RAM ONLY または RAM OR FLASH. RAM ONLY を選択する場合、直接的な Flash-RO M読み込みをサポートするとき必要なコードと、変数が削除されます。RAM OR FLASH を選択する場合は、送信するデータの RAM バッファ、または Flash-ROM、バッファのいずれかを判別するために、コードと変数のサポート機能がマスターに提供されます。RAM OR FLASH を選択する場合、RAM- または Flash- 読み取りバッファを使用するか選択できるように API 呼び出しを使用してください。

Communication_Service_Type

ユーザーは、このパラメータを使用して割り込み信号ベースのデータ処理方法やポーリング方法のどちらかを選択することができます。割り込みベースの転送方式の場合は、既に指定されたバッファと、逆に開始されます。次に、データを可能な限り迅速に背景に移動するか、バッファの範囲を超えています。データ移動の機能を処理する ISR ルーチンが含まれています。ポーリングのデータ処理の方式を選択する場合、データの移動が発生するか制御することができます。ポーリング方式を実装するには、機能を定期的に呼び出してください。I2CHW_Poll()(正確な例名は slave.h ファイルを参照してください)。一つの例 I2CHW ポーリングを呼び出すたびに一つのバイトが送信されます。他の I²C 機能が同じように使用されます。割り込みが非常に重要な状態 (同期通信の割り込みが原因で問題が発生する可能性があります) で、ポーリング通信方式を使用してください。データ転送時間を確実に制御する必要があるときに再び利用することができます。ポーリングの欠点は、I²C 状態の機械が有効になっている場合、ポーリング機能が呼び出されるまで、個々のバイト後、バスは自動的にストールするということです。

割り込み生成の制御

次の場合には、次のパラデータのみアクセス可能です。**割り込みの生成を制御する機能が有効になった場合** PSoC Designer のチェックボックスをチェックした場合。これは、以下で使用可能です。**プロジェクト > 設定 > チップエディタ** .

IntDispatchMode

IntDispatchMode パラメータを使用して同じブロックですが、別のオーバーレインに存在する複数のモジュールによって共有される割り込み信号のため割り込み要求を処理する方法を指定します。**ActiveStatus** を選択する場合は、ファームウェアが共有される割り込み要求信号を使用する前に、どのようなオーバーレイが有効になっているかテストします。これらの試験は、共有された割り込み信号が要求されるたびに発生します。これは、レイテンシを追加し、非決定的な共有割り込み依頼を使用するプロシージャを書き込みますが、特定の RAM は必要ないです。」**OffsetPreCalc** を選択した場合、オーバーレイが最初にロードされると、ファームウェアが共有される割り込み要求信号に対するソースを計算します。これらの計算により、割り込みレイテンシが減少し、共有されている割り込み信号に対する決定的な手続きを生成しますが、RAM バイトが失われることになります。

アプリケーションプログラミングインターフェイス

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作業をサポートする高かい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。作成バッファは RAM メモリにのみ設定されます。

Note ここでは、API 機能を呼び出して、A および X レジスタの値を、すべてのユーザーのモジュールの API に変更することができます。呼び出された後、それらの値が必要な場合、呼び出しに前に A と X の値を保持する責任は、呼び出し機能のあるデバイスにあります。このような “registers are volatile(レジスタが揮発性である)” 政策が効率性のために使用され、PSoC Designer のバージョンが 1.0 であるため、このポリシーが有効になります。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、また、そのコードが、この方針を遵守しているか確認する必要があります。何人かのユーザーモジュールの API の機能で、A と X が変更されていない状態で残されている可能性があるにもかかわらず、今後もそのまま残されていない可能性もあります。

I2CHW_Start

説明：

なし。インタフェースの一貫性のためにのみ提供されます。

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

副作用：

今回またはこの機能を今後実装するときに A 、X レジスタを変更することができます。これは、大容量メモリモデル (CY8C29xxx) RAM ページポインタレジスタの場合には、事実です。必要によって、fastcall16 機能を呼び出すときに値を保持する責任は、呼び出し機能の持つデバイスにあります。

I2CHW_Stop

説明：

I^2C 割り込み信号を無効にして I^2CHW 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

副作用：

今回または、この機能を今後実装するときに A 、X レジスタを変更することができます。これは、大容量メモリモデル (CY8C29xxx) RAM ページポインタレジスタの場合には、事実です。必要によって、fastcall16 機能を呼び出すときに値を保持する責任は、呼び出し機能を持つデバイスにあります。

I2CHW_EnableInt

説明：

開始条件を検出することができる I^2C 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を有効にする機能を呼び出す必要があるということを覚えてください。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

副作用：

今回、またはこの機能を今後実装するときに A 、X レジスタを変更することができます。これは、大容量メモリモデル (CY8C29xxx) RAM ページポインタレジスタの場合には、事実です。必要によって、fastcall16 機能を呼び出すときに値を保持する責任は、呼び出し機能を持つデバイスにあります。

I2CHW_DisableInt

説明：

SDA 割り込み信号を無効にして、I^2C スレーブを無効ししてください。次のような同じ動作を実行してください。I2Cs_Stop.

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

今回、または、この機能を今後実装するとき A および X レジスタを変更することでできます。これは、大容量メモリモデル (CY8C29xxx) RAM ページポインタレジスタの場合は事実です。必要によって、fastcall16 機能を呼び出すときに値を保持する責任は、呼び出し機能を持っているデバイスにあります。

I2CHW_Poll

説明：

Communication_ Service_Type パラメータを “Polled? に設定する場合に使用。この機能は、ユーザーの制御入力の項目を I/O 処理ルーチンに提供しています。Communication_Service_Type パラメータが “Interrupt? に設定されている場合、機能のデバイスが何も動作しません。

注意事項：I2CHW_Poll を呼び出す場合は、I2C バスが解放され、それによりスレーブのファームウェアは、以前の要求を最終的に処理する前に、マスターがデータを読み出したり、書き込むことができます。

C プロトタイプ：

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

アセンブラ：

```
lcall I2CHW_Poll
```

パラメタ：

なし

戻り値：

なし

副作用：

1 つの I²C 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。一つのイベントは、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を設定します。これらのルーチンを呼び出すことにより、発生する可能性の 3 つの結果が存在します。

1. データが存在しない場合、動作が実行されない。
2. 1 つが使用可能な場合は、アドレスまたはデータバイトが受信または送信されます。
3. 外部マスターが作成作業を完了する場合、停止のイベントが処理される。

作成作業が終了される重要な時点で停止状態が処理される場合にのみ、状態変数が更新されます。停止状態が処理されるときに I²C バイトがストップされている場合は、機能を再起動して、それを処理してください I2CHW_Poll。I2CHW_Poll() 機能は Communication_Service_Type を割り込みに設定する場合、効果は発生しません。バスからの起動 / 再起動の条件とアドレスが検出された場合、機能が呼び出されるまではバスがストールします。I2CHW_Poll() アドレスが NAK された場合は、別の開始 / 再起動し、アドレスが検出されるまでの処理のため転送される従属的なバイトが無視され、それ以外の場合には、I²C バスは I2CHW_Poll() 機能が呼び出されるまで、それぞれのデータバイトを呼び出します。

I²C データは I²C ハードウェアによってストールされているためである Communication_Service_Type“Polled? に設定されている場合、この機能が呼び出されるまでです。受信したデータの場合、バイト終了し、SCL(クロック) ラインを下に固定して ACK/NAK を作成する前にバスがストールしています。転送されたデータの場合は、ACK/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_EnableSlave

説明：:

I²C HW ブロックのための I²C スレーブの機能を I2C_CFG レジスタで有効なスレーブのビットを設定することにより活性化されます。

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

副作用：

今回、またはこの機能を今後実装するときに A 、X レジスタを変更することができます。これは、大容量メモリモデル (CY8C29xxx) RAM ページポインタレジスタの場合には、事実です。必要によって、fastcall16 機能を呼び出すときに値を保持する責任は、呼び出し機能を持つデバイスにあります。

CY8C27xxxA のデータシートに記載されているように、I²C が 6MHz CPU クロック速度に設定レジスタを作成します。ここで実装されたルーチンは、これらの作業を実行します。構成レジスタの側に直接書き込まれている場合、作業が適切に行われているか、または I²C 通信障害が発生するか確認してください。

I2CHW_DisableSlave

説明：

I²C スレーブの機能は、I2C_CFG レジスタの有効にするスレーブのビットを削除して無効にします。

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

副作用：

今回、またはこの機能を今後実装するときに A 、X レジスタを変更することができます。これは、大容量メモリモデル (CY8C29xxx) RAM ページポインタレジスタの場合には、事実です。必要によって、fastcall16 機能を呼び出すときに値を保持する責任は、呼び出し機能を持つデバイスにあります。

CY8C27xxxA のデータシートに記載されているように、I²C 設定レジスタを 6MHz CPU クロック速度で作成します。ここで実装されたルーチンは、これらの作業を実行します。構成レジスタの側に直接書き込まれている場合、作業が適切に行われているか、または、I²C 通信障害が発生するかを確認してください。

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。

副作用：

今回、またはこの機能を今後実装するときに A 、X レジスタを変更することができます。これは、大容量メモリモデル (CY8C29xxx) RAM ページポインタレジスタの場合には、事実です。必要によって、fastcall16 機能を呼び出すときに値を保持する責任は、呼び出し機能を持つデバイスにあります。現在、CUR_PP ページポインタレジスタのみが変更されます。

定数	値	説明
I2CHW_RD_NOERR	01h	マスタによって決定されたデータは、一般的な ISR があります
I2CHW_RD_ オーバーフロー	02h	マスタが読み取ったより多くのデータバイトが使用されている。
I2CHW_RD_ 完了	04h	読み取り作業が開始され、完了される
I2CHW_READFLASH	08h	フラッシュの場所で、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正常に作成される。
I2CHW_WR_OVERFLOW	20h	マスタが作成バッファのために多すぎのバイトを作成する。
I2CHW_WR_COMPLETE	40h	マスタの作成作業が新しいアドレスまたは停止信号により完了
I2CHW_ISR_ACTIVE	80h	I^2C ISR がまだ存在しないが、有効

I2CHW_ ClrRdStatus

説明：

I2CHW_RsrcStatus レジスタからの読み取り状態のビットを削除します。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

回収値：

なし

副作用：

今回、またはこの機能を今後実装するときに A 、X レジスタを変更することができます。これは、大容量メモリモデル (CY8C29xxx) RAM ページポインタレジスタの場合には、事実です。必要によって , fastcall16 機能を呼び出すときに値を保持する責任は、呼び出し側にあります。現在、CUR_PP ページポインタレジスタのみが変更されます。

I2CHW_ClrWrStatus

説明：

I2CHW_RsrcStatus レジスタの作成状態のビットを削除します。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

回収値

なし

副作用：

今回、またはこの機能を今後実装するときに A 、X レジスタを変更することができます。これは、大容量メモリモデル (CY8C29xxx) RAM ページポインタレジスタの場合には、事実です。必要に応じて , fastcall16 機能を呼び出すときに値を保持する責任は、呼び出し側にあります。現在、CUR_PP ページポインタレジスタのみが変更されます。

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。マスタ作成の次の例では、データがこの機能デバイスによって指定されたアドレスに位置しています。

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 バッファの位置側のポインタ

buf_len: 作成バッファの長さ。

戻り値：

なし

副作用：

今回、またはこの機能を今後実装するときに A 、X レジスタを変更することができます。これは、大容量メモリモデル (CY8C29xxx) RAM ページポインタレジスタの場合には、事実です。必要に応じて , fastcall16 機能を呼び出すときに値を保持する責任は、呼び出し側にあります。現在、CUR_PP ページポインタレジスタのみが変更されます。

I2CHW_InitRamRead

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。I2CHW_SlaveStatus フラグ I2CHW_READFLASH を 0 までを削除すると、*以前に* RAM に設定されたバッファの位置から次の読み取り操作が試行されます。

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

副作用：

今回、またはこの機能を今後実装するときに A 、X レジスタを変更することができます。これは、大容量メモリモデル (CY8C29xxx) RAM ページポインタレジスタの場合には、事実です。必要に応じて , fastcall16 機能を呼び出すときに値を保持する責任は、呼び出し側にあります。現在、CUR_PP ページポインタレジスタのみが変更されます。

I2CHW_InitFlashRead

説明：

データの取得するために、フラッシュのデータバッファを初期化します。I2CHW_SlaveStatus フラグ I2CHW_READFLASH を 0 までを削除すると、*以前に* RAM に設定されたバッファの位置から次の読み取り操作が試行されます。

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

副作用：

今回、またはこの機能を今後実装するときに A 、X レジスタを変更することができます。これは、大容量メモリモデル (CY8C29xxx) RAM ページポインタレジスタの場合には、事実です。必要に応じて , fastcall16 機能を呼び出すときに値を保持する責任は、呼び出し側にあります。現在、CUR_PP ページポインタレジスタのみが変更されます。

サンプルファームウェアのソースコード

これは I2CHW スレーブのモジュールの実装作業の例です。

```
/*****************************************************************/
/* 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();
}
```

これは、アセンブリコードで記述されたスレーブ用に構成された I2CHW ユーザーのモジュールの実装作業の例です。

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

INT_MSK3:  equ 0xDE
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

    call 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
```

設定レジスタ

このセクションには、I2CHW ユーザーのモジュールを使用するか、または変更された PSoC リソースのレジスタの説明が記載されています。

Table 1. リソース I2C_CFG：バックの 0 reg[D6] 構成レジスタ

ビット	7	6	5	4	3	2	1	0
値	保存	PinSelect	保存	IE 停止	クロック Rate[1]	クロック Rate[0]	保存	有効にする

ピンの選択：P1[5]/P1[7] または P1[0]/P1[1] いずれかの SCL および SDA を選択します。

エラーの割り込みの停止を有効に I^2C 停止条件で I^2C 割り込みの機能を有効にします。

クロックレート [1,0]: 3 つの有効な値を選択します。クロックレート 50- 100- 400kbps.

有効にする I^2C HW ブロックを有効にする

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	保存	停止状態	ACK アウト	アドレス	伝送	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 停止状態の検出を表します。

ACK アウト：I^2C ブロックの受信バイトを承認 (1) 、または承認しないように (0) に指示します。

アドレス：受信または送信されたバイトは、アドレスです。

送信：このビットは、続くバイト転送のためのコンバータの方向を設定します。コンバータは、I^2C バスで、常にデータを変換しますが、SDA 出力ラインをドライブするために '1' を書き込みイネーブルする。次の転送機能を初期化するため、このビットを作成する前に、データをデータレジスタに作成する必要があります。受信モード (0) で、これらの作成作業を実行する前に、データレジスタから以前に受信したデータを読み込む必要があります。ファームウェアは、受信したスレーブアドレスの RW ビットでは、これらの方向を派生させます。これらの方向制御は、データ転送の場合にのみ有効です。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。

最終に受信されたビット (LRB): 送信シーケンスで、最終受信ビット (ビット 9) の値は、指定デバイスの Ack/Nak の状態です。

バイト完了：8 データビットが受信されました。受信モードの場合には、バスがストールして Ack/Nak を待機します。転送モードの場合、Ack Nak また、受信 (LRB を参照) され、バスがストールされ、次の動作のために待機しています。

Table 3. リソース I2C_DR：バック 0 の 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。データを転送するには、I2C_SCR レジスタを作成する前に、これらのレジスタを案内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。このレジスタは、アドレスやデータが含まれている場合があります。

バージョン履歴

バージョン	作成者	説明
1.5	DHA	起動の機能が次のように変更されました。 1. ユーザーモジュールのピン Initial Open-Drain Low 駆動モードでは、HI-Z アナログに変更されていました。 2. I^2C クロックを有効にする。 3. 遅延 5 (nop) ガイドラインが提供される。 4. 初期の I^2C ピンの駆動モードを回復します。

Note PSoC Designer 5.1 は、すべてのユーザーモジュールのデータシートバージョン履歴を紹介しています。このセクションでは、現在および以前のユーザーモジュールの違さに対する高いレベルの説明が記載されています。

Copyright © 2003-2011 Cypress Semiconductor Corporation. The information contained herein is subject to change without notice. Cypress Semiconductor Corporation assumes no responsibility for the use of any circuitry other than circuitry embodied in a Cypress product. Nor does it convey or imply any license under patent or other rights. Cypress products are not warranted nor intended to be used for medical, life support, life saving, critical control or safety applications, unless pursuant to an express written agreement with Cypress. Furthermore, Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress products in life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

PSoC Designer ™ and Programmable System-on-Chip ™ are trademarks and PSoC® is a registered trademark of Cypress Semiconductor Corp. All other trademarks or registered trademarks referenced herein are property of the respective corporations.

Any Source Code (software and/or firmware) is owned by Cypress Semiconductor Corporation (Cypress) and is protected by and subject to worldwide patent protection (United States and foreign), United States copyright laws and international treaty provisions. Cypress hereby grants to licensee a personal, non-exclusive, non-transferable license to copy, use, modify, create derivative works of, and compile the Cypress Source Code and derivative works for the sole purpose of creating custom software and or firmware in support of licensee product to be used only in conjunction with a Cypress integrated circuit as specified in the applicable agreement. Any reproduction, modification, translation, compilation, or representation of this Source Code except as specified above is prohibited without the express written permission of Cypress.

Disclaimer: CYPRESS MAKES NO WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, WITH REGARD TO THIS MATERIAL, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. Cypress reserves the right to make changes without further notice to the materials described herein. Cypress does not assume any liability arising out of the application or use of any product or circuit described herein. Cypress does not authorize its products for use as critical components in life-support systems where a malfunction or failure may reasonably be expected to result in significant injury to the user. The inclusion of Cypress' product in a life-support systems application implies that the manufacturer assumes all risk of such use and in doing so indemnifies Cypress against all charges.

Use may be limited by and subject to the applicable Cypress software license agreement.