Easy I²C スレーブのデータシート EzI2Cs V 1.2

Copyright © 2004-2010 Cypress Semiconductor Corporation. All Rights Reserved.

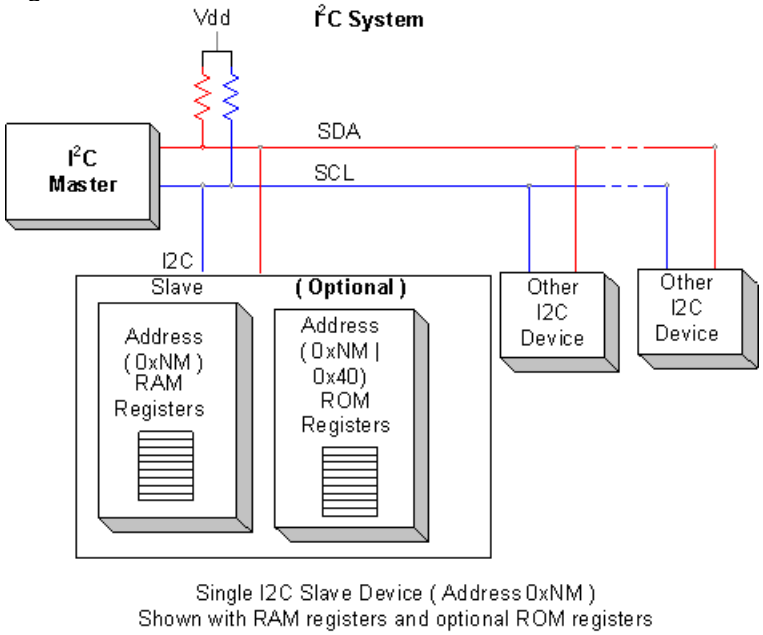
リソース	PSoC® ブロック			API メモリ（バイト）		ピン（外部 I/O あたり）
	デジタル	アナログ CT	アナログ SC	Flash	RAM	
CY8C29/27/24/22/21xxx、CY8C23x33、CY8CLED02/04/08/16、CY8CLED0xD、CY8CLED0xG、CY8CTST110、CY8CTMG110、CY8CTST120、CY8CTMG120、CY8CTMA120、CY8C21x45、CY8C22x45、CY8C28x45、CY8CPLC20、CY8CLED16P01、CY8C28xxx、CY8C21x12						
RAM のリード / ライト バッファ	0	0	0	315	6	2
ROM バッファ サポート用	0	0	0	439	6	2

特徴と概要

- 業界標準の フィリップス社 I²C バス互換インターフェイス
- 一般的な I²C EEPROM インターフェイスをエミュレート
- 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 kbps の標準データ レート
- ユーザ プログラミングを最小限に抑える高レベル API

EzI2C ユーザ モジュールは、I²C レジスタ ベースのスレーブ デバイスを実装します。I²C バスは、フィリップス社 ® によって開発された、業界標準の 2 配線ハードウェア インターフェイスです。マスタは、I²C バスの全通信を開始し、すべてのスレーブ デバイスへクロックを供給します。EzI2C ユーザ モジュールは、400 kbps の速度まで、標準モードをサポートします。このモジュールでは、デジタルまたはアナログ PSoC ブロックは消費しません。EzI2C ユーザ モジュールは、同じバスで複数デバイスと互換性を持ちます。

Figure 1. I²C 回路図



機能説明

EzI2C ユーザ モジュールは、PSoC Designer で提供されている I2CHW ユーザ モジュールとは異なるアプローチを取ります。このユーザ モジュールは、1 つまたは 2 つの I²C アドレスを持つ I²C スレーブ構成のみをサポートします。最初のアドレスは、常に RAM が割り当てられた領域で、オプションの 2 つ目のアドレスは ROM 領域にアクセスします。2 つ目のアドレスは、0x40 で論理和した RAM 領域アドレスです。たとえば、0x05 のアドレスを選択すると、RAM 領域アドレスは 0x05、オプションの 2 つ目のアドレスは 0x45 となります。両方のアドレスは、右揃えです。RAM および ROM 領域は、1 ー 255 バイトのデータ構造を持つことができます (バイト、整数、アレイ、構造)。

I²C ハードウェアが割り込み駆動であるため、このユーザ モジュールで、グローバルな割り込みを有効にする必要があります。このユーザ モジュールに割り込みが必要となる場合でも、ISR (割り込みサービス ルーチン) にコードを追加する必要はありません。モジュールは、コードから独立してすべての割り込み (データ転送) を担います。このインターフェイスに割り当てられたメモリ バッファは、アプリケーションと I²C マスタ間のシンプルなデュアル メモリのように見えます。

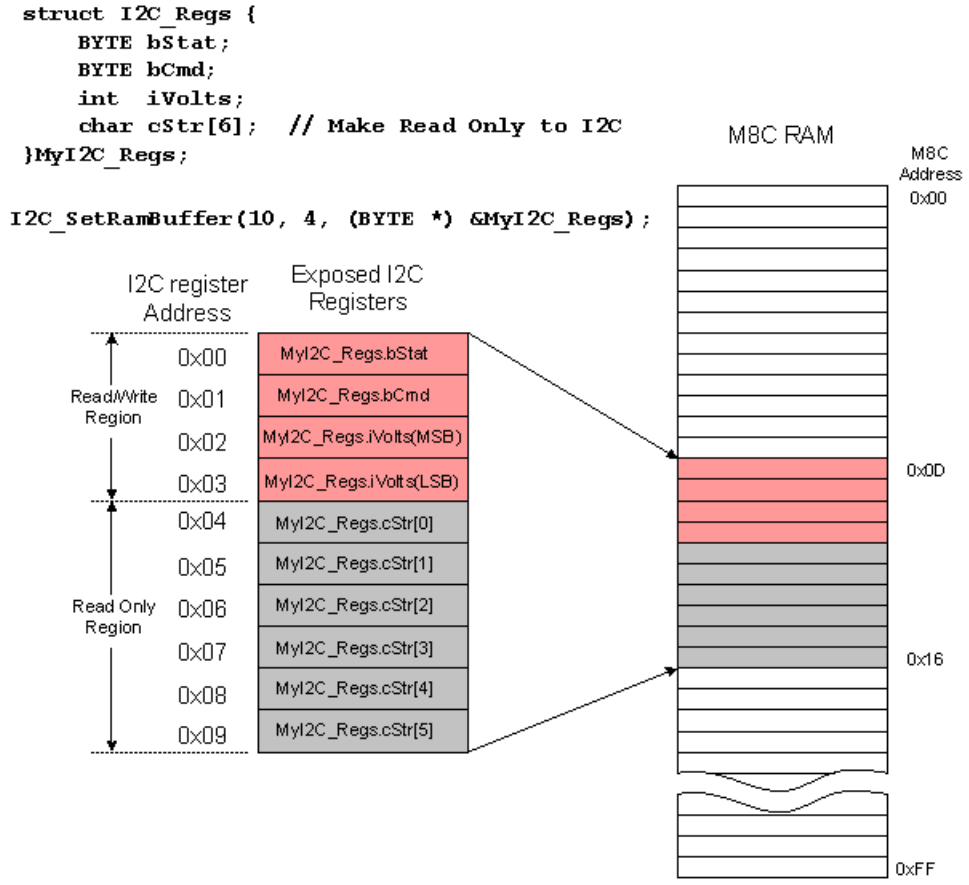
必要に応じ、データ構造でセマフォとコマンドの位置を定義して、マスタとこのスレーブ間でより高レベルのインターフェイスを作成できます。

RAM 領域インターフェイス

I²C マスタへのインターフェイスは、一般的な 256 バイトの I²C EEPROM へのインターフェイスとよく似ています。PSoC API は、シンプルな変数、アレイ、または構造として定義された RAM として扱われます。ある意味では、これは、プログラムと I²C バス上の I²C マスタ間の共有 RAM インターフェイスのように動作します。API を使うと、I²C マスタにデータ構造を公開することができます。ユーザ モジュールは、I²C マスタが RAM の特定の領域にアクセスすることだけを許可し、その領域外での読み取りや書き込みを防止します。I²C インターフェイスには、単一の変数、値のアレイ、構造といったデータを公開することができます。ここで必要なのは、初期化される際、変数またはデータ構造の始まりへのポインタのみです。以下の図を参照してください。

[+] Feedback

Figure 2. RAM 領域インターフェイス



たとえば、この構造を作成するとします。

```
struct I2C_Regs {   // Example I2C interface structure
    BYTE bStat;
    BYTE bCmd;
    int  iVolts;
    char cStr[6];  // Read only string
} MyI2C_Regs;
```

この構造には、メモリに内でポインタで参照できるさえすれば、任意の名前を持つ変数のグループを含めることができます。インターフェイス (I2C マスタ) は、バイトのアレイとしてのみこれを見るため、定義された領域外のメモリにはアクセスできません。前述の構造例を使うと、供給される API は、I2C インターフェイスにデータ構造を公開するために使用されます。最初のパラメータが I2C に公開する RAM のサイズを設定します (この例では構造全体)。2 つ目のパラメータは、読み取り / 書き込み領域でバイト数を設定することで、読み取り / 書き込み領域と読み取り専用領域の境界を設定します。読み取り / 書き込み領域が最初で、次に読み取り専用領域が続きます。この場合、最初の 4 バイトが I2C マスタに書き込まれ、すべてのバイトが I2C によって読み取られます。3 つ目のパラメータはデータへのポインタです。

```
EzI2Cs_SetRamBuffer(sizeof(MyI2C_Regs), 4,(BYTE *) &MyI2C_Regs);
```

以下の例では、15 バイトのアレイが作成され、I2C インターフェイスに公開されます。アレイの最初の 8 バイトは読み取り / 書き込み、残りの 7 バイトは読み取り専用です。

```
char theArray[15];
EzI2Cs_SetRamBuffer(15, 8, (BYTE *) theArray);
```

以下の例は、単一の整数 (2 バイト) のみが公開される、非常に単純な例です。両バイトとも、I2C マスタによる読み取りと書き込みが可能です。

```
int iRegister;
EzI2Cs_SetRamBuffer(2, 2, (BYTE *) (&iRegister);
```

ROM 領域インターフェイス (オプション)

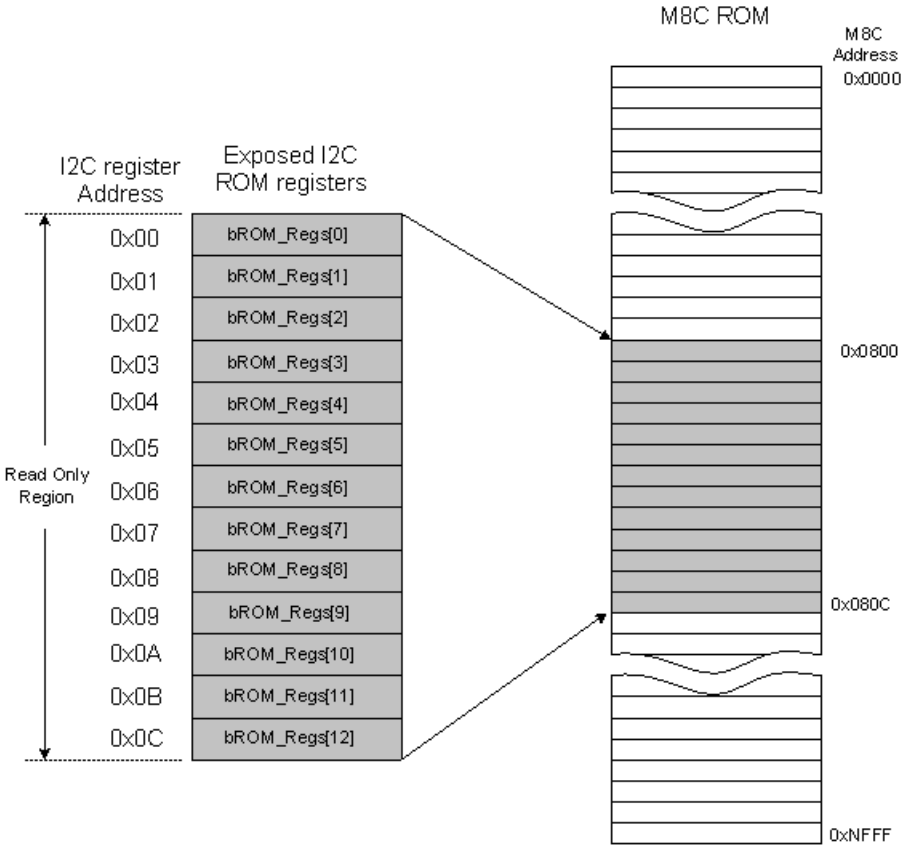
ROM インターフェイスは、読み取り専用であるという以外は、RAM インターフェイスとほぼ同じです。以下の例で見られるように、SetRomBuffer 関数は、SetRamBuffer 関数と似ています。この例では、データ領域は単純な定数文字列 (長さは 13 バイト) で、終端 NULL (0x00) 文字が続きます。

```
Const BYTE bROM_Regs[13] = "I2C ROM AREA";

EZI2Cs_SetRomBuffer(13, (const BYTE *)bROM_Regs);
```

前述のように、ROM 領域は独自の I2C アドレスを持ちます。このアドレスは、0x40 で論理和した RAM 領域アドレスです。メイン RAM 領域のアドレスが 0x05 である場合は、ROM 領域にアクセスする I2C アドレスは、0x45 となります。この図は、この例がメモリでどのように表現されるかを示しています。

Figure 3. ROM 領域インターフェイス



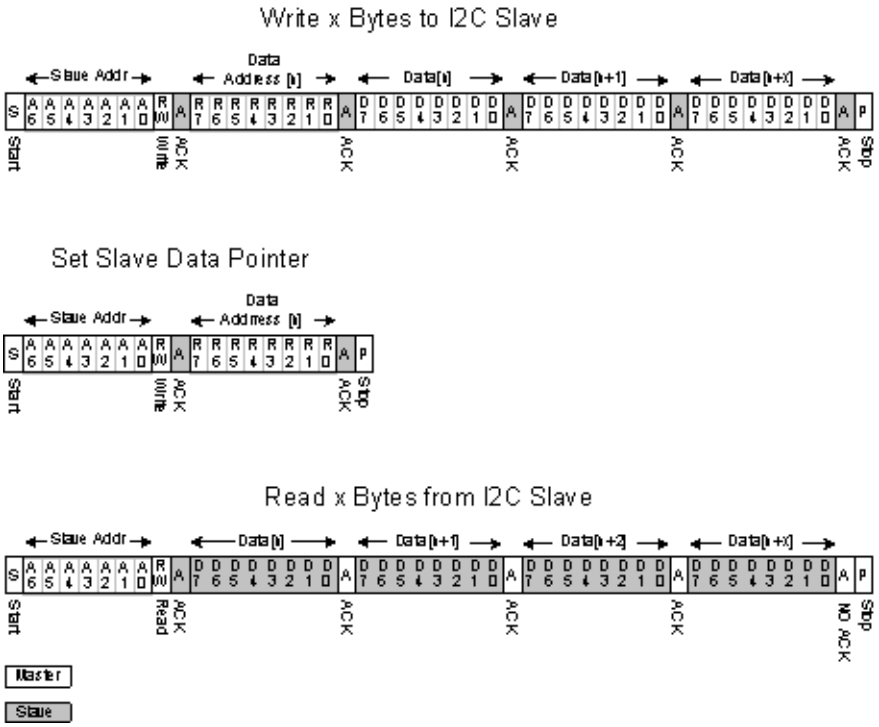
外部マスタから見たインターフェイス

EzI2C ユーザ モジュールは、RAM 領域での基本的な読み取りおよび書き込み、ROM 領域での読み取り専用動作をサポートします。RAM および ROM 領域インターフェイスは、書き込み動作の最初のデータ バイトで設定され、個別のデータ ポインタが含まれます。ROM 領域でも、データ ポイントを設定する、1 バイトの書き込みを受け入れます。1 または複数の RAM バイトが書き込まれる場合は、最初のデータ バイトは常にデータ ポインタです。データ ポインタに続くバイトが、データ ポインタ バイトでポイントされる位置に書き込まれます。3 番目のバイト (2 番目のデータ バイト) は、ポインタ + 1 に書き込まれるというように、続きます。このデータ ポインタは、各バイトの読み取りまたは書き込みで増分されますが、新しい読み取り動作の開始時に、書き込まれる最初の値にリセットされます。新しい読み取り動作は、データ ポインタでポイントされている位置でデータの読み取りを開始します。

たとえば、データ ポインタが 4 に設定されている場合は、読み取り位置が 4 でのデータ読み取りを開始し、データの終わりまたはホストが読み取り動作を終えるまで、連続して実行されます。たとえば、データ ポインタが 4 に設定されている場合は、各読み取り動作はデータ ポインタを 4 にリセットし、その位置から連続して読み取りを行います。これは、単一または複数のいずれの読み取り動作が実行される場合でも同じです。データ ポインタは、新しい書き込み動作が開始されるまで変更されません。

I^{2}C マスタが、SetRamBuffer() 関数で指定されている領域を超えるデータの書き込みを試みると、データが破棄されるため、RAM 内部または指定された RAM 領域外には影響を与えません。データは許可されている範囲外から読み取ることはできません。マスタから、許可されている範囲外からの読み取りがリクエストされると、無効の戻り値が返されます。

以下の図は、データ書き込み、データ ポインタ書き込み、データ読み取り動作のバス通信を示しています。データ書き込み動作は、データ ポインタを常に書き換えるので注意が必要です。



リセット時または電源オン時に、EzI2C ユーザ モジュールが構成され、API が供給されますが、リソースは EzI2Cs_Start() 関数を使って、明示的にオンにする必要があります。

I²C リソースは、バイト - バイトでデータ転送をサポートします。各アドレスまたはデータ送受信の終わりで、ステータスが報告されるか、指定された割り込みがトリガされます。ステータスの報告と割り込みの生成は、データ転送の方向およびハードウェアで検出される I²C バスの状態に依存します。割り込みは、バイトの完了時または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 トランザクションは、開始、アドレス、R/W 方向、データ、終端で構成されます。アドレス付きの I²C スレーブでは、割り込みは、受信データの 8 番目のビット後に発生します。バス上に別のスレーブがある場合は、アドレスがバスに配置された後、1 つの割り込みのみが存在します。この時点で、受信デバイスは、アドレスまたはデータ受信時、受信バイトの認識 (ack) または非認識 (nak) を決定する必要があります。次に、受信デバイスが I2C_SCR レジスタに適切な制御ビットを書き込み、ack/nak ステータスを I²C リソースに伝えます。I2C_SCR レジスタへの書き込みは、バスをインストールし、ack/nak ステータスをバス上に配置して次の受信データをシフトします。トランスミッタの 2 つ目の場合では、外部受信デバイスが ack または nak を提供した後で割り込みが発生します。このビットのステータスを決定するため、I2C_SCR が読み取られる場合があります。トランスミッタでは、データが I2C_DR レジスタにロードされ、次の送信をトリガし、I2C_SCR レジスタが再び書き込まれます。

I²C バスの詳細な説明とここでの実装は、フィリップス社のウェブサイトに完全な I²C の仕様として掲載されています。また、PSoC Designer とともに提供されているデバイス データシートにも記載されています。

設計について

メイン オシレータは、どのようなクロック速度にでも設定できます。データのスループットは、プロセッサの速度に影響されることがありますが、バイトごとのデータ転送は、指定された I²C 速度で動作します。頻繁に割り込みを生成する (非常の高速なアナログからデジタルへの変換器など) その他のユーザ モジュールを使用している場合は、メイン オシレータのクロック速度を高めるか、その他のユーザ モジュールの速度を遅くして、I²C を介した通信を維持しなければならない場合があります。I²C 割り込みは、ほとんどのその他の割り込みより低い優先順位であるため、別のユーザ モジュールが割り込みを高速で生成すると、I²C 割り込みがまったくまたはほとんど実行されないという状態が発生します。

ダイナミックな再構成

EzI2C リソースをダイナミックにロードされるまたはロードされないオーバーレイに統合しないでください。EzI2C リソースは、ベース　コンフィグレーションの一部としてのみ配置します。EzI2C ブロックの動作は、動作要件により変更できますが、ダイナミック　リコンフィグレーションの一部としてリソースを削除しようとすると、外部 I²C デバイスに逆効果を与えることがあります。また、I²C 関数を無効にするようピンが誤ってリコンフィグレーションされる可能性があります。コンフィグレーション間のスイッチングでは、この可能性を避けるよう注意してください。

外部電気接続

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、バス上のキャパシタンスによって決定されます。出力ステージでは、すべてのデバイス (マスタまたはスレーブ) で、最小シンク電流を V_{OLmax} = 0.4V で 3 mA 以上にしてください。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、バス上のキャパシタンスとクロック速度によって異なります。150 pF のバス上にキャパシタンスを持つ 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、フィリップス社のウェブサイト (www.philips.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、フィリップス社 I²C 特許権を譲渡し、これらのコンポーネントを フィリップス社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。

DC および AC の電気的特性

I²C インターフェイ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。

配置

EzI2C ユーザ モジュールでは、デジタルまたはアナログ PSoC ブロックは必要ありません。配置制限はありません。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

Slave_Addr

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、スレーブを指定します。有効な選択肢は、RAM レジスタが使用されている場合にのみ、0x00 ～ 0x7F です。ROM レジスタが有効になっている場合は、0x00 ～ 0x3F のアドレスのみが有効です。これは、ROM レジスタのスペースが、RAM アドレス + 0x40 (0x40 ～ 0x7F) であるためです。

Auto_Addr_Check

ハードウェア アドレス機能は、無効または有効のいずれかを選択します。無効にすると、ハードウェア アドレスの比較機能が利用できなくなります。このオプションを有効にすると、I2C ブロックが特定のシステム アドレス定義をサポートしなくなります。

このパラメータは、CY8C28045 デバイスでのみユーザが設定できます。

Address_Type

アドレスがスタティックかダイナミックかを選択します。スタティックに設定すると、I²C アドレスを変更することができなくなりますが、RAM を 1 バイト節約できます。このオプションをダイナミックに選択すると、ファームウェアがアドレスをいつでも変更できるようになります。このオプションは、I²C アドレスの LSB を設定するために外部レジスタまたはディップ スイッチが使用される場合の用途では一般的です。

ROM_Registers

RAM 領域は常に有効になりますが、ROM 領域へのアクセスを許可する 2 つ目のアドレスを有効にできます。ROM アドレスを使用する場合はこのパラメータのオプションを有効にし、必要ない場合は無効にしてください。この 2 つ目のアドレスは、0x40 との論理和のベース アドレスです。

CPU_CLK_SPEED_CY8C27xxxA

このパラメータは、初期の CY8C27xxxA ファミリのパーツに存在した エラーを修正します。この状態は、CY8C27xxxB ファミリ や、ハードウェア I²C をサポートするその他の全ファミリで、修正されています。CPU が 6 MHz 以下で動作する場合は、**6MHz または Less** オプションを選択してください。CPU クロックが 6 MHz を超える場合は、**Above 6MHz (CY8C27xxxA のみ)** のオプションを選択してください。CY8C27xxxA パーツ以外の全パーツでは、**Not CY8C27xxxA** オプションを選択してください。

このパラメータは、エラーが修正された製品ファミリでは表示されません。

I2C_Clock

このパラメータは、このスレーブで使用されるクロック速度を選択します。オプションは、50 kHz、100 kHz、400 kHz です。IMO (SysClk) が 24 MHz である場合のみ、400 kHz I^2C 速度を使用できます。

I2C_Pin

I^2C 信号との使用では、ポート 1 からピンを選択してください。PSoC Designer が自動で行うため、ピンの正しいドライブ モードを選択する必要がありません。これで、I^2C クロックとデータ信号を P1[5] - P1[7] または P1[1] - P1[0] に配置できます。

割り込み生成の制御

PSoC Designer で、**[Enable interrupt generation control (割り込み生成の制御を有効にする)]** チェックボックスが選択されいる場合は、さらにもう 1 つのパラメータが利用できるようになります。これは、**[Project (プロジェクト)] - > [Settings (設定)] - > [Chip Editor (チップ エディタ)]** から利用できます。「割り込み生成の制御」は、オーバーレイ全体で複数のユーザ モジュールで割り込みを共有し、複数のオーバーレイで使用される場合に重要です。

IntDispatchMode

IntDispatchMode パラメータは、同じブロック、異なるオーバーレイに存在する複数のユーザ モジュールによって共有される割り込みで、割り込みリクエストをどのように取り扱うかを指定します。「ActiveStatus」を選択すると、ファームウェアが、共有される割り込みリクエストに応答する前に、どちらのオーバーレイがアクティブであるかをテストします。このテストは、共有割り込みがリクエストされるたびに実行されます。これはレイテンシーが増加し、共有割り込みリクエストに対応する非確定プロシージャ (nondeterministic procedure) を生成しますが、RAM は必要としません。「OffsetPreCalc」を選択すると、ファームウェアが、オーバーレイが最初にロードされるときだけ、共有割り込みリクエストのソースを計算するようになります。この計算は、割り込みレイテンシーを低減し、共有割り込みリクエストに応答する確定プロシージャ (deterministic procedure) を生成しますが、RAM のバイト消費が発生します。

アプリケーション プログラミング インタフェース (API)

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ることがあります。A と X の値が呼び出し後に必要な場合は、呼び出し元関数で A と X の値を保存してください。PSoC Designer のバージョン 1.0 以降では、効率を高めるために、この「registers are volatile (レジスタの揮発性)」ポリシーが選択され、実施されています。C コンパイラは、自動的にこの条件で処理されています。アセンブラ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。

API 変数

2 以上のバイト長を持つ変数のデータ整合性を保証するには、コードでこれらの変数を変更する前に、EzI2Cs_bBusy_Flag 変数をチェックしてください。例 :

```
if(!(EzI2Cs_bBusy_Flag == EzI2Cs_I2C_BUSY_RAM_READ))
{
data.wData1++; //safely increment some 2 byte variable
}
```

EzI2Cs_bBusy_Flag 変数は ISR で自動的に更新され、以下の事前定義値を持ちます。

Table 1. EzI2Cs_bBusy_Flag 変数の事前定義値

シンボル名	値	説明
I2C_FREE	0x00	現在のトランザクションはありません。
I2C_BUSY_RAM_READ	0x01	RAM の読み取りトランザクションが進行中です。
I2C_BUSY_RAM_WRITE	0x03	RAM の書き込みトランザクションが進行中です。
I2C_BUSY_ROM_READ	0x01	ROM の読み取りトランザクションが進行中です。
I2C_BUSY_ROM_WRITE	0x03	ROM の書き込みトランザクションが進行中です。

API 関数

EzI2Cs_Start

説明：

I^{2}C スレーブと割り込みを有効にします。この関数を呼び出す前に、必要な EzI2Cs_SetRamBuffer() 関数を呼び出します。

C プロトタイプ：

```
void  EzI2Cs_Start(void);
```

アセンブラ：

```
lcall EzI2Cs_Start
```

パラメータ：

なし

戻り値：

なし

副作用：

この関数によって、A および X レジスタが変更される場合があります。

EzI2Cs_SetAddr

説明：

この関数は、EzI2C ユーザ モジュールによって認識されるアドレスを設定します。このコマンドは、Address_Type がパラメータで「Dynamic (ダイナミック) 」に設定されている場合にのみ有効です。この関数は、EzI2Cs_Start() 関数の後でのみ呼び出します。これは、開始関数がデフォルトのアドレスを設定し、この関数がこのアドレスを上書きするためです。この関数はオプションで、デフォルトのアドレスを変更しない限り必要ありません。

C プロトタイプ：

```
void  EzI2Cs_SetAddr(BYTE bAddr);
```

アセンブラ：

```
mov    A,0x05             ; Set I2C slave address to 0x05
lcall  EzI2Cs_SetAddr
```

パラメータ：

BYTE char：スレーブのアドレスは、1 ～ 127 です。

戻り値：

なし

副作用：

この関数によって、A および X レジスタが変更される場合があります。

EzI2Cs_GetActivity

説明：

この関数は、最後にこの関数が呼び出されてから、I²C の読み取りまたは書き込みが発生した場合に、ゼロ以外の値を返します。関数の呼び出しの最後で、動作フラグがゼロにリセットされます。

C プロトタイプ：

```
BYTE EzI2Cs_GetActivity(void);
```

アセンブラ：

```
lcall  EzI2Cs_GetActivity     ; Return value in A
```

パラメータ：

なし。

戻り値：

I²C の読み取りまたは書き込みが発生した場合に、ゼロ以外の値を返します。最後にこの関数が呼び出されてから動作が実行されなかった場合は、ゼロを返します。

副作用：

この関数によって、A および X レジスタが変更される場合があります。

EzI2Cs_GetAddr

説明：

この関数は、この I²C スレーブの現在のアドレスを返します。この関数はオプションで、通常、アプリケーションが現在またはデフォルトのアドレスを決定しなければならない場合にのみ必要です。

C プロトタイプ：

```
BYTE EzI2Cs_GetAddr(void);
```

アセンブラ：

```
lcall  EzI2Cs_GetAddr     ; Return value in A
```

パラメータ：

なし。

戻り値：

この I² スレーブの現在の I²C アドレスを返します。

副作用：

この関数によって、A および X レジスタが変更される場合があります。

EzI2Cs_Stop

説明：

I²C の割り込みを無効にして、EzI2C を無効にします。

C プロトタイプ：

```
void   EzI2Cs_Stop(void);
```

アセンブラ：

```
lcall  EzI2Cs_Stop
```

パラメータ：

なし

戻り値：

なし

副作用：

この関数によって、A および X レジスタが変更される場合があります。

EzI2Cs_DisableSlave

説明：

I²C の割り込みを無効にせずに、EzI2C を無効にします。

C プロトタイプ：

```
void   EzI2Cs_DisableSlave(void);
```

アセンブラ：

```
lcall  EzI2Cs_DisableSlave
```

パラメータ：

なし

戻り値：

なし

特殊作用：

この関数によって、A および X レジスタが変更される場合があります。

EzI2Cs_EnableInt

説明：

I²C 割り込みを有効にして、開始状態の検出を可能にします。以下のマクロを使って、グローバル割り込みのイネーブル関数を有効にしてください。M8C_EnableGInt. この関数は、EzI2Cs_Start 関数が呼び出される場合は必要ありません。デフォルトで、この関数は、保留中の割り込みをクリアします。保留中の割り込みをクリアしないようにするには、「ResumeInt 関数」を参照してください。

C プロトタイプ：

```
void   EzI2Cs_EnableInt(void);
```

アセンブラ：

```
lcall  EzI2Cs_EnableInt
```

パラメータ：

なし

戻り値：

なし

副作用：

この関数によって、A および X レジスタが変更される場合があります。

EzI2Cs_ResumeInt

説明：

I^2C 割り込みを有効にして、開始状態の検出を可能にします。この関数は、割り込みを有効にする前に、保留中の割り込みをクリアしません。以下のマクロを使って、グローバル割り込みのイネーブル関数を有効にしてください。M8C_EnableGInt. この関数は、EzI2Cs_Start 関数が呼び出される場合は必要ありません。

C プロトタイプ：

```
void   EzI2Cs_ResumeInt(void);
```

アセンブラ：

```
lcall  EzI2Cs_ResumeInt
```

パラメータ：

なし

戻り値：

なし

副作用：

この関数によって、A および X レジスタが変更される場合があります。

EzI2Cs_DisableInt

説明：

SDA 割り込みを無効にして、I^2C スレーブを無効にします。EzI2Cs_Stop. と同じように動作します。

C プロトタイプ：

```
void   EzI2Cs_DisableInt(void);
```

アセンブラ：

```
lcall EzI2Cs_DisableInt
```

パラメータ：

なし

戻り値：

なし

副作用：

この関数によって、A および X レジスタが変更される場合があります。

EzI2Cs_SetRamBuffer

説明：

この関数は、I²C マスタに公開される RAM バッファ (255 バイトまで) の位置とサイズを設定します。この関数は必須で、EzI2Cs_Start(). を呼び出す前に呼び出します。

C プロトタイプ：

```
void EzI2Cs_SetRamBuffer(BYTE bSize, BYTE bRWboundry, (BYTE *)pAddr);
```

アセンブラ：

```
lcall  EzI2Cs_SetRamBuffer
```

パラメータ：

BYTE bSize：I²C マスタに公開されるデータ構造のサイズ。bSize の最小値は 1、最大値は 255 です。BYTE bRWboundry：先頭から始まる、書き込み可能な位置のサイズ。正しい動作を実行するには、この値が常に bSize 以下となる必要があります。BYTE * pAddr：データ構造へのポインタ。

戻り値：

なし

副作用：

この関数によって、A および X レジスタが変更される場合があります。bRWboundry が bSize よりも大きい値に設定されると、bRWboundry 全体のサイズが書き込み可能になります。bSize を 0 の無効値に設定すると、I²C ホストが未定義のメモリ位置に書き込めるようになります。

EzI2Cs_SetRomBuffer

説明：

この関数は、I²C マスタに公開される ROM バッファ (255 バイトまで) の位置とサイズを設定します。ROM を使用する場合、この関数は、EzI2Cs_Start 関数を有効にする前に必要になります。

C プロトタイプ：

```
void EzI2Cs_SetRomBuffer(BYTE bSize,(const BYTE *)pAddr);
```

アセンブラ：

```
lcall  EzI2Cs_SetRomBuffer
```

パラメータ：

BYTE bSize：I²C マスタに公開されるデータ構造のサイズ。BYTE * pAddr：Flash ROM でのデータ構造へのポインタ。

戻り値：

なし

副作用：

この関数によって、A および X レジスタが変更される場合があります。

ファームウェア ソースコードの例

C コードの例

```
//************************************************************
// Example code to demonstrate the use of the EzI2Cs UM
//
// This code sets up the EzI2Cs slave to expose both
// a RAM and a ROM data structure.  The RAM structure is
// addressed at I2C address 0x05, and the ROM structure
// is at I2C address 0x45.
//
// * The instance name of the EzI2Cs User Module is “EzI2Cs".
// * The “Address_Type" parameter is set to “Dynamic"
// * The “ROM_Registers" parameter is set to “Enable"
//
// NOTE: to guarantee data integrity of variables with length 2 or more
// bytes check the EzI2Cs_bBusy_Flag variable before changing values of such
// variables.
//************************************************************

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

struct I2C_Regs {    // Example I2C interface structure
    BYTE bStat;
    BYTE bCmd;
    int  iVolts;
    char cStr[6];    // Read only string
} MyI2C_Regs;

const BYTE DESC[] = "Hello I2C Master";

void main(void)
{

    // Set up RAM buffer
   EzI2Cs_SetRamBuffer(sizeof(MyI2C_Regs), 4, (BYTE *) &MyI2C_Regs);


   EzI2Cs_SetRomBuffer(sizeof(DESC), DESC);   // Set up ROM buffer

   M8C_EnableGInt ;                           // Turn on interrupts

   EzI2Cs_Start();                            // Turn on I2C
   EzI2Cs_SetAddr(5);                         // Change address to 5


   while(1) {
      // Place user code here to update and read structure data.
   }
}
```

ASM コードの例

```
;-------------------------------------------------------------------------
```

```
; Example code to demonstrate the use of the ExI2Cs UM
;
; This code sets up the EzI2Cs slave to expose both
; a RAM and a ROM data structure.  The RAM structure is
; addressed at I2C address 0x05, and the ROM structure
; is at I2C address 0x45.
;
; * The instance name of the EzI2Cs User Module is "EzI2Cs".
; * The "Address_Type" parameter is set to "Dynamic".
; * The "ROM_Registers" parameter is set to "Enable".
;
; NOTE: to guarantee data integrity of variables with length 2 or more
; bytes check the EzI2Cs_bBusy_Flag variable before changing values of such
; variables.
;-----------------------------------------------------------------------

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

area bss (RAM, REL, CON)        ; Allocate Variables & define constants
RAM_BUF_SIZE:     equ  10       ; Top of ramp value
BUF_RW_SIZE:      equ   5       ; Only first five bytes writable from I2C Master

I2C_RAM_Buf:  blk  RAM_BUF_SIZE ; Reserve RAM for I2C buffer

area text (ROM, REL, CON)

.LITERAL
I2C_ROM_BUF:                    ; I2C ROM Buffer
   DB  11h, 22h, 33h, 44h, 55h, 66h, 77h, 88h
.ENDLITERAL
ROM_BUF_SIZE:     equ   8

export _main

_main:
    ; Set RAM Buffer
    mov    A,>I2C_RAM_Buf       ; Save MSB of RAM buffer address
    push   A
    mov    A,<I2C_RAM_Buf       ; Save LSB of RAM buffer address
    push   A
    mov    A,BUF_RW_SIZE        ; Save RW size param
    push   A
    mov    A,RAM_BUF_SIZE       ; Save I2C buffer size
    push   A
    call   EzI2Cs_SetRamBuffer
    ADD    SP,-4                ; Reset Stack

    ; Set ROM Buffer
    mov    A,>I2C_ROM_BUF       ; Save MSB of ROM buffer address
    push   A
    mov    A,<I2C_ROM_BUF       ; Save LSB of ROM buffer address
    push   A
    mov    A,ROM_BUF_SIZE       ; Save ROM buffer size
```

```
    push   A
    call   EzI2Cs_SetRomBuffer
    add    SP,-3                ; Reset Stack

    M8C_EnableGInt              ; Enable Global Interrupts

    call   EzI2Cs_Start         ; Start and enable IRQ
    mov    A,5                  ; Set address to 5
    call   EzI2Cs_SetAddr

.Loop:

    ;; User Code goes here

    jmp    .Loop
```

設定レジスタ

ここでは、EzI2C ユーザ モジュールで使用または変更される PSoC リソース レジスタについて説明します。EzI2C ユーザ モジュールを使用する際に、これらのレジスタの使用は必須ではありませんが、リファレンスとして提供されています。

Table 2. リソース I2C_CFG：バンク 0 reg[D6] 設定レジスタ

ビット	7	6	5	4	3	2	1	0
値	予約	PinSelect	Bus Error IE (バス エラー IE)	Stop IE (ストップ IE)	Clock (クロック) Rate[1]	Clock (クロック) Rate[0]	0	スレーブの有効化

ピン選択：P1[5] - P1[7] または P1[1] - P1[0] から、SCL および SDA のを選択します。

バス エラー割り込みの有効化：バス エラーでの I^2C 割り込みの生成を有効にします。

ストップ エラー割り込みの有効化：I^2C 停止状態で I^2C 割り込みの生成を有効にします。

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

スレーブの有効化：バス スレーブとして、I^2C ハードウェア ブロックを有効にします。

Table 3. リソース I2C_SCR：バンク 0、reg[D7] ステータス制御レジスタ

ビット	7	6	5	4	3	2	1	0
値	Bus Error (バス エラー)	なし	Stop Status (停止ステータス)	ACK out (ACK 出力)	Address (アドレス)	Transmit (送信)	Last Recd Bit (LRB) (最後の受信ビット：LRB)	Byte Complete (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。

停止ステータス：I^2C 停止状態の検出を示します。

ACK 出力：I^2C ブロックを受信バイトの認識 (1) または非認識 (0) にダイレクトします。

アドレス：受信または送信バイトがアドレスです。

最後の受信ビット (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 ステータス。

バイト完了：8 バス ビットが受信されました。受信モードでは、バスが Ack/Nac を待機するためストールします。送信モードでは、Ack Nac も受信され (SRB を参照)、バスの次の動作がストールします。

Table 4. 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							

受信または送信したデータ。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、受信内容にはアドレスやデータが含まれる場合があります。

改訂履歴

バージョン	作成者	説明
1.2	DHA	SCL が低でスタックする状態を防止するよう更新されました。 Start 関数に、以下の変更が加えられました。 1. ユーザ モジュール ピンの初期のオープン - ドレイン低ドライブ モードが HI-Z アナログに変更されました。 2. ^{2}C ブロックを有効にしました。 3. 遅延 5 nop 命令を追加しました。 4. 初期の ^{2}C ピン ドライブ モードを復元しました。 EzI2C_StopSlave API をパブリックにしました。 Start API にオフセット ポインタの初期化を追加しました。

Note PSoC Designer 5.1 により、全ユーザ モジュール データシートが改訂されます。このセクションは、現在および以前のユーザ モジュール バージョン間の差を高レベルに説明するものです。

Copyright © 2004-2010 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.