SPI マスタ データシート SPIM V 2.6

Copyright © 2002-2011 Cypress Semiconductor Corporation. All Rights Reserved.

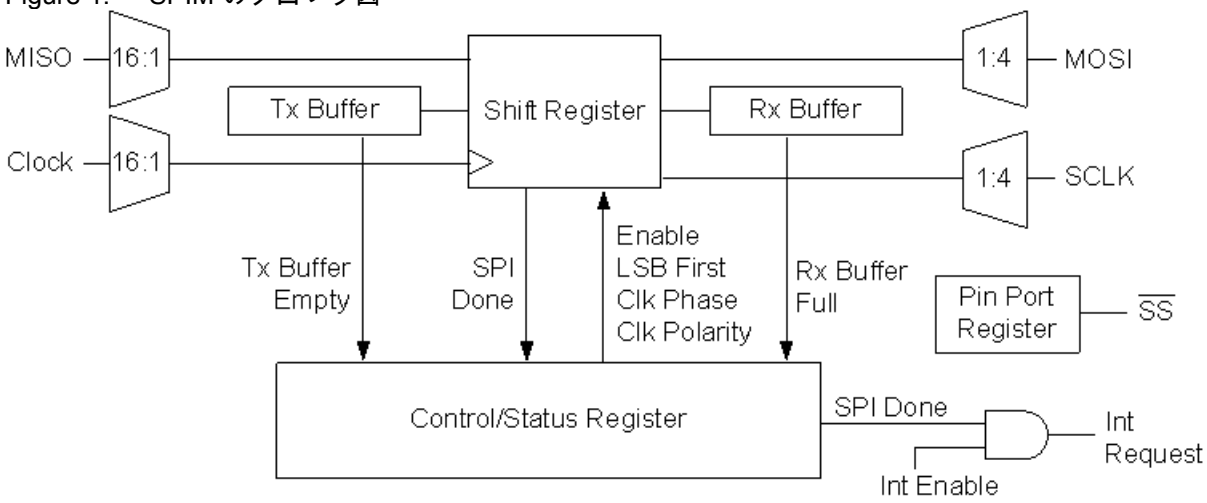
リソース	PSoC® ブロック			API メモリ（バイト）		ピン（外付け I/O あたり）
	デジタル	アナログ CT	アナログ SC	Flash	RAM	
CY8C29/27/24/22/21xxx, CY7C603xx, CY7C64215, CYWUSB6953, CY8C23x33, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01, CYRF69xx3, CY8C28xxx						
	1	0	0	27	0	3 - 4
CY8C26/25xxx	1	0	0	37	0	3 - 4

特徴と概要

- シリアルペリフェラル相互接続（SPI）マスタ プロトコルをサポート
- SPI クロックモード 0、1、2 をサポート
- クロックと MISO 用の選択可能な入力ソース
- MOSI と SCLK 用の選択可能な出力ルーティング
- SPI 済み状態へのプログラム可能な割り込み
- SPI スレーブデバイスは個別に選択することができます。

SPIM ユーザ モジュールは、シリアルペリフェラル相互接続マスタです。これは、全二重同期 8-bit データ転送を実行します。SCLK フェーズ、SCLK 極性、LSB 優先を指定すると、殆どの SPI クロックモードに対応できます。ユーザ提供のソフトウェアによって制御されるスレーブ選択信号は、1 つまたは複数の SPI スレーブデバイスを制御するように構成できます。SPIM PSoC ブロックには、入力・出力信号用の選択可能なルーティングと、プログラム可能な割り込みによる制御があります。

Figure 1. SPIM のブロック図



機能説明

SPIM は、シリアルペリフェラル相互接続マスタを実行するユーザ モジュールです。これは、デジタル コミュニケーション タイプ PSoC ブロックおよび 1 つ以上のピンポート レジスタに対する送信バッファ、受信バッファ、制御およびシフト レジスタを使用します。

制御レジスタは、SPIM デバイス エディタ、SPIM ユーザ モジュール ファームウェア アプリケーション プログラミング インタフェース (API) ルーチンを用いて初期化および構成されます。初期化には、LSB First の構成と SPI 送受信クロックモードの設定が含まれます。SPI モード 0、1、2、3 がサポートされます。SPI マスタとスレーブは両方とも、適切に通信するために、同じクロックモードとビット構成で設定される必要があります。SPI モードの定義は表 1 に記載されている通りです。

Table 1. SPI モード

モード	SCLK エッジ実行データラッチ	クロック極性	注
0	立ち上がり	反転なし	立ち上がりエッジ ラッチデータ。クロックの立ち下がりエッジへのデータ変更。
1	立ち上がり	反転済み	
2	立ち下がり	反転なし	立ち下がりエッジ ラッチデータ。立ち上がりエッジでデータ変化。
3	立ち下がり	反転済み	

Active Low スレーブ選択信号 ~SS は、選択されたピンポート レジスタのビットを制御し、SPI スレーブデバイスを適切に有効化するために、ユーザ提供ソフトウェア ルーチンに設定しなければなりません。1 つまたは複数のスレーブ選択信号を構成することが可能ですが、同じ時間でアクティブにできるスレーブ選択信号は 1 つだけです。

すべての SPI クロックモードに対応するために、スレーブ選択信号は、SPI マスタから選択された SPI スレーブデバイスへ送信されるデータの各バイト毎に、アサート / デアサートする必要があります。ただし、この要件は、SPI マスタとスレーブデバイス間の SPI 通信に使用される特定のバイト移動プロトコルに様々なものがあるため、必須ではありません。

SCLK 信号は SPI 送受信クロックです。このクロックレートは、入力クロック信号の半分です。有効な送受信ビット レートを求めるには、入力クロックを 2 で割ります。入力クロックはデバイス エディタを使用して定義されます。

MOSI 信号は、マスタからスレーブの SPI プロトコル準拠デバイスへデータを送信するマスタアウトスレーブイン データ信号です。MISO 信号は、スレーブからこのユーザ モジュールへデータを送信するマスタインスレーブアウト データ信号です。

SPIM ハードウェアは、マスタ SPI デバイスから MOSI 信号へのデータを送信し、また同時に選択されたスレーブ SPI デバイスから MISO 信号へのデータを受信します。同じ SCLK 信号が、マスタとスレーブデータの送受信に使用されます。

SPI プロトコルは、マスタだけが始動できるレスポンス プロトコルです。選択されたスレーブデバイスがコマンドの準備またはデータ受信の準備ができているかの判定を下すのは、マスタの役割です。

API ルーチンを使用して SPI イネーブル ビットが制御レジスタにセットされている場合、SPIM ユーザ モジュールの作動は有効です。このとき、すべてのスレーブ選択信号は High にアサートされています。

選択された SPI スレーブデバイスに 1 バイトを送信する前に、指定されたスレーブ選択信号は Low にアサートされています。

送信されるデータのバイトは、送信バッファ レジスタに書き込まれます。これにより、制御レジスタの送信バッファ Empty（空）ステータスビットがクリアされます。次のクロックで、送信バッファのデータはシフト レジスタに転送され、送信バッファ Empty（空）ステータスビットがセットされます。送信オーバーランの状態を避けるために、送信バッファ レジスタに 1 バイトを書き込む前に、バッファ空ステータスビットを確認しなければなりません。この時点では、送信する別のデータバイトを送信バッファ レジスタに書き込む（プリロード済み）ことができます。現在のバイトの送信完了後直ちに、このデータは次の SCLK 信号に送信されます。

シフト レジスタのデータバイトのうち各ビットに対して、SCLK 出力信号が生成され、シフト レジスタのデータが MOSI 出力にシフトされ、スレーブ SPI からの入力データは MISO 入力からシフト レジスタにシフトされます。SCLK、MOSI、MISO 信号のタイミングは、SPI クロックモード構成に基づいています。

すべてのビットが転送され、同時に受信された後、受信データがシフト レジスタから受信バッファ レジスタに転送され、送信バッファ レジスタはシフト レジスタに送信されます。受信バッファ Full（フル）と SPI 完了のステータスビットがセットされます。割り込みが有効の場合、SPI 完了ステータスビットはトリガに割り込みを生じます。

受信バッファ レジスタからデータバイトを受信する際に SPI 完了割り込み条件が使用されない場合、受信バッファ Full（フル）ステータスビットをモニタするために、制御レジスタはポーリングされます。受信データは、次のデータバイトの受信が完了する前、もしくはオーバラン エラー ステータスビットがセットされる前に、受信バッファ レジスタから読み込まなければなりません。

データの保留バイトが送信バッファ レジスタにプリロードされている場合、このバイトが SPI 状態マシンを再起動し、送信が直ちに開始します。

各データバイトの送信後にスレーブ選択信号の制御を実行するには、「割り込み無し」と「割り込みレベル」という 2 つの方法があります。どちらの方法を選択するかは、SPI クロックモード、バイト移動プロトコル、送信データのビットレート速度要件によって異なります。SPI クロックモード 0 と 1 では、スレーブ選択がトグルオフになっており、次のデータバイトの送信前に再びオンになっている必要があります。同じスレーブデバイスへの複数バイト送信に使用されるバイト移動プロトコルでは、スレーブ選択がトグルしていることが必要な場合と不要な場合があります。スレーブ選択のトグルでは、スレーブデバイスが分岐していることがあり、その場合は、SCLK 信号の実行前にスレーブデバイスがデータを MISO 信号にアサートするために、セットアップ時間が必要なこともあります。

割り込み無しレベルでスレーブ選択信号を制御する場合、SPI データの送信中は、SPI 完了ステータスビットのモニタまたはサンプリング専用にマイクロプロセッサが必要です。データのビット レートが 1 MHz レベルやそれ以上と非常に高い場合、ステータスビットのモニタリングでのオーバーヘッドは大幅に制限されます。

SPI 完了ステータスビットのモニタリングでのオーバーヘッドによって、マイクロプロセッサの動作が、他の必要な動作の妨げになるほどの低速のビットレートの場合、スレーブ選択制御は割り込みレベルで実行できます。最低割り込みレイテンシは 833 ns（クロックレートが 24 MHz）で、これにスレーブ選択を管理する指示を実行する時間を足します。

送信バッファに送信するデータがプリロードされている状態での複数バイトデータ転送は、心配の種となる場合があります。しかし、選択されたスレーブ信号の管理に必要なバイト間処理が最低またはゼロで、送信バイト転送が同じ SPI スレーブデバイスで実行されれば、1 MHz 以上のビット転送レートの達成は可能なことがあります。

最終データが送信されたら、SPIM ユーザ モジュールをディスエーブルにするタイミングを決定するために、SPI 完了ビットをモニタします。これにより、すべてのクロック信号がマスタとスレーブ SPI デバイスの間で完了したことが保証されます。

タイミング

SPIM 転送の通常のタイミングを次の図に示します。SPIM タイミングの詳細については、Technical Reference Manual を参照してください。

Figure 2.　モード 0 と 1 における通常の SPIM タイミング

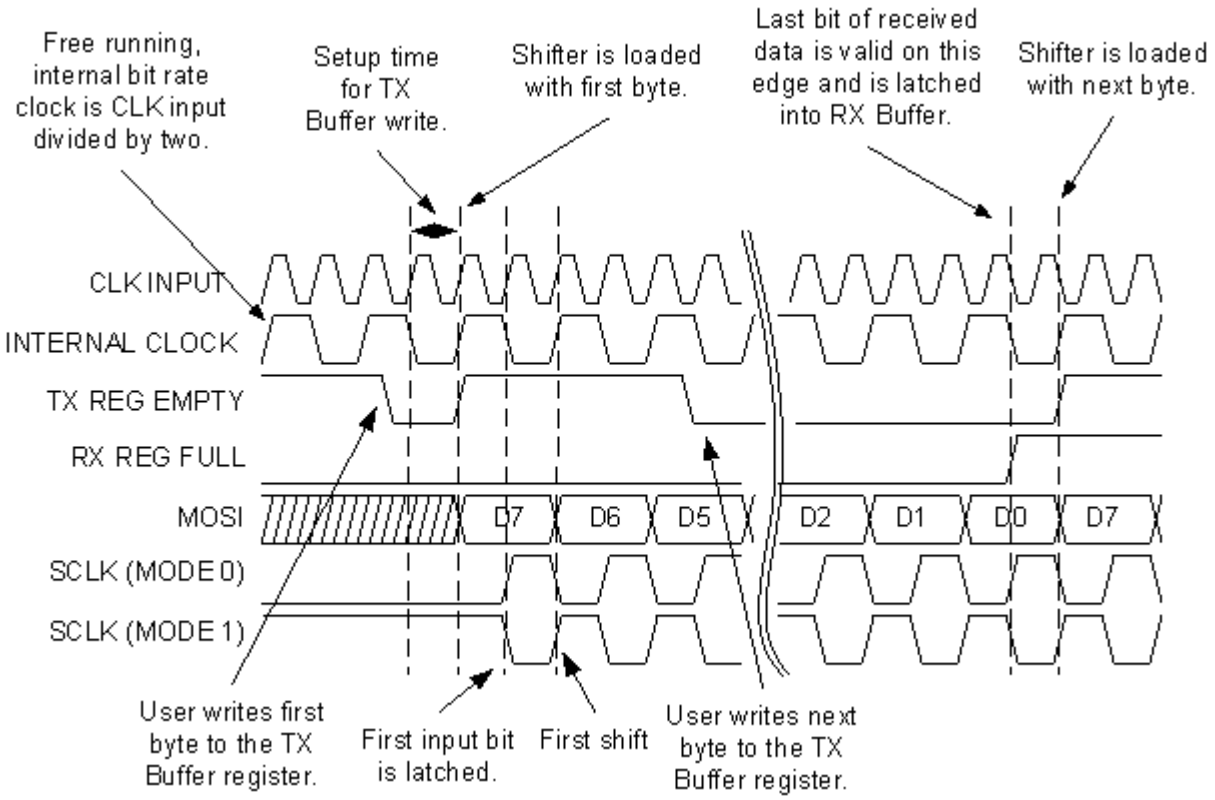
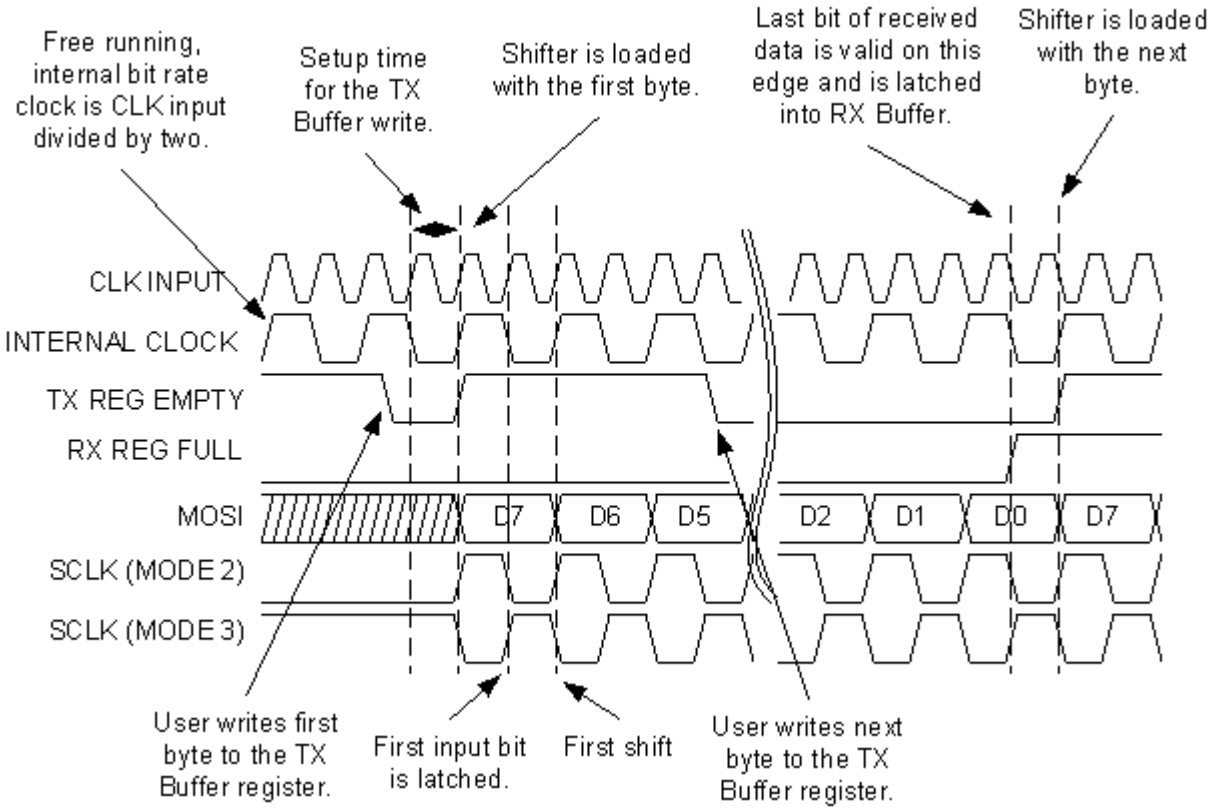


Figure 3. モード 2 と 3 における通常の SPIM タイミング



DC および AC の電気的特徴

Table 2. SPIM DC 電気的特性と AC 電気的特性

パラメータ	条件および注意	典型	制限	単位
F_{max}	最大ビット レート	4.1[a]	--	Mbps

a. 標準レートは、多くの PSoC デバイスにも当てはまりますが、最大ビット レートはデバイスによって異なります。最大データ転送速度は、最大クロック値を 2 で割ったものです。またこれは、デバイスの割り込みレイテンシに基づいて制限されることもあります。最大値については、デバイスのデータシートを参照してください。

配置

SPIM は 1 つの PSoC ブロックにマッピングされ、任意のデジタルコミュニケーションブロックに配置されることがあります。

ピンポート レジスタビットは、スレーブ選択信号で SPI スレーブデバイスを制御するために確保する必要があります。これらのポートビットは、標準 CPU ポートピンとして構成できます。

パラメータおよびリソース

Clock (クロック)

SPIM は 16 ソースのうちのひとつによってクロック処理されます。グローバル I/O バスは、クロック信号を別々の PSoC ブロックによって生成された外部のピンまたはクロック機能装置へ接続するのに使えます。ブロックで外部デジタル クロックを使用している場合は、最高の精度およびスリープ動作を得るため、列入力同期がオフになります。48 MHz クロック、CPU_32 kHz クロック、分割クロックのうちの 1 つ (24V1 もしくは 24V2)、または別の PSoC ブロック出力を SPIM クロック入力として指定できます。

クロック伝送率を、希望するビット レートの 2 倍に設定しなければなりません。1 つのデータビットは、入力クロック 2 つごとに送受信されます。

MISO

マスタインスレーブアウト入力信号は、16 の入力ソースのうちの 1 つに接続できます。使用可能な MISO ソースには、High、Low、グローバル I/O バス、アナログコンパレータバス、別の PSoC ブロックがあり、これらを MISO 入力に供給するために指定できます。

MOSI

SPIM のマスタアウトスレーブイン出力は、グローバル出力バスの 1 つに接続できます。その後、グローバル出力バスは、外部ピンやその他の PSoC ブロックに接続して、さらなる処理を実行できます。

SCLK

この出力クロックは、SPI マスタによって生成されます。これは通常、グローバル出力ラインの 1 つを介してポートピンに接続され、その後 SPI スレーブに接続されます。このクロックは、有効ビット転送レートを定義します。

割り込みモード

このオプションは、TX ブロック用に割り込みが生成されるタイミングを決定します。「TxRegEmpty」オプションでは、データレジスタからシフト レジスタにデータが転送されるとすぐに、割り込みを生成します。2 つ目のオプション 「TxComplete」では、最終ビットがシフト レジスタからシフトアウトするまで割り込みを延期します。このオプションは、文字が完全に送信されたことが確認できるという意味で役立ちます。最初のオプション 「TxRegEmpty」は、トランスミッタの出力を最大にする上で役立ちます。前のバイトの送信中に次のバイトを読む込むことができます。

ClockSync (クロック同期)

PSoC デバイスでは、デジタル ブロックは、システム クロックに加えて、クロック ソースを提供できます。デジタル クロック ソースは、リップル形式で連結することも可能です。これで、システム クロックに関するスキューが可能になります。様々なデータ最適化作業、とりわけシステム バスに適用された最適化作業のため、このようなスキューは CY8C29/27/24/22/21xxx および CY8CLED04/08/16 PSoC デバイスの製品群にとって特に重要です。このパラメータは、制御クロック スキューに使用され、PSoC ブロック レジスタ値を読み書きする場合に、正しい動作を保証します。このパラメータの適切な値は、以下の表から決定してください。

[+] Feedback

ClockSync 値	使用
SysClk への同期	2 以上で除算される 24 MHz (SysClk) 派生クロック ソースでこの設定を使用します。例には、VC1、VC2、VC3 (VC3 が SysClk によって駆動される場合)、32KHz、SysClk ベースのソースを持つデジタル PSoC ブロックが含まれます。正しい同期が行われるよう、外部で生成されたクロック ソースもこの値を使用してください。
SysClk*2 への同期	結果の周波数が 48 MHz でない限り (言い換えると、すべての除算結果が 1 になる場合)、48 MHz (SysClk*2) ベースのクロックでは、この設定を使用します。
SysClk 指示の使用	24 MHz (SysClk/1) クロックが適切な場合に使用します。これは、同期を実際には実行しませんが、システム クロック自体への低スキュー アクセスを提供します。選択すると、このオプションは、上の Clock パラメータの設定を上書きします。組み合わせた全除算値の最終結果が 24 MHz 出力を生成する場合は、VC1、VC2、VC3、またはデジタル ブロックの代わりに常に使用してください。
Unsynchronized (非同期)	48 MHz (SysClk*2) 入力が選択される場合に使用します。 非同期入力が適切な場合に使用します。一般に、割り込み生成がカウンタの単独の用途である場合にのみ使用することを推奨します。

InvertMISO

このパラメータは、MISO 入力の反転を可能にします。

タイミング

クロック伝送率を、希望するビット レートの 2 倍に設定しなければなりません。

SPI データ送信のタイミングは、スレーブ選択信号の処理によって発生する遅延の理由でもあります。SPI スレーブデバイスの選択信号の設定時間を考慮に入れることが重要です。

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ている場合、さらに 2 つのパラメータが使用できます。これは、**[Project] > > [Setting] > > [Chip Editor] を選択すると使用できます。**「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 モジュールにより共有される割り込みとともに、複数のオーバーレイが使用される場合に重要です。

- 割り込み API
- IntDispatchMode

InterruptAPI

InterruptAPI パラメータを使うと、ユーザ モジュールの割り込みハンドラと割り込みベクトル テーブル エントリの状況に応じた生成が可能になります。「Enable (有効)」を選択すると、割り込みハンドラと割り込みベクトル テーブル エントリが生成されます。「Disable (無効)」を選択すると、割り込みハンドラと割り込みベクトル テーブル エントリがバイパスされます。1 つのブロック リソースが異なるオーバーレイで使用されるような、複数のオーバーレイを持つプロジェクトでは特に、割り込み API が生成されるかどうかを正しく選択してください。割り込み API の生成のみを選択すると、割り込みディスパッチ コードを生成する必要がなくなり、オーバーヘッドを軽減できます。

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、共有割り込みリクエストに応答する前に、ファームウェアが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を生むほか、共有割り込みリクエストに対応する非決定性のプロシージャを生成しますが、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決定性のプロシージャを生成しますが、RAM の 1 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)

アプリケーション プログラミング インタフェース (API) ルーチンは、設計者が高級言語でモジュールを操作できるようにユーザ モジュールをコンポーネントとして提供します。このセクションでは、各機能に対するインタフェースを 「include」ファイルによって提供される関連定数とともに示します。

ユーザ モジュールを配置するたびに、インスタンス名がアサインされます。デフォルトでは、PSoC Designer は指定されたプロジェクトで、SPIM_1 をこのユーザ モジュールの最初の例として指定されています。これは識別子の構文ルールに従って、独自のルールに変更できます。割り当てられたインスタンスは、グローバル関数名、変数、定数記号のプレフィクスとなります。次の説明では、簡潔化するために、インスタンス名は省略されて単に 「SPIM」となっています。

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ることがあります。関数を呼び出す場合、呼び出し後に A と X の値が必要になるならば、必ず呼び出し前に 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。

このセクションでは、SPIM が供給する API 関数を挙げます。

SPIM_Start

説明：

SPI インターフェースのモード構成を設定し、制御レジスタに適切なビットをセットすることにより SPIM モジュールを有効にします。この関数を呼び出す前に、すべてのスレーブ選択信号は、接続されている SPI スレーブデバイスの選択を外すために高にアサートされていなければなりません。これはユーザ提供ルーチンで実行されるはずです。

C プロトタイプ：

```
void  SPIM_Start(BYTE bConfiguration)
```

アセンブリ：

```
mov   A,SPIM_MODE_2 | SPIM_LSB_FIRST
lcall  SPIM_Start
```

パラメータ：

bConfiguration: SPI モードと LSB 優先構成を指定する 1 バイト。累算器を通過します。C およびアセンブリで容易されたシンボル名、およびそれらに関連付けられた値は、以下の表で示されます。

注：記号名は 「OR」で結合し、SPI インターフェース構造を構成することができます。注：ユーザ モジュールのインスタンス名は、下記に挙げる記号名に添付されています。例えば、ユーザ モジュールの配置時に 「SPIM1」という名前をつけた場合、第 1 モードの記号名は 「SPIM1_SPIM_MODE_0.」となります。

記号名	値
SPIM_MODE_0	0x00
SPIM_MODE_1	0x02
SPIM_MODE_2	0x04
SPIM_MODE_3	0x06
SPIM_LSB_FIRST	0x80
SPIM_MSB_FIRST	0x00

戻り値：

なし

特殊作用：

A および X レジスタがこの機能により変更される場合があります。

SPIM_Stop

説明：

制御レジスタ中のイネーブル ビットをクリアして、SPIM モジュールを無効にします。この関数を呼び出すと、すべてのスレーブ選択信号が High にアサートされ、接続されているすべての SPI スレーブデバイスが無効になります。これはユーザ提供ルーチンで実行すべきものです。

C プロトタイプ：

```
void  SPIM_Stop(void)
```

アセンブリ：

```
lcall  SPIM_Stop
```

パラメータ：

なし

戻り値：

なし

特殊作用：

A および X レジスタがこの機能により変更される場合があります。

SPIM_EnableInt

説明：

SPI 完了状態における SPIM 割り込みを有効にします。SPIM の配置位置によって、割り込みベクトルと優先順位が決定します。

C プロトタイプ：

```
void  SPIM_EnableInt(void)
```

アセンブリ：

```
lcall  SPIM_EnableInt
```

パラメータ：

なし

戻り値：

なし

特殊作用：

A および X レジスタがこの機能により変更される場合があります。

SPIM_DisableInt

説明：

SPI 完了状態における SPIM 割り込みを無効にします。

C プロトタイプ：

```
void  SPIM_DisableInt(void)
```

アセンブリ：

```
lcall  SPIM_DisableInt
```

パラメータ：

なし

戻り値：

なし

特殊作用：

A および X レジスタがこの機能により変更される場合があります。

SPIM_SendTxData

説明：

スレーブ SPI デバイスへの SPI 送信を開始します。この呼び出し直前に、指定した SPI スレーブデバイスの信号が Low にアサートされる必要があります。これはユーザ提供ルーチンで実行されるはずです。

C プロトタイプ：

```
void  SPIM_SendTxData(BYTE bSPIMData)
```

アセンブリ：

```
mov   A, bSPIMData
lcall  SPIM_SendTxData
```

パラメータ：

BYTE bSPIMData: SPI スレーブデバイスに送信されるデータ。アキュムレーターを通過します。

戻り値：

なし

特殊作用：

A および X レジスタがこの機能により変更される場合があります。

SPIM_bReadRxData

説明：

スレーブデバイスからの受信データバイトを戻します。データバイトが受信されたことを検証するために、このルーチン呼び出し前に受信バッファ Full（フル）のフラグがチェックされます。

C プロトタイプ：

```
BYTE  SPIM_bReadRxData(void)
```

アセンブリ：

```
lcall  SPIM_bReadRxData
mov   bRxData, A
```

パラメータ：

なし

戻り値：

スレーブ SPI から受信し、アキュムレーターを通して戻されるデータバイト。

特殊作用：

A および X レジスタがこの機能により変更される場合があります。

SPIM_bReadStatus

説明：

現在の SPIM 制御 / 状態レジスタを読み取り、これを戻します。

C プロトタイプ：

```
BYTE  SPIM_bReadStatus(void)
```

アセンブリ：

```
lcall  SPIM_bReadStatus
and   A, SPIM_DONE | SPIM_RX_BUFFER_FULL
jnz   SpimCompleteGetRxData
```

パラメータ：

なし

戻り値：

状態バイト読み取り結果を返し、累算器を通して戻されます。指定されたマスクを使用し、特定の状態条件をテストします。注：マスクを OR 処理することで、結合条件をテストできます。注：ユーザ モジュールのインスタンス名は、下記に挙げる記号名に添付されています。例えば、ユーザ モジュールの配置時に 「SPIM1」という名前をつけた場合、第 1 マスクの記号名は「SPIM1_SPIM_SPI_COMPLETE.」となります。

SPIM 状態マスク	値
SPIM_SPI_COMPLETE	0x20
SPIM_RX_OVERRUN_ERROR	0x40
SPIM_TX_BUFFER_EMPTY	0x10
SPIM_RX_BUFFER_FULL	0x08

特殊作用：

この関数が呼び出されると、ステータスビットはクリアされます。A および X レジスタがこの機能により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

次の C とアセンブリ サンプルコードでは、SPI マスタが RAM に保存されているゼロ終端文字列を送信します。この関数は、SPIM API を使用してデジタル PSoC ブロック送信レジスタに、1 回に 1 バイトずつ読み込みます。このデジタルブロックは、送信レジスタをシフトレジスタに転送することで、送信を開始します。API 呼び出しを繰り返すことで、この関数は送信レジスタに次のバイトをすぐに再読み込みします。これにより、最大連続レートで送信が進行します。この簡略化バージョンでは、このスレーブからの受信データは破棄されます。この関数は、最終バイトが送信レジスタに読み込まれた直後に戻りますが、そのとき、最終バイトの次のバイトは MOSI ラインからシフトアウトしている最中です。送信レジスタに値を読み込む前にステータスを必ず確認するので、この関数が次に呼び出されても、実行中の転送が破損することがありません。

```
#include "PSoCAPI.h"    // PSoC API definitions for all User Modules

CHAR Message1[] = "Hello World.";
CHAR *pbStrPtr = Message1;

void main(void)
{
    SPIM_Start(SPIM_SPIM_MODE_0 | SPIM_SPIM_MSB_FIRST);

    while( *pbStrPtr != 0 )  /* While data remains to be sent */
    {
        /* Ensure the transmit buffer is free */
        while( ! (SPIM_bReadStatus() & SPIM_SPIM_TX_BUFFER_EMPTY ) );

        SPIM_SendTxData( *pbStrPtr );  /* load the next byte */
        pbStrPtr++;
    }
}
```

アセンブル言語で書かれた同等コード：

```
AREA bss (RAM, REL)
Message1: blk 13
AREA text (ROM, REL)

_main:

    ; [...]                    ; ( String is copied to RAM )

    mov   A, SPIM_SPIM_MODE_0 | SPIM_SPIM_MSB_FIRST
    call  SPIM_Start           ; Initialize the digital PSoC Block
    mov   X, Message1          ; Set index to beginning of string.
    ; [...]                    ; Set SlaveSelect signal low to enable slave)
.TxNextByteLoop:
    mov   A, [X+0]             ;    Data remaining to be sent? ...
    jz    .Finish              ;    No, bail out of the loop.
.WaitForTxEmpty:
    call  SPIM_bReadStatus     ;   Transmit buffer free? ...
    and   A, SPIM_SPIM_TX_BUFFER_EMPTY
    jz    .WaitForTxEmpty      ;   No,  keep checking the status
    mov   A, [X+0]             ;   Reload next value into A
```

```
    call  SPIM_SendTxData      ;   Yes, load TX with the next byte, ...
    inc   X                    ;        Advance the pointer, ...
    jmp   .TxNextByteLoop      ;        and repeat until done.
.Finish:                       ;   Transmit complete!
    call  SPIM_bReadStatus     ;   Buffer empty for last time? ...
    and   A, SPIM_SPIM_TX_BUFFER_EMPTY
    jz    .Finish              ;   No,  keep checking the status
.WaitForTxComplete:
    call  SPIM_bReadStatus     ;   Last byte transmission complete? ...
    and   A, SPIM_SPIM_SPI_COMPLETE
    jz    .WaitForTxComplete   ;   No,  keep checking the status
    ; [...]                    ;   (Reset SlaveSelect signal back high)
.AllDone:
    ; Endless loop
    jmp   .AllDone
```

設定レジスタ

下記に、このユーザーモジュールの構成に使用するデジタルコミュニケーションタイプ A PSoC ブロックレジスタに関する説明を示します。パラメータ化された記号のみ説明されています。

Table 3. ブロック SPIM、レジスタ : 関数

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	1	1	0

このレジスタは、SPIM ユーザ モジュールになるデジタルコミュニケーション タイプ 「A」ブロックの特徴を定義します。

Table 4. ブロック SPIM、レジスタ : Input (入力)

ビット	7	6	5	4	3	2	1	0
値	MISO				Clock (クロック)			

MISO はマスタインスレーブアウト入力信号です。クロックは、SPIM タイミングを管理するために選択されたクロックです。両方とも、パラメータ選択の際、デバイス エディタを使用してセットされます。

Table 5. ブロック SPIM、レジスタ : Output (出力)

ビット	7	6	5	4	3	2	1	0
値	0	0	SCLK			MOSI		

MOSI はマスタアウトスレーブイン入力信号です。SCLK は SPI クロック信号です。両方とも、パラメータ選択の際、デバイス エディタを使用してセットされます。

Table 6. ブロック SPIM、シフト レジスタ : DR0

ビット	7	6	5	4	3	2	1	0
値	シフト レジスタ							

シフト レジスタは SPI シフト レジスタです。

Table 7. ブロック SPIM、送信データ バッファレジスタ : DR1

ビット	7	6	5	4	3	2	1	0
値	送信バッファレジスタ							

送信バッファレジスタ : このバッファに書き込まれるデータは、PSoC ブロックが有効化されると、シフト レジスタに転送されます。

Table 8. ブロック SPIM、受信データ バッファレジスタ : DR2

ビット	7	6	5	4	3	2	1	0
値	受信バッファレジスタ							

受信バッファレジスタ : シフト レジスタに受信されたデータは、SPI 送信サイクルの完了後、このレジスタに転送されます。

Table 9. ブロック SPM、制御レジスタ : CR0

ビット	7	6	5	4	3	2	1	0
値	LSB 優先	受信オーバーランエラー	SPI_COMPLETE	送信バッファ Empty (空)	受信バッファ Full (フル)	クロック位相	クロック極性	SPIM イネーブル

LSB 優先は、LSB ビットが最初に送信されることを指定します。

受信オーバラン エラーは、前回の受信データバイトが次のバイトが受信されるまで読み取られないことを示すフラグです。

SPI 完了は、完全な SPI 送受信サイクルが完了したことを示すフラグです。

送信バッファ Empty (空) は、送信バッファが空であることを示すフラグです。

受信バッファ Full (フル) は、シフト レジスタからデータの 1 バイトが受信されたことを示すフラグです。

クロックフェーズは、SCLK 信号のフェーズを示し、SPI モードを定義するパラメータの 1 つです。

クロック極性は、SCLK 信号の極性を示し、SPI モードを定義するパラメータの 1 つです。

SPIM イネーブルは、SPIM PSoC ブロックがセットされている場合にそれを有効にします。

バージョン ヒストリー

バージョン	考案者	説明
2.6	TDU	クロックの説明に関する更新：ブロックで外部デジタル クロックを使用している場合は、最高の精度およびスリープ動作を得るため、列入力同期がオフになります。

Note PSoC Designer 5.1 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ています。このセクションでは、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。

Copyright © 2002-2011 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.