SPI マスタ データシート SPIM V 2.5

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

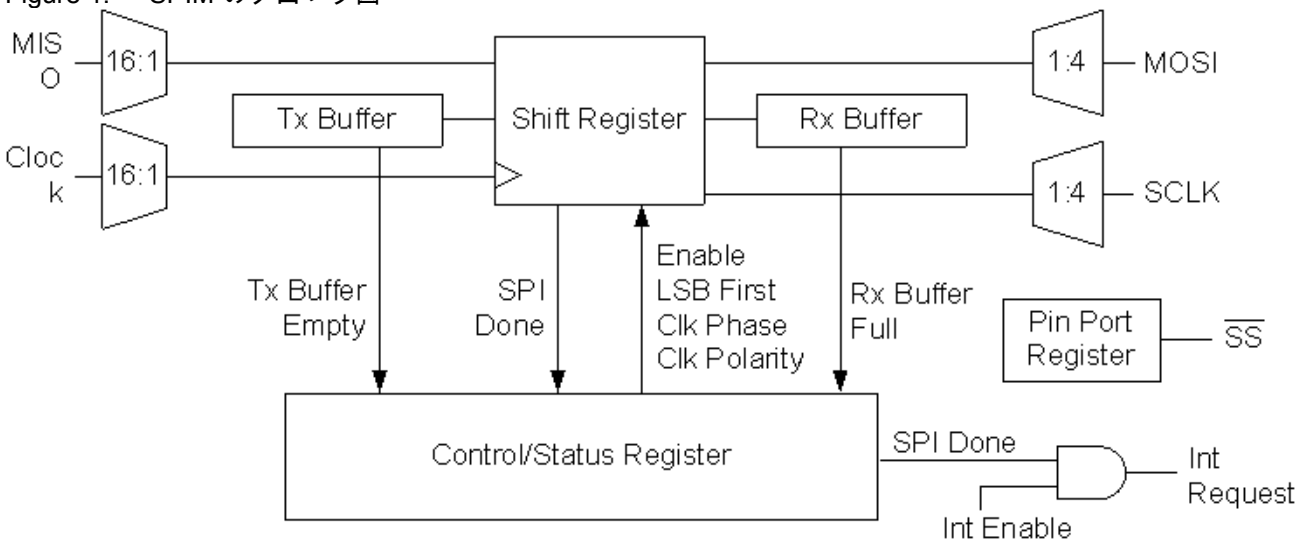
リソース	PSoC® ブロック			API メモリ (バイト)		ピン (外部入出力ごと)
	CapSense®	I2C/SPI	タイマ	Flash	RAM	
CY8C20x34、CY8C20x24、CY8C20x66、CY8C20x36、CY8C20336AN、CY8C20436AN、CY8C20636AN、CY8C20x46、CY8C20x96、CY7C604xx、CY7C643xx、CYONS2010、CYONS2011、CYONSFN2051、CYONSFN2053、CYONSFN2061、CYONSFN2151、CYONSFN2161、CYONSFN2162、CY8CTST200、CY8CTMG2xx						
		X		27	0	3 - 4

特徴と概要

- シリアルペリフェラルインターコネクト (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、3 をサポート。
- クロックと MISO の選択可能な入力ソース。
- MOSI と SCLK の選択可能な出力ルーティング。
- SPI 完了状態でのプログラマブルな割り込み。
- SPI スレーブ デバイスは個別に選択可能。

SPIM ユーザ モジュールは、シリアル周辺装置相互接続マスタです。これは、全二重同期の 8-bit データ転送を実行します。ほとんどの SPI クロック モードに対応するよう、SCLK フェーズ、SCLK 極性、LSB First を利用できます。ユーザが提供するソフトウェアによって制御されるスレーブ選択信号を使って、1 台または複数台の SPI スレーブ デバイスを制御することができます。SPIM PSoC ブロックでは、入力と出力信号用のルーティングは選択可能で、プログラム可能な割り込み駆動による制御も可能です。

Figure 1. SPIM のブロック図



機能説明

SPIM は、シリアル周辺装置相互接続マスタを実装するユーザ モジュールです。I2C/SPI PSoC ブロックの送信バッファ、受信バッファ、コントロール、コンフィグレーションレジスタ、データ シフト用のデータ レジスタ、1 つまたは複数のピン ポート 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 モードは次のように定義されます。

Table 1. SPI モード

モード	SCLK エッジ実行データラッチ	Clock Polarity	注
0	立ち上がり	非反転	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	
2	立ち下がり	非反転	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	

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 デバイスを正しく有効にするために、アクティブなロー スレーブ選択信号 ~SS はユーザが提供するソフトウェア ルーチンで設定する必要があります。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、同時に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 レートは、入力クロックを 2 で除算した値になります。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 信号とは、マスタからスレーブ SPI プロトコル互換デバイスにデータを送信する、マスタ アウト スレーブ インのデータ信号です。MISO 信号とは、スレーブ SPI デバイスからこのユーザ モジュールにデータを送信する、マスタ イン スレーブ アウトのデータ信号です。

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、マスタによってのみ実行されるレスポンス 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。

送信されたデータ 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。送信オーバラン状態を防止するには、送信バッファ レジスタにバイトを書き込む前に、バッファの Empty (空) ステータス ビットを確認してください。送信する次のデータ バイトは、この時に送信バッファ レジスタに書き込まれます (プリロード)。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 モードの構成に基づきます。

すべてのビットが送信され同時に受信されると、受信されたデータがシフト 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 レジスタに転送されます。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。割り込みを有効にしている場合は、SPI 完了ステータス ビットにより割り込みがトリガされます。

受信バッファ レジスタからのデータ バイト取得に SPI 完了割り込み条件を使用しない場合は、制御レジスタをポーリングして、受信バッファの Full (フル) ステータス ビットをモニタしてください。次のデータ バイトが完全に受信されるか、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、受信したデータを受信バッファ レジスタから読み取ってください。

保留中のデータ バイトが送信バッファ レジスタにプリロードされた場合は、このバイトで SPI 状態マシンが再起動し、送信が直ちに開始されます。

各データ バイトが送信された後で、スレーブの選択信号を制御するには、非割り込みと割り込みレベルという 2 つの方法があります。どちらの方法を使用するかは、SPI クロック モード、バイト トランスポート プロトコル、送信されるデータの必要なビット レート速度に応じます。SPI クロックモード 0 および 1 では、データの次のバイトが送信される前に、スレーブ選択をオフにして再度オンにする (トグルする) 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 トランスポート プロトコルでは、スレーブ選択のトグルを必要とする場合と、必要としない場合があります。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。

非割り込みレベルでのスレーブ選択信号の制御は、SPI データが送信される間に、SPI 完了ステータス ビットをモニタリングまたはサンプリングする、専用のマイクロプロセッサが必要になります。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 ビットをモニタリングするオーバヘッドによって、マイクロプロセッサで他の必要な動作が実行できなくなるような遅いビット レートでは、スレーブ選択信号の制御は割り込みレベルで行います。最低割り込みレイテンシは 833 ns (クロック レートが 24 MHz の場合) であり、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。

送信するデータとともに送信バッファがプリロードされる複数バイトのデータ転送では、注意が必要です。ただし、選択したスレーブ信号を管理するのに必要なバイト間処理が最小またはまったく必要ない場合や、複数バイト転送が同じ SPI スレーブ デバイスに対して実行される場合は、1 MHz 以上のビット転送レートを達成できる可能性があります。

最後のデータが送信されたら、SPI 完了ビットをモニタして、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します。これによって、マスタおよびスレーブの SPI デバイス間で、クロック信号をすべて完了したことを保証します。

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	標準値	制限	単位
F_{max}	最大ビット レート	--	12	MHz

配置

SPIM は 1 つの PSoC ブロックにマッピングされ、I2C/SPI ブロックに配置することができます。

SPI スレーブ デバイスを制御するために、スレーブ選択信号が使用するピン ポート レジスタを予約してください。これらのポート ビットは、標準の CPU ポート ピンとして構成します。

タイミング

クロック レートは、必要なビット レートの 2 倍に設定する必要があります。

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。

パラメータおよびリソース

Interrupt Mode (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た直後に、割り込みを生成します。2 つ目のオプション 「TxComplete」を選択すると、最後のビットがシフト レジスタからシフトアウトされるまで、割り込みが延期されます。この 2 つ目のオプションは、データ バイトが完全に送信された時を知る必要がある場合に便利です。最初のオプション「TxRegEmpty」は、トランスミッタの出力を最大にしたい場合に役立ちます。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。

Clock Select (クロック選択)

このパラメータは、SPI マスタの動作周波数を設定します。これは、[Global Resources (グローバル リソース)] の SysClk 設定に基づきます。例えば、[Global Resources (グローバル リソース)] の電力設定で 「3.3V/12MHz」または 「5.0V/12MHz」を選択して SysClk 周波数を 12 MHz に設定し、[User Module Parameter (ユーザ モジュール パラメータ)] のクロック選択で 「SysClk/4」が選択されている場合は、SPI マスタのビットレートは 3MBits/sec になります。[Global Resources (グローバル リソース)] と [User Module Parameter (ユーザ モジュール パラメータ)] は、デバイス エディタの 「Interconnect View (相互接続ビュー)」にあります。

[Clock Select (クロック選択)] のオプションは次の通りです。

SysClk/2

SysClk/4

SysClk/8

SysClk/16

SysClk/32

SysClk/64

SysClk/128

SysClk/256

Data Order (データの順序)

このオプションは、シリアルデータをシフトアウトする方法を LSb First または MSb First のいずれかに設定します。

Interrupt Generation Control (割り込み生成制御)

以下のパラメータは、**[Enable interrupt generation control** (割り込み生成制御をイネーブルにする)] チェックボックスが、PSoC Designer でチェックされている場合にのみアクセス可能です。これは **[Project (プロジェクト)] - > [Settings (設定)] - > [Chip Editor (チップ エディタ)]** にあります。

IntDispatchMode

IntDispatchMode パラメータを使用して、同一ブロック内の異なるオーバーレイ内に存在する複数のユーザ モジュールで共用されている割り込みについて、割り込み要求をどのように処理するかを指定します。「ActiveStatus」を選択すると、共用割り込み要求を処理する前に、ファームウェアはどのオーバーレイがアクティブかをテストします。このテストは、共用割り込みが要求されるたびに行われます。このためにレイテンシが発生し、共用割り込み要求を処理する非決定性のプロセスも生じますが、RAM は不要です。「OffsetPreCalc」を選択すると、オーバーレイが最初にロードされたときにだけ、ファームウェアは共用割り込み要求のソースを計算します。この計算によって割り込みレイテンシは減少し、共用割り込み要求を処理する決定性のプロセスが生じますが、RAM のバイトを消費します。

アプリケーション プログラミング インタフェース

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、より高度なレベルでモジュールを処理できるようにユーザ モジュールの一部として提供されています。このセクションでは、「include」ファイルによって提供される、各関数に対するインタフェースおよび定数を示します。

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。呼び出し後に A と X の値が必要な場合は、呼び出し元関数で A と X の値を保持してください。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。

SPIM で提供されている API 関数のリストは次の通りです。

SPIM_Start

説明：

SPI インタ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を High にアサートし、接続している SPI スレーブ デバイスの選択を解除します。これは、ユーザが提供するルーチンで行ってください。

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。アキュムレーターを通過します。C およびアセンブリで用意されたシンボル名、およびそれらに関連付けられた値は、以下の表で示します。シンボル名は、SPI インタフェースを構成するよう 「OR」を使って結合することができます。また、以下の表に示されているシンボル名の前に、ユーザ モジュールのインスタンス名が追加されます。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 モードのシンボル名は 「SPIM1_SPIM_MODE_0.」となります。

シンボル名	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

注意事項：

A および X レジスタがこの関数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。

この関数の呼び出しを行った後で、すべてのスレーブ選択信号を High にアサートし、接続されている SPI スレーブ デバイスをすべて無効にします。これは、ユーザが提供するルーチンで行ってください。

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

注意事項：

A および X レジスタがこの関数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

注意事項：

A および X レジスタがこの関数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

注意事項：

A および X レジスタがこの関数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。

この関数を呼び出す前に、指定した SPI スレーブ デバイス信号を Low にアサートします。これは、ユーザが提供するルーチンで行ってください。

C プロトタイプ：

```
BOOL  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 デバイスに送信するデータ。累算器を通過します。

戻り値：

なし

注意事項：

A および X レジスタがこの関数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。このルーチンを呼び出す前に、受信バッファの Full (フル) フラグをチェックして、データ バイトが受信されたことを確認してください。

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、累算器に戻されたデータ バイト。

注意事項：

A および X レジスタがこの関数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_SPI_COMPLETE | 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。

特定の状態条件をテストするには、定義済みマスクを使用してください。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。また、以下の表に示されているシンボル名の前に、ユーザ モジュールのインスタンス名が追加されます。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 マスクのシンボル名は 「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

注意事項：

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 レジスタがこの関数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

以下の C およびアセンブリのサンプルコードでは、SPI マスタは、RAM に保存されているゼロで終端された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。I2C/SPI ブロックは、その送信レジスタからシフト レジスタに転送して送信を開始します。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ロードされた直後に戻り値を返します。一方、最終バイトの次のバイトは、MOSI ラインでまだシフトアウトされています。送信レジスタに値をロードする前に、ステータスが常にチェックされるため、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はありません。

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

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

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++;
    }
}
```

同等のアセンブリ言語コードを以下に示します。

```
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, 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

このユーザ モジュールを構成するのに使用する I2C/SPI PSoC ブロック レジスタに関する説明を以下に示します。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、構成レジスタ

ビット	7	6	5	4	3	2	1	0
値	Clock Sel			Bypass	SS_	SS_EN_	Int Sel	Slave

Clock Sel - クロック選択。これらのビットは、SPI マスタの動作周波数を設定します。

000b - SysClk / 2

001b - SysClk / 4

010b - SysClk / 8

011b - SysClk / 16

100b - SysClk / 32

101b - SysClk / 64

110b - SysClk / 128

111b - SysClk / 256

Bypass - バイパス同期。このビットは、入力が SYSCLK に同期されるかどうかを決定します。

SS_ - スレーブ選択。このビットは、SS_EN_ 信号がアサートされる場合 (SS_EN_ = 0) に、SS_ 信号の論理値を決定します。

SS_EN_ - スレーブ選択イネーブル。このアクティブな下位ビットは、スレーブ選択 (SS_) 信号が内部的に駆動されるかどうかを決定します。内部的に駆動される場合は、その口ジック レベルは、SS_ ビットによって決定されます。また外部的に駆動される場合は、その口ジック レベルは、外部ピンによって決定されます。

Int Sel - 割り込み選択。このビットは、TX Reg Empty 状態または SPI 完了状態のどちらで割り込みを生成するかを選択します。

Slave - このビットは、ブロックがマスタまたはスレーブのどちらとして機能するかを決定します。

Table 4. ブロック SPIM、送信データ バッファ レジスタ

ビット	7	6	5	4	3	2	1	0
値	送信バッファ登録							

送信バッファ レジスタ : PSoC ブロックを有効にした場合、このバッファに書き込まれるデータは、シフト レジスタに転送されます。

Table 5. ブロック SPIM、受信データ バッファ レジスタ

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 6. ブロック SPIM、制御レジスタ

ビット	7	6	5	4	3	2	1	0
値	LSB First	RX Overrun Error	SPI_ Complete	TX Buffer Empty	RX Buffer Full	Clock Phase	Clock Polarity	Enable

LSB First (LSB 優先) - LSB ビットが最初に送信されるよう指定します。

RX Overrun Error (RX オーバラン エラー) - 次のバイトを受信する前に、すでに受信したデータ バイトが読み込まれなかったことを示すフラグ。

SPI Done (SPI 完了) - 完全な SPI 送受信周期が完了したことを示すフラグ。

Tx Buffer Empty - (送信バッファ - 空) - 送信バッファが空であることを示すフラグ。

Rx Buffer Full (受信バッファ フル) - データ バイトがシフト レジスタから受信されたことを示すフラグ。

Clock Phase (クロック位相) - SCLK 信号の位相を示します。SPI モードを定義するパラメータの 1 つです。

Clock Polarity (クロック極性) - SCLK 信号の極性を示します。SPI モードを定義するパラメータの 1 つです。

Enable (イネーブル) - これが設定されている場合は、SPI PSoC ブロックを有効にします。

バージョン履歴

バージョン	著者	説明
2.5	DHA	P1[0] が、PSoC デバイス CY7C64315 および CY7C64316 で SPI CLK ピンとして設定されました。

Note PSoC Designer 5.1 では、すべてのユーザ モジュール Data Sheet にバージョン履歴を導入しました。このセクションには、ユーザ モジュールの現行バージョンと旧バージョンとの相違に関する高度な説明が記載されています。

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.