SPI スレーブ データシート SPIS V 2.60

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

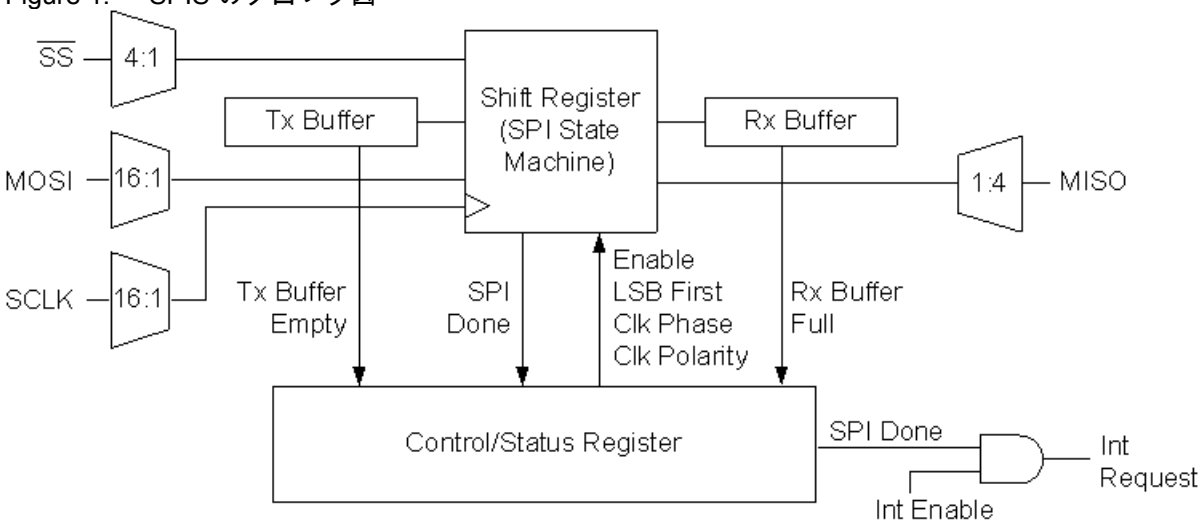
リソース	PSoC® ブロック			API メモリ （バイト数）		ピン （外部入出力ごと）
	デジタル	アナログ CT	アナログ SC	フラッシュ	RAM	
CY8C29/27/24/22/21, CY8C23x33, CY7C603xx, CY7C64215, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx, CY8C21x12						
	1	0	0	43	0	4
CY8C26/25xxx	1	0	0	37	0	4

特性および概要

- シリアルペリフェラル相互接続（SPI）スレーブプロトコルをサポート
- プロトコルモード 0、1、2、3 をサポート
- MOSI、SCLK、~SS 用の選択可能な入力ソース
- MISO 用の選択可能な出力ルーティング
- SPI 完了状態へのプログラム可能な割り込み
- SS はファームウェア制御も可能（PSoC デバイスの CY8C26/25xxx ファミリを除く）

SPIS ユーザ モジュールは、シリアル周辺装置相互接続スレーブで、全二重同時 8-bit データ転送を行います。殆どの SPI プロトコルに対応できるようように、SCLK フェーズ、SCLK 極性、LSB First を指定することができます。SPIS 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起動によるコントロールもあります。アプリケーション プログラミング インタフェース（API）ファームウェアは、アセンブリ言語または C 言語によるアプリケーションソフトウェア用の高レベルのプログラミングインターフェースを提供します。

Figure 1. SPIS のブロック図



機能説明

SPIS は、シリアル周辺装置相互接続スレーブを実装するユーザ モジュールで、デジタルコミュニケーション形態の PSoC ブロック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、デバイス エディタ、ユーザーモジュール 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ーフェース (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 優先と SPI 送受信プロトコルモードの設定を含みます。SPI モード 1、2、3 がサポートされています。SPI マスタと SPI スレーブは両方が、正しく通信するために、同じモードとビット構成に設定されていなければなりません。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	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におけるデータ変化
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 立ち上がりエッジにおけるデータ変化
3	立ち下がり	反転済み	

SCLK 入力信号は、SPI マスタによって生成される SPI 送受信クロックで、送受信データのビット レートを定義します。

MOSI 入力信号は、SPI マスタからデータを受信するマスタアウトスレーブイン データ信号です。

MISO 出力信号は、シフト レジスタから SPI マスタデバイスに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。通常、複数のスレーブの MISO が一つにまとめられています。各スレーブは、スレーブ選択信号がアサートされるまで、それぞれの MISO 信号をトライステートにします。このユーザ モジュールは、MISO 出力信号をトライステートにしません。この機能は、必要な場合に簡単に追加できます。

SPIS ハードウェアは、MOSI 信号上のマスタ SPI デバイスからデータを受信し、同時に、MISO 信号上の SPI マスタデバイスにデータを送信します。同じ SCLK 信号は、マスタおよび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。マスタは、スレーブ選択（~SS）入力信号を Low にアサートして、特定の SPIS デバイスをアクティブな通信用に有効にします。

選択されたスレーブデバイスがコマンド受け取りやデータ受信の準備ができているかを判断するのは、マスタの役割です。

SPI イネーブル ビットが API ルーチンを使用して制御レジスタで設定されている場合、SPIS ユーザ モジュールは稼動できます。

SPI マスタに送信されるデータは、送信バッファレジスタに書き込まれます。これによって、送信バッファ Empty（空）ステータスビットがクリアされます。

スレーブ選択信号の下りエッジでは、データは送信バッファレジスタからシフト レジスタに転送されます。次に、送信されるデータタイプの最初のビットが、MISO 出力信号でアサートされます。このとき、別の送信データタイプを送信バッファレジスタに書き込むことができます。現在のバイトの送信完了後、このデータは SPI マスタへの送信の準備ができます。

各 SCLK 入力信号のアサートで、データはシフト レジスタから MISO 出力に、MOSI 入力からシフト レジスタに、同時にシフトします。SCLK、MOSI、MISO 信号のタイミングは、SPI モード構成によります。

すべてのビットが送受信されると、受信データはシフト レジスタから受信バッファレジスタに転送され、送信バッファがシフト レジスタに送信されます。受信バッファ Full（フル）と SPI 完了 ステータスビットが設定されます。割り込みが有効のときは、SPI 完了 ステータスビットがトリガとなります。この割り込みは、データのバイトが受信されたこと、またはバイトが送信されたことを、ソフトウェアにアラート通知するために使用されます。

保留中のデータバイトが送信バッファレジスタに読み込み中である場合、このバイトは、マスタが次のトランザクションを行うときに送信する準備ができます。

SPI 完了割り込み条件が受信バッファレジスタからデータバイトを受信するために使用されない場合、制御レジスタは受信バッファ Full（フル）ステータスビットをモニタするためにポーリングされます。次のデータバイトが完全に受信される、またはオーバラン 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受信データを読み出さなければなりません。

SPIS ユーザ モジュールを無効にするタイミングを特定するために、SPI 完了ビットをモニタします。これによって、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。

割り込みが使用される場合、次の割り込みが認識されるためには、一回一回の割り込みの後、SPIS 状態レジスタがクリアされなければなりません。状態レジスタを読み出すと、割り込みコントローラによって認識される内部信号がクリアされます。この信号が High のままでいると、次の SPIS 割り込みがマスクされます。TxComplete と RxComplete 割り込みはこれに該当しますが、TxRegEmpty と RxRegEmpty 割り込みは該当しません。レジスタの読み込みやクリアには、アセンブリ言語 **MOV** または **TST** オプコードを使用できます。

タイミング

SPIS 転送の通常のタイミングは、次の図に示すとおりです。SPIS 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 の通常の SPIS タイミング

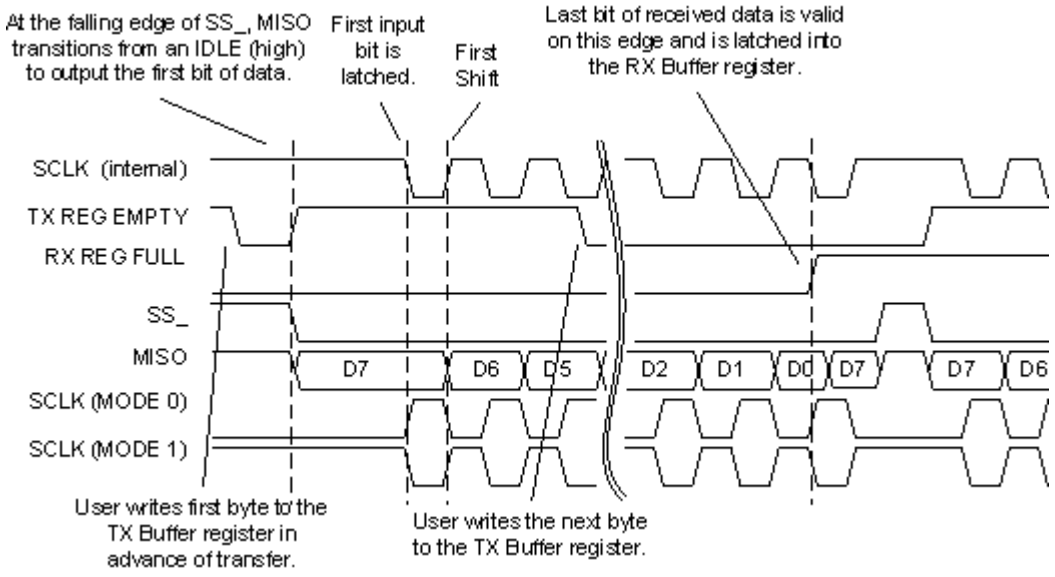
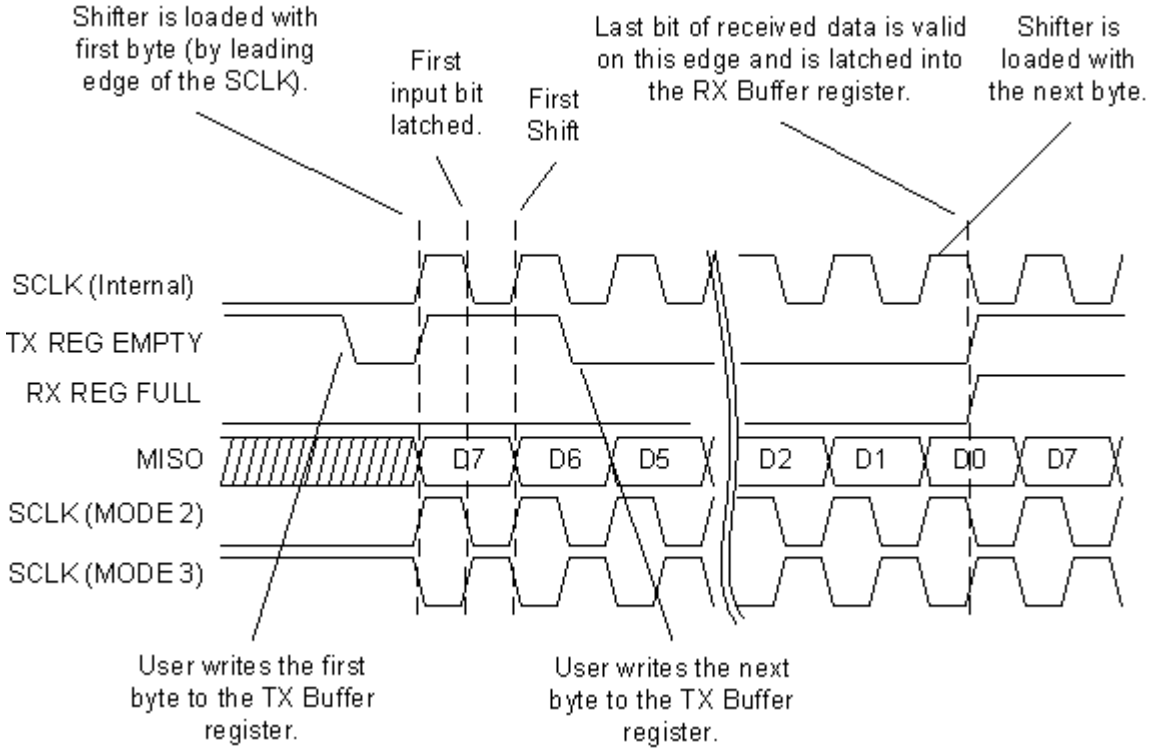


Figure 3.　モード 2 と 3 の通常の SPIS タイミング



DC 電気的特性と AC 電気的特性

Table 2. SPIS DC 電気的特性と AC 電気的特性

パラメータ	条件および注記	標準値	制限	単位
F_{InMax}	最大入力 SCLK 周波数	4.1	8.2	MHz

配置

SPIS は単一の PSoC ブロックにマッピングされます。ブロック名は、PSoC デザイナ デバイス エディタの SPIS です。これはデジタルコミュニケーションブロックの任意の部分に配置されることがあります。

パラメータおよびリソース

SCLK

SPIS は、SPI マスタ生成による SCLK 信号によってクロック制御されます。この信号は使用可能なソースの一つからルーティングできます。高、低、グローバル I/O バス、アナログコンパレータバス、またはその他の PSoC ブロックを、SCLK 入力供給として指定できます。このクロックは、有効ビット転送レートを指定します。デフォルトでは、SCLK は SysClk に同期されます。SCLK が非同期または SysClk*2 に同期されている場合、出力レジスタの ClockSync ビットフィールドへの直接書き込みを使用します。利用可能なクロックとクロックレートは、PSoC デバイスによって異なります。

MOSI

マスタアウトスレーブイン入力信号は、16 の可能なソースの 1 つからルーティングできます。High、Low、グローバル I/O バス、アナログコンパレータバス、またはその他の PSoC ブロックを、MOSI 入力供給として指定できます。

~SS

スレーブ選択入力信号は、4 つの可能なグローバル入力の 1 つからルーティングできます。さらに、~SS はファームウェアで制御することも可能です。SW_SlaveSelect を使用した場合、スレーブ選択信号はユーザ モジュールの起動時にアサートされます。これはハードウェア SS を使用しているときはアサートされません。

MISO

マスタインスレーブアウト出力信号は、グローバル出力バスの 1 つにルーティングされます。次に、グローバル出力バスは外部ピンまたは別の PSoC ブロックに接続して、さらに処理することができます。

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を生成するタイミングを指定します。「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、割り込みが生成されます。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、割り込みは延期されます。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み込むことができます。

InvertMOSI

このパラメータは、MOSI 入力の反転を可能にします。

割り込み生成制御

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)] > > [Setting(設定)] > > [Chip Editor(チップ エディタ)] を選択すると使用できます。**「Interrupt Generation Control(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。

- 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 パラメータを使用して、同一ブロック内の異なるオーバレイ内に存在する複数のユーザ モジュールで共用している割り込みについて、割り込みをどのように処理するかを指定します。「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付加され、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、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、これは 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) ルーチンは設計者が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。

Note ここでは、全てのユーザ モジュール API 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。API をコールした後でも A と X の全値を保持したいときは、API をコールするファンクションで AtoX の値を保持する必要があります。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を処理します。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しなければなりません。一部のユーザーモジュール 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、SPIS が供給する API 関数を挙げます。

SPIS_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 SPIS 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 High 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  SPIS_Start(BYTE bConfiguration)
```

アセンブラ

```
mov   A,SPIS_MODE_2 | SPIS_LSB_FIRST
lcall  SPIS_Start
```

パラメータ

bConfiguration: SPI モードと LSB First 構成を指定する 1 バイト。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 「SPIS1」という名前をつけた場合、第 1 モードの記号名は 「SPIS1_SPIM_MODE_0.」となります。

記号名	値
SPIS_MODE_0	0x00
SPIS_MODE_1	0x02
SPIS_MODE_2	0x04
SPIS_MODE_3	0x06
SPIS_LSB_FIRST	0X80
SPIS_MSB_FIRST	0X00

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。

SPIS_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、SPIS モジュールを無効にします。

C プロトタイプ

```
void SPIS_Stop(void)
```

アセンブラ

```
lcall  SPIS_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。

SPIS_EnableInt

説明

SPI 完了状態における SPIS 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 SPIS_EnableInt(void)
```

アセンブラ

```
lcall  SPIS_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。

SPIS_DisableInt

説明

SPI 完了状態における SPIS 割り込みを無効にします。

C プロトタイプ

```
void SPIS_DisableInt(void)
```

アセンブラ

```
lcall  SPIS_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。

SPIS_SetupTxData

説明

SPI マスタに送信されるデータバイトを送信バッファレジスタに書き込みます。

C プロトタイプ

```
void SPIS_SetupTxData(BYTE bTxData)
```

アセンブラ

```
mov   A, bTxData
lcall  SPIS_SetupTxData
```

パラメータ

bTxData: SPI マスタデバイスに送信され、累算器へと送られるデータです。

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。

SPIS_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、このルーチンを呼び出す前に受信バッファ Full（フル）のフラグがチェックされます。

C プロトタイプ

```
BYTE SPIS_bReadRxData(void)
```

アセンブラ

```
lcall  SPIS_bReadRxData
mov   bRxData, A
```

パラメータ

なし

戻り値

スレーブ SPI から受信し、Accumulator を通して返されるデータバイト。

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。

SPIS_bReadStatus

説明

現在の SPIS 制御 / 状態レジスタを読み取り、これを返します。

C プロトタイプ

```
BYTE SPIS_bReadStatus(void)
```

アセンブラ

```
lcall  SPIS_bReadStatus
and    A, SPIS_DONE | RX_BUFFER_FULL
jnz    SPI_COMPLETE_GET_RX_DATA
```

パラメータ

なし

戻り値

状態バイト読み取り結果を返し、Accumulator を通して返されます。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。

SPIM 状態マスク	値
SPI_COMPLETE	0x20
RX_OVERRUN_ERROR	0x40
BUFFER_EMPTY	0x10
RX_BUFFER_FULL	0x08

副作用

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。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。

SPIS_DisableSS

説明

外部 ~SS 信号が不要なとき、ファームウェアを用いてアクティブローのスレーブ選択信号 (~SS) を High にセットします。この関数を使用するには、「スレーブ選択入力」という SPI パラメータが、row 入力の 1 つではなく、値 「SW_SlaveSelect」に設定されていなければなりません。外部信号が接続されている場合、この関数は無効です。

C プロトタイプ

```
void SPIS_DisableSS(void)
```

アセンブラ

```
lcall SPIS_DisableSS
```

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。

SPIS_EnableSS

説明

外部 ~SS 信号が不要なとき、ファームウェアを用いてアクティブローのスレーブ選択信号 (~SS) を Low にセットします。この関数を使用するには、「スレーブ選択入力」という SPI パラメータが、row 入力の 1 つではなく、値 「SW_SlaveSelect」に設定されていなければなりません。外部信号が接続されている場合、この関数は無効です。

C プロトタイプ

```
void SPIS_EnableSS(void)
```

アセンブラ

```
lcall SPIS_EnableSS
```

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。

ファームウェア ソースコードの例

この例は、SPI ループ バックの作り方を示しています。これは SPI マスタから受信したデータを反映しています。エラーが検知されると Nak が送信されます。

```
;***************************************************************
;  Loop Back:
;
;  This code sample illustrates how to setup a SPI Slave loop back.
;  Data received is echoed back to the SPI master.
;
;  This sample is written to be performed in the non-interrupt
;  processing.  This code could easily be written to be
;  interrupt driven.
;
;  It is assumed that the SPI Slave User Module is setup properly
;  and started.
;
;***************************************************************
include  "SPIS.inc"
export   LoopBack

LoopBack:
        ; wait for data to be received
    call SPIS_bReadStatus
    and A, SPIS_SPIS_SPI_COMPLETE
    jz LoopBack
        ; read the data from the receiver
    call SPIS_bReadRxData
        ; setup to transmit the response data
    call SPIS_SetupTxData
        ; go wait for next byte
    jmp LoopBack
```

C で書かれた同じコードは以下のようになります。

```
#include  "SPIS.h"

void  LoopBack(void)
{
    BYTE  bData;
    while(1)
    {
        /* wait for data to be received */
        while( !( SPIS_bReadStatus() & SPIS_SPIS_SPI_COMPLETE ) );

        /* read the received data */
        bData = SPIS_bReadRxData();


        /* setup to transmit the response data */
        SPIS_SetupTxData(bData);
    }
}
```

コンフィグレーション 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 SPIS: レジスタ関数

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	1	1	1	0

このレジスタは、SPIS 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 SPIS: レジスタ入力

ビット	7	6	5	4	3	2	1	0
値	MOSI				SCLK			

MOSI はマスタアウトスレーブイン入力信号です。SCLK は SPI マスタからのクロック信号です。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 5. ブロック SPIS: レジスタ出力

ビット	7	6	5	4	3	2	1	0
値	0	ClockSync (クロック同期)		~SS		MISO		

~SS はスレーブ選択入力信号です。MISO はマスタインスレーブアウト出力信号です。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 SPIS: 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 SPIS: 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たデータは、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 SPIS: 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 SPIS: 制御レジスタ CR0

ビット	7	6	5	4	3	2	1	0
値	LSB First	受信オーバーランエラー	SPI 完了	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	SPIS 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。

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

バージョン ヒストリー

バージョン	著者	説明
2.5	DHA	更新されたスレーブ選択パラメータ。
2.60	DHA	CY8C21x12 デバイスのサポートを追加。

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 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.