可変長 SPI スレーブデータシート SPISVL V 1.0

Copyright © 2009-2011 Cypress Semiconductor Corporation. All Rights Reserved.

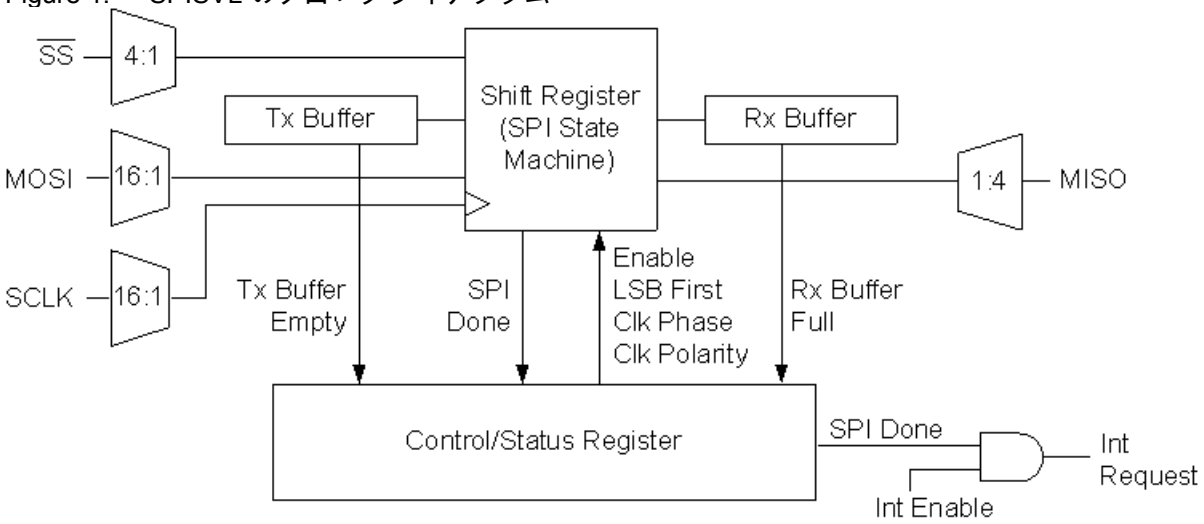
リソース	PSoC® ブロック			API メモリ （バイト数）		ピン （外部入出力ごと）
	デジタル	アナログ CT	アナログ SC	Flash	RAM	
CY8C21x45, CY8C22x45, CY8C28x45, CY8C28xxx						
	2	0	0	142	1	4

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 Complete （完了）または送信レジスタ Empty （空）状態でのプログラミング可能な割り込み
- SS はファームウェアで管理可能
- 選択可能なデータ長 –9 ~ 16 ビット

SPISVL ユーザ モジュールは、可変データ長として構成できるシリアル周辺装置相互接続スレーブです。9 ~ 16 ビットの間の任意のデータ長で全二重の同期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に、SCLK フェーズ、SCLK 極性、LSB First を指定することができます。SPISVL 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. SPISVL のブロック ダイアグラム



機能説明

SPISVL は、シリアルペリフェラル相互接続スレーブを実装するユーザ モジュールで、9-16 ビットデータをサポートするため、2 つの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 レジスタのペアを使用します。

Control0、Control1 レジスタは、SPIM デバイス エディタ、SPISVL 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 優先構成の設定、SPI 送受信クロックモード、データ長の設定が含まれます。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	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	

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。

SPISVL 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 LSB ブロックの Control0 レジスタで設定されている場合、SPISVL 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、LSB ブロックの送信バッファ Empty（空）ステータスビットがクリアされます。

スレーブ選択信号の立ち下がりエッジでは、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、送信されるデータワードの最初のビットが、MISO 出力信号でアサートされます。このとき、別の送信データワードを送信バッファレジスタに書き込むことができます。現在のワードの送信完了後、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 モード構成に基づいています。

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 ステータスビットによりトリガされます。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。

保留中のデータワードが送信バッファレジスタに読み込み中の場合、このワードは、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整ったことを示します。

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、Control0 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。受信データは、次のデータワード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。

SPISVL 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、SPISVL 状態レジスタが読み込ま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。この信号が高の状態のままだと、次の SPISVL 割り込みがマスクされます。

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	典型値	制限	単位
F_{max}	最大受信 / 送信周波数	--	12	MHz

配置

SPISVL は、ここでアーキテクチャアンサンブルとして見なされる 2 つの PSoC ブロックにマッピングします。SPISVL は、同じ行のデジタルコミュニケーションブロックの任意のペアに配置されます。

パラメータおよびリソース

SCLK

SPISVL 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 15 の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。

[+] Feedback

MOSI

マスタアウトスレーブイン入力信号は、15 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。

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。

割り込みタイプ

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「SPI Complete」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を確認したいときに役立ちます。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。

InvertMOSI

このパラメータは、MOSI 入力の反転を可能にします。

DataLength

このパラメータは、SPI 通信プロトコルのデータ長を設定します。

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ている場合のみです。これは **[Project(プロジェクト)] > > [Setting(設定)] > > [Chip Editor(チップ エディタ)]** でアクセスできます。「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、ファームウェアは共用されている割り込み要求を処理する前に、どのオーバ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、ファームウェアはオーバレイが最初にロードされたときだけ共用割り込みの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、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、「include」ファイルによって提供される、各関数のインタフェースおよび関連する定数を示します。

Note ** ここでは、全てのユーザー モジュールの API では、A と X レジスタの値は、API 関数を呼び出すことによって変更することができます。関数を呼び出す場合、呼出し後に 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を処理します。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しなければなりません。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。

ユーザ モジュールのインスタンス名は、定数名や関数名中の「SPISVL」という言葉で置き換えられます。例えば、ユーザ モジュールの配置時に「SPISVL1」という名前をつけた場合、第 1 モードの記号名は「SPISVL_1_MODE_0」、SPISVL_Start 関数は SPISVL_1_Start. として利用可能となります。

次に、SPISVL による API 関数のリストを挙げます。

SPISVL_Start

説明

SPI インターフェースのモード構成を設定し、Control0 レジスタに適切なビットをセットすることにより SPISVL 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  SPISVL_Start(BYTE bConfiguration)
```

アセンブリ :

```
mov   A,SPISVL_MODE_2 | SPISVL_LSB_FIRST
lcall  SPISVL_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。

記号名	値
SPISVL_MODE_0	0x00
SPISVL_MODE_1	0x02
SPISVL_MODE_2	0x04
SPISVL_MODE_3	0x06
SPISVL_LSB_FIRST	0X80
SPISVL_MSB_FIRST	0X00

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

SPISVL_Stop

説明

Control0 レジスタ中のイネーブル ビットをクリアして、SPISVL モジュールを無効にします。

C プロトタイプ：

```
void SPISVL_Stop(void)
```

アセンブリ：

```
lcall  SPISVL_Stop
```

パラメータ：

なし

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

SPISVL_EnableInt

説明

SPI 完了状態における SPISVL 割り込みを有効にします。SPISVL 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 SPISVL_EnableInt(void)
```

アセンブリ：

```
lcall  SPISVL_EnableInt
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項 ** を参照してください。

SPISVL_DisableInt

説明

SPI 完了状態における SPISVL 割り込みを無効にします。

C プロトタイプ：

```
void SPISVL_DisableInt(void)
```

アセンブリ：

```
lcall  SPISVL_DisableInt
```

パラメータ：

なし

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

SPISVL_ClearInt

説明

掲示された SPISVL 割り込みをクリアします。

C プロトタイプ：

```
void  SPISVL_ClearInt(void);
```

アセンブリ：

```
lcall  SPISVL_ClearInt
```

パラメータ：

なし

戻り値：

なし

副作用：

API セクションの冒頭にある注意事項 ** を参照してください。

SPISVL_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 SPISVL_SetupTxData(WORD wTxData)
```

アセンブリ：

```
mov   X, [wTxData]
mov   A, [wTxData+1]
lcall  SPISVL_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

特殊作用：

API セクションの冒頭にある注意事項 ** を参照してください。

SPISVL_wReadRxData

説明

スレーブデバイスから受信データワードを返します。データワード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：

```
WORD SPISVL_wReadRxData(void)
```

アセンブリ：

```
lcall  SPISVL_wReadRxData
mov   [wRxData], X
mov   [wRxData+1], A
```

パラメータ：

なし

戻り値：

スレーブ SPI から受信し、累算器を通して戻されるデータワード。

特殊作用：

API セクションの冒頭にある注意事項 ** を参照してください。

SPISVL_bReadStatus

説明

現在の SPISVL 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 SPISVL_bReadStatus(void)
```

アセンブリ：

```
lcall  SPISVL_bReadStatus
and    A, SPISVL_COMPLETE | SPISVL_RX_BUFFER_FULL
jnz    SPIVL_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
SPISVL_COMPLETE	0x20
SPISVL_RX_OVERRUN_ERROR	0x40
SPISVL_TX_BUFFER_EMPTY	0x10
SPISVL_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。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。

SPISVL_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、ロー入力の 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 SPISVL_DisableSS(void)
```

アセンブリ：

```
lcall SPISVL_DisableSS
```

パラメータ：

なし

戻り値：

なし

特殊作用：

API セクションの冒頭にある注意事項 ** を参照してください。

SPISVL_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、ロー入力の 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 SPISVL_EnableSS(void)
```

アセンブリ :

```
lcall SPISVL_EnableSS
```

パラメータ :

なし

戻り値 :

なし

特殊作用 :

API セクションの冒頭にある注意事項 ** を参照してください。

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。

```
void main(void)
{
WORD wData;

SPISVL_Start(SPISVL_MODE_0|SPISVL_MSB_FIRST);

M8C_EnableGInt;

while(1)
{
while( !(SPISVL_bReadStatus() & SPISVL_RX_BUFFER_FULL) ); // wait for received data
wData = SPISVL_wReadRxData(); // read received data from the Rx buffer

while( !(SPISVL_bReadStatus() & SPISVL_TX_BUFFER_EMPTY) ); // check for Tx buffer empty
SPISVL_SetupTxData(wData); // send echo packet back

while( !(SPISVL_bReadStatus() & SPISVL_SPI_COMPLETE) ); // encure transaction is
complete
}
}

The same example in ASM will look like following:

export _main

area bss(RAM, REL)
```

```
 wData: blk 2

area text (ROM, REL)

_main:

mov A, (SPISVL_MODE_0|SPISVL_MSB_FIRST)
lcall SPISVL_Start

M8C_EnableGInt

.nextLoop:

.WaitRx:
lcall SPISVL_bReadStatus
and A, SPISVL_RX_BUFFER_FULL
jz .WaitRx

lcall SPISVL_wReadRxData;
push A
push X

.WaitTx:
lcall SPISVL_bReadStatus
and A, SPISVL_TX_BUFFER_EMPTY
jz .WaitTx

pop X
pop A

lcall SPISVL_SetupTxData

.WaitComplete:
lcall SPISVL_bReadStatus
and A, SPISVL_SPI_COMPLETE
jz .WaitComplete
jmp .nextLoop
```

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 SPIMVL、レジスタ : FunctionMSB 優先のコンフィギュレーション :

ビット	7	6	5	4	3	2	1	0
LSB	InvertMOSI	0	0	割り込みタイプ	1	1	1	0
MSB	0	0	0	0	1	1	1	0

Table 4. ブロック SPIMVL、レジスタ : LSB First 機能のコンフィギュレーション :

ビット	7	6	5	4	3	2	1	0
LSB	0	0	0	割り込みタイプ	1	1	1	0
MSB	InvertMOSI	0	0	0	1	1	1	0

このレジスタは SPISVL ユーザーモジュールになるデジタルコミュニケーションブロックの特性を指定します。

Table 5. ブロック SPISVL: レジスタ InputMSB 優先のコンフィギュレーション

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

Table 6. ブロック SPISVL: レジスタ InputLSB 優先のコンフィギュレーション

ビット	7	6	5	4	3	2	1	0
LSB					SCLK			
MSB	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 7. ブロック SPISVL: レジスタ OutputMSB 優先のコンフィギュレーション

ビット	7	6	5	4	3	2	1	0
LSB	0	0	0	~SS				
MSB	0	0	0			MISO		

Table 8. ブロック SPISVL: レジスタ OutputLSB First のコンフィギュレーション

ビット	7	6	5	4	3	2	1	0
LSB				~SS		MISO		
MSB								

~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 9. ブロック SPISVL: シフト レジスタ DR0

ビット	7	6	5	4	3	2	1	0
LSB	シフト レジスタ							
MSB	シフト レジスタ							

Table 10. ブロック SPISVL: 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
LSB	送信バッファレジスタ							
MSB	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 11. ブロック SPISVL: 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
LSB	受信バッファレジスタ							
MSB	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 12. ブロック SPISVL: 制御レジスタ CR0

ビット	7	6	5	4	3	2	1	0
LSB	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
MSB	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。

Table 13. ブロック SPISVL: 制御レジスタ CR1

ビット	7	6	5	4	3	2	1	0
LSB	1	1	0	SPI 長さ				
MSB	1	0	0	SPI 長さ				

SPI 長さは、チェーンモードの SPI の長さを指定します。

[+] Feedback

バージョン ヒストリー

バージョン	著者	説明
1.0	DHA	初期バージョン

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 © 2009-2011 © Cypress Semiconductor Corporation. 本文書に記載される情報は、予告なく変更される場合があります。Cypress Semiconductor Corporation (サイプレス セミコンダクタ社) は、サイプレス 製品に組み込まれた回路以外のいかなる回路を使用することに対しても一切の責任を負いません。特許またはその他の権限下で、ライセンスを譲渡または暗示することもありません。サイプレス 製品は、サイプレスとの書面による合意に基づくものでない限り、医療、生命維持、救命、重要な管理、または安全の用途のために使用することを保証するものではなく、また使用することを意図したものでもありません。さらにサイプレスは、誤動作や故障によって使用者に重大な傷害をもたらすことが合理的に予想される、生命維持システムの重要なコンポーネントとしてサイプレス製品を使用することを許可していません。生命維持システムの用途にサイプレス 製品を供することは、製造者がそのような使用におけるあらゆるリスクを負うことを意味し、その結果サイプレスはあらゆる責任を免除されることを意味します。

PSoC® は Cypress Semiconductor Corp の登録商標であり、PSoC Creator ™ および Programmable System-on-Chip ™ は Cypress Semiconductor Corp. の商標です。本文書で言及するその他すべての商標または登録商標は、各社の所有物です。

全てのソース コード (ソフトウェアおよび / またはファームウェア) はサイプレス セミコンダクタ社 (以下「サイプレス」) が所有し、全世界の特許権保護 (米国およびその他の国)、米国の著作権法ならびに国際協定の条項により保護され、かつそれらに従います。サイプレスが本書面によりライセンシーに付与するライセンスは、個人的、非独占的かつ譲渡不能のライセンスであって、適用される契約で指定されたサイプレスの集積回路と併用されるライセンシーの製品のみをサポートするカスタム ソフトウェアおよび / またはカスタムファームウェアを作成する目的に限って、サイプレスのソース コードの派生著作物をコピー、使用、変更そして作成するためのライセンス、ならびにサイプレスのソース コードおよび派生著作物をコンパイルするためのライセンスです。上記で指定された場合を除き、サイプレスの書面による明示的な許可なくして本ソース コードを複製、変更、変換、コンパイル、または表示することは全て禁止されます。

免責条項 : サイプレス は、明示的または黙示的を問わず、本資料に関するいかなる種類の保証も行いません。これには、商品性または特定目的への適合性の黙示的な保証か゛含まれますが、これに限定されません。サイプレスは、本文書に記載される資料に対して今後予告なく変更を加える権利を留保します。サイプレスは、本文書に記載されるいかなる製品または回路を適用または使用したことによって生ずるいかなる責任も負いません。サイプレスは、誤動作や故障によって使用者に重大な傷害をもたらすことが合理的に予想される生命維持システムの重要なコンポーネントとしてサイプレス製品を使用することを許可していません。生命維持システムの用途にサイプレス製品を供することは、製造者がそのような使用におけるあらゆるリスクを負うことを意味し、その結果サイプレスはあらゆる責任を免除されることを意味します。

ソフトウェアの使用は、適用されるサイプレス ソフトウェア ライセンス契約によって制限され、かつ制約される場合があります。