SPI スレーブ データベース SPIS 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 （バイト数）		ピン （外部入出力ごと）
	CapSense®	I²C/SPI	タイマ	フラッシュ	RAM	
CY8C20x34, CY8C20x24, CY8C20x66, CY8C20x36, CY8C20336AN, CY8C20436AN, CY8C20636AN, CY8C20x46, CY8C20x96, CY7C604xx, CY7C643xx, CYONS2010, CYONS2011, CYONSFN2051, CYONSFN2053, CYONSFN2061, CYONSFN2151, CYONSFN2161, CYONSFN2162, CY8CTST200, CY8CTMG2xx						
		X		43	0	4

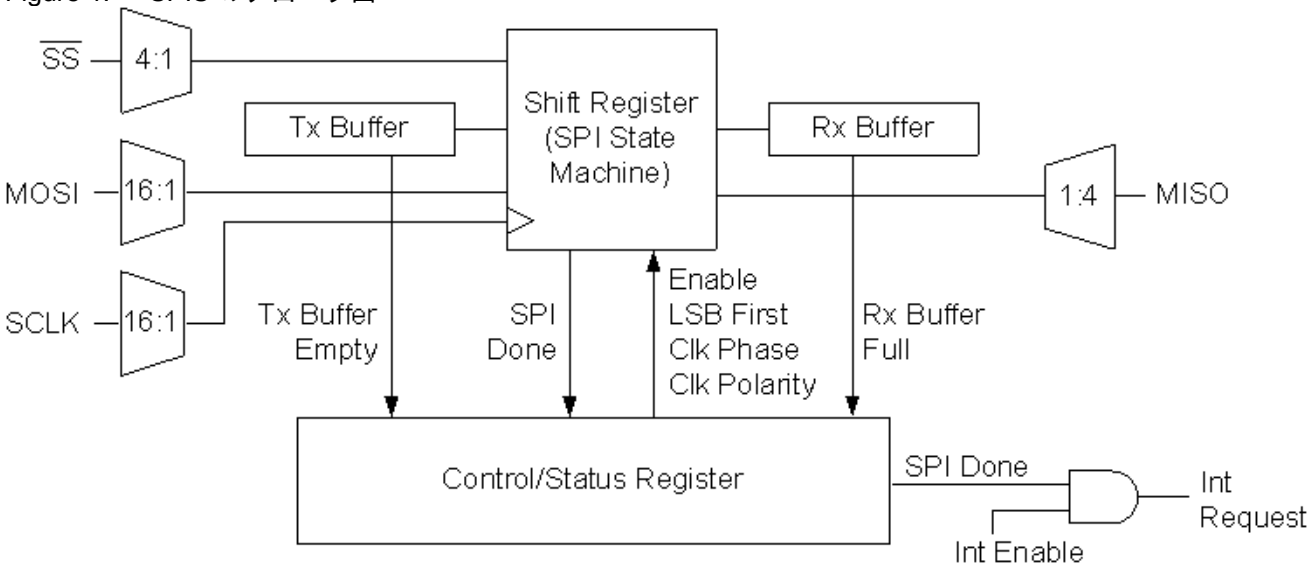
このユーザ モジュールを使用する、ひとつまたは複数の完全設定されたサンプルプロジェクトについては、以下を参照してください。www.cypress.com/psocexampleprojects

特性および概要

- シリアルペリフェラルインターコネクト （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 済み状態へのプログラム可能な割り込み
- SS はファームウェアで管理することもできます。

SPIS ユーザ モジュールは、シリアル　ペリフェラルインターコネクト　スレーブ （SPIS）で、全二重同期の 8 ビット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 優先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ハイ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、I2C/SPI PSoC ブロックの送信バッファ、受信バッファ、制御、構成レジスタ、およびデータシフト用のデータ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 ファームウェアの 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 0、1、2、3 がサポートされます。正しく通信するために、SPI マスタと SPI スレーブの両方を同じモードとビット構成に設定します。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へのデータ変更。
1	立ち上がり	反転	
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	

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、必要な場合にユーザ モジュール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。マスタは、スレーブ選択（~SS）入力信号を低アサートして、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。データバイトで最初に転送されたビットは、MISO 入力信号上でアサートされます。このとき、別の送信データバイトが送信バッファレジスタに書き込まれます。現在のバイトの転送が終了すると、このデータを SPI マスタに送信する準備が整います。

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 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。

留保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の場合、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、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します。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みます。

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。

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。

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_{max}	最大受信 / 送信周波数	--	12	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。I2C/SPI ブロックに配置することもできます。

パラメータおよびリソース

SS イネーブル

このオプションでは、スレーブ選択 (SS_) 信号が内部で駆動するかどうかを特定します。SW_SlaveSelect オプションを選択すると、SPI_CFG レジスタの SS_ bit (マスク 0x08) によって論理レベルが指定されます。外部で駆動している場合、論理レベルは外部ピンによって指定されます。

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 つ目のオプションの 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ことが重要なときに役立ちます。最初のオプションの 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。

割り込み生成制御

以下のパラメータにアクセスできるのは、PSoC Designer の **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チェックされている場合だけです。これらは、**[Project] (プロジェクト) > > [Settings] (設定) > > [Chip Editor] (チップ エディタ) >** でアクセスできます。複数のオーバーレイが使用され、そのオーバーレイ全体の複数のユーザ モジュールにより割り込みが共用されている場合は、割り込み生成制御が重要です。

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、RAM のバイトを消費します。

[+] Feedback

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、各関数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。効率を高めるために、この「レジスタが揮発性である」というポリシーが使用され、PSoC Designer のバージョンが 1.0 であるため、このポリシーが有効になります。C コンパイラは、これらの要求条件を自動的に処理します。アセンブリ言語のプログラマは、コードでこのポリシーを考慮する必要があります。ユーザ モジュール API 関数の中には、A と X を変更しないものもありますが、今後も変更されないという保証はありません。

次に、SPIS による API 関数のリストを挙げます。

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高にアサートします。これはユーザ提供のルーチンで実行します。

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 優先構成を指定する 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。

シンボル名	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

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。

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

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。

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

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。

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

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。

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、アキュムレーターへと送られるデータです。

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。

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、このルーチン呼び出し前に受信バッファ Full（フル）のフラグをチェックします。

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、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：

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 ステータス / コントロールレジスタを読み取り、これを戻します。

C プロトタイプ：

```
BYTE SPIS_bReadStatus(void)
```

アセンブラ：

```
lcall  SPIS_bReadStatus
and   A, SPIS_SPI_COMPLETE | 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、アキュムレーターを通して戻されます。指定されたのマスクを使用し、特定のステータス条件をテストしてください。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。

SPIM 状態マスク	値
SPIS_SPI_COMPLETE	0x20
SPIS_RX_OVERRUN_ERROR	0x40
SPIS_BUFFER_EMPTY	0x10
SPIS_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。

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) を高にセットします。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、行入力の 1 つではなく、値 「SW_SlaveSelect」に設定します。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

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。

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、「SS Enable」という SPIS パラメータを SlaveSelect_Pin. ではなく、SW_SlaveSelect の値に設定します。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

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。

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否定応答が送信されます。

```
;***************************************************************
;  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 so that it is performed in the non interrupt
;  processing. This code is easily written as interrupt driven.
;
;  You must first properly set and start the SPI Slave user module.
;
;***************************************************************
include  "SPIS.inc"
export   LoopBack
export   _main
_main:
        call 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);
    }
```

```
}
void main(void)
{
   LoopBack ();
}
```

コンフィグレーション レジスタ

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: コンフィグレーション レジスタ

ビット	7	6	5	4	3	2	1	0
値	Clock Sel			Bypass	SS_	SS_EN_	IntSel	スレーブ

Clock Sel - クロック選択。これらのビットは、SPI スレーブの操作周波数を指定します。

000b - SysClk / 2

001b - SysClk / 4

010b - SysClk / 8

011b - SysClk / 16

100b - SysClk / 32

101b - SysClk / 64

110b - SysClk / 128

111b - SysClk / 256

バイパス - バイパス同期化。このビットは、入力が SYSCLK と同期するかどうかを指定します。

SS_ - スレーブ選択。このビットは、SS_EN_ 信号がアサートされている場合（SS_EN_ = 0）に、the SS_ 信号の理論値を指定します。

SS_EN_ - スレーブ選択イネーブル。このアクティブロービットは、スレーブ選択（SS_）信号が内部で駆動するかどうかを指定します。内部駆動の場合、その理論レベルは SS_ ビットによって指定されます。外部で駆動している場合、理論レベルは外部ピンによって指定されます。

Int Sel - 割り込み選択。このビットは、どの条件が割り込みを生成するか、また送信レジスタの空条件に基づくか、SPI 完了条件に基づくかを選択します。

スレーブ - このビットは、ブロックがマスタとして機能するか、スレーブとして機能するかを指定します。

Table 4. ブロック SPIS: 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 5. ブロック SPIS: 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 SPIS: コントロールレジスタ

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 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	Enable (イネーブル)

LSB First - LSB ビットを最初に送信します。

受信オーバラン エラー - 前回の受信データバイトが次のバイトを受信するまで読み取られないことを示すフラグです。

SPI 完了 - SPI 送受信サイクルの完了を示すフラグです。

送信バッファ Empty (空) - 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) - シフト レジスタがデータの 1 バイトが受信したことを示すフラグです。

クロックフェーズ - 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 - 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 - 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	著者	説明
2.5	DHA	P1[0] は CY7C64315 および CY7C64316PSoC デバイス用の SPI CLK ピンとして設定します。

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 モジュールの過去のバージョンと現在のバージョンの違いの概要を説明しています。

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.