# 8-Bit シリアルレシーバーデータシート RX8 V 3.50

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

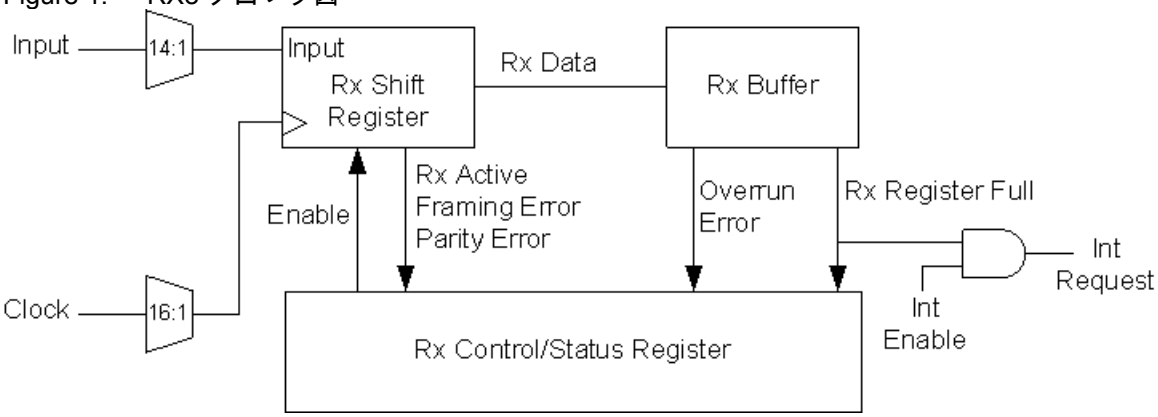
| リソース | PSoC® ブロック | | | API メモリ （バイト） | | ピン |
|---|---|---|---|---|---|---|
| | デジタル | アナログ CT | アナログ SC | フラッシュ | RAM | |
| CY8C29/27/24/22/21xxx, CY8C23x33, CYWUSB6953, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTST300, CY8CTMG300, CY8CTMA300, CY8CTMA301, CY8CTMA301D, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx, CY8C21x12 | | | | | | |
| RxCmdBuf 有効 | 1 | 0 | 0 | 273 | 3 + バッファ | 1 |
| RxCmdBuf 無効 | 1 | 0 | 0 | 77 | 0 | 1 |
| CY8C26/25xxx | | | | | | |
| RxCmdBuf 有効 | 1 | 0 | 0 | 283 | 3 + バッファ | 1 |
| RxCmdBuf 無効 | 1 | 0 | 0 | 87 | 0 | 1 |

## 特性と概要

- 6 Mbit/ 秒以下のバーストレート
- スタートビット、パリティビット、およびストップビットを含むフレームに準拠する RS-232 データフォーマット
- 偶数パリティ、奇数パリティ、もしくはパリティなしのシリアルデータフォーマット
- オプション割り込み装置はレジスタの全状況を受信します。
- 自動フレーミング、オーバーランおよびパリティ・ エラーの検知

RX8 ユーザーモジュールは、クロッキングのプログラミングが可能で、割り込みもしくはポーリングコントロールが選択できる、RS-232 データフォーマット準拠の 8-bit シリアル受信機です。受信したデータのフォーマットは、スタートビット、パリティビット、および終端のトレーリング ストップビットから構成されています。受信機のファームウェアは、デバイスの初期化、受信したバイトの読み取り、およびエラー検知に使われます。

Figure 1. RX8 ブロック図




## 機能説明

RX8 ユーザーモジュールはシリアルレシーバーを実装しています。RX8 は、PSoC デザイナデバイスエディターが “RX” に指定した、PSoC ブロックの一つにマッピングされます。これは、デジタルコミュニケーションタイプの PSoC ブロックのバッファ、シフトおよび制御レジスタを使用します。

制御レジスタは、RX8 ユーザーモジュールのファームウェアのアプリケーション　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。RX8 の初期化では、パリティの設定、任意で RX8 レジスタ フル条件の割り込みの設定に構成、そして受信機の有効化を行います。

スタートビットが RX8 入力で検知された場合、divide-by-eight ビットクロックが起動し、受信ビットの中央のデータをサンプリングするよう同期します 。次の 8 ビットクロックの立ち上がりエッジで、入力データがサンプリングされ、シフトレジスタに移動します。パリティが有効な場合、次のビットクロックでパリティビットをサンプリングします。次のクロックでストップビットがサンプリングされた場合、受信したデータバイトはバッファレジスタに移動され、以下のうちひとつ以上のイベントが発生します。

- 制御レジスタの Rx レジスタ フルビットが設定され、また RX8 割り込み信号が有効な場合は関連した割り込み信号がトリガされます。
- データストリーム内の予想ビット位置からストップビットが検知されなかった場合、制御レジスタにフレーミングエラービットが設定されます。
- 受信されたデータからストップビットを読み込む前にバッファレジスタが読み取られなかった場合、制御レジスタにオーバーランエラービットが設定されます。
- パリティエラーが検知された場合、制御レジスタにパリティエラービットが設定されます。

完全に受信したデータバイトのポーリングを検知するには、制御レジスタの Rx レジスタ フルビットをモニタリングしなければなりません。オーバーラン エラーを防ぐためには、次のバイトが完全に受信される前に、バッファレジスタからデータを読みとらなければなりません。

## High LevelAPI

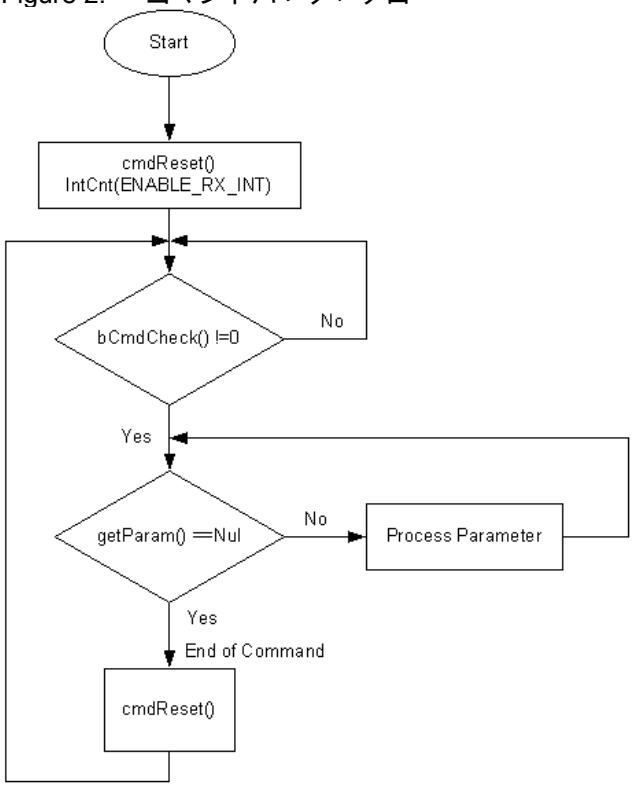
高レベル API では、キャラクターレベルの関数だけではなく、コマンドおよびストリングレベルの関数に対応できる追加のファームウェアが提供されます。ユーザーは、このデバイスエディタを使って、受信機コマンドバッファ、コマンドターミネーター、パラメータデリミタ、および受信機が文字を無視する上限値を設定することができます。

受信機の高レベル関数を仕様するためには、デバイスエディタのウィンドーに移動し、UART を選択してから “RxCmdBuffer” パラメータの 「有効」オプションを選択してください。次に、最も大きいコマンドに 1 を加えた長さを保持できる “RxBufferSize” を選択してください。コマンドターミネーター文字 “CommandTerminator” を選択してください。一般的にはキャリッジリターン (13) またはラインフィード (10) が使用されます。コマンドに 2 つ以上のパラメータが含まれている場合、パラメータ区切り文字 “Param_Delimiter.” を選択してください。一般的なコマンド区切り文字はスペース (32) またはコンマ (44) です。選択された値以下の制御文字も無視される可能性があります。ほとんどの制御文字は 0 から 31 の範囲にあります。このような文字を無視するには “IgnoreCharsBelow” を 32 に設定してください。すべての文字が有効な場合、このパラメータを ‘1‘ に設定してください。コマンドターミネーター (CommandTerminator) に選択された文字は、“IgnoreCharsBelow” オプションの影響を受けません。下のフローチャートは、コマンドバッファ関数の基本的な操作における正しい手順を示します。

コマンドバッファは、UART RX 割り込みサービスルーチン内で、コマンドターミネーターキャラクターを受信するまでキャラクターをバッファ内に取り込みます。この時、コマンドバッファは読み取られる準備ができたことを示すフラグが設定されます。受信バッファは配列を直接読み込むか、INSTANCE_NAME_aRxBufferszGetParam() もしくは szGetRestOfParams() 関数を通してアクセスできます。buffer_size -1 より多いキャラクター数を受信した場合、余分なキャラクターは無視されます。

コマンドバッファは、バッファが埋まるか、コマンドターミネーターキャラクターを受信するまでキャラクターを取り込みます。この二つの条件のいずれかが成立した後に受信したすべてのキャラクターは、CmdReset コマンドが実行されるまで無視されます。CmdReset コマンドが実行された後、RX ISR ファムウェアがは再度キャラクターを取り込みます。

Figure 2.　コマンドバッファフロー



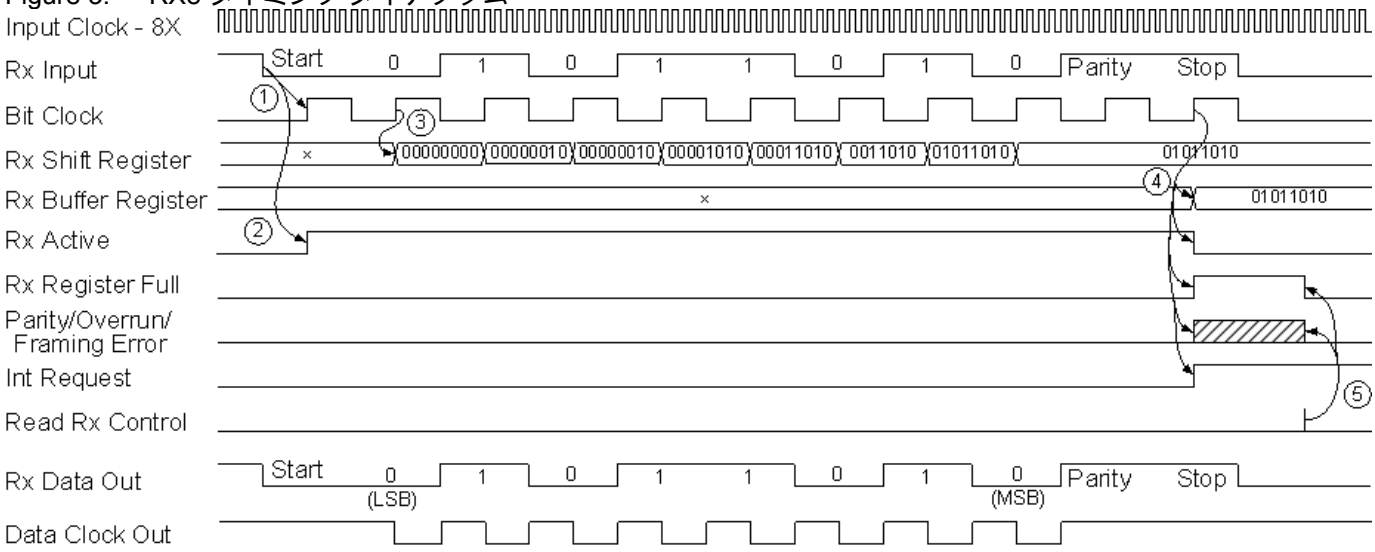
## タイミング

受信した各ビットは、スタートビットの中央と同期された divide-by-eight ビットクロックの立ち上がりエッジでサンプリングされます。

有効な場合、RX8 割り込みが受信バイトあたり 1 回発生します。割り込み信号の有効化および無効化は、API を通じてコントロールされます。

以下の RX8 タイミング ダイアグラムは、RX8 ユーザーモジュールの操作を示しています。

Figure 3. RX8 タイミング ダイアグラム




### 図の備考

1. スタートビットが検知され、divide-by-eight ビットクロックが起動し同期されます。ビットクロックの立ち上がりエッジで、入力ビットストリームがサンプリングされます。
2. スタートビットが検知されることで、Rx 制御 / 状態レジスタに Rx 有効ビットが設定されます。
3. 次のビットクロックの立ち上がりエッジ以降の 8 つのクロックの間、入力信号はサンプリングされ、Rx シフトレジスタに移動されます。
4. ストップビットが検知されると、データが Rx バッファに転送され、Rx 有効がクリアされ、Rx レジスタフルが設定され、パリティ / オーバーラン / フレーングエラーが計算され、有効ならば割り込み要求がトリガされます。
5. Rx 制御レジスタを読み取ると、状態データがデータバスで伝送され、またすべての状態ビットが解除されます。パリティまたはオーバーフローエラーが検知された場合、ファームウェアはいかなる動作も行う必要がありません。フレーミング エラーが検知された場合、フレーマーは次のデータバイトを直ちに探し始めます。システム内である程度の長い時間、RX 入力信号が ロジック 0 で固まる可能性がある場合、受信機を止めなければなりません。また、受信機を再び有効にする前に、ロジック 1 に戻すためラインをポーリングしなければなりません。RX 入力信号が ロジック 0 で固まった場合、ラインがロジック 0 の間に 「虚偽」のスタートビットを繰り返し検知すること、およびスタートビットがあるべき場所でロジック 0 が検知されたことによりフレーミングエラーが発生することを防ぐためです。



[+] Feedback

### コミュニケーション システムの精度

PSoC 装置の SLIMO モードが使われた場合、PSoC システムクロック (SysClk) は有効な UART コミュニケーションを保証できる精度ではありません。また、PSoC が USB に接続されていない場合、PSoC CY8C24x94 製品群の SysClk は、有効な UART コミュニケーションを保証できる精度ではありません。UART コミュニケーションが適切に動作するためには、システムエラー、つまりコミュニケーションリンクの両端のエラーの和は 5% 未満でなければいけません。SysClk の精度についてのより詳しい情報は、装置データシートをご参照ください。

## DC および AC 電気的特性

Table 1. RX8 の DC および AC 電気的特性

| パラメータ | 条件および注記 | 標準値 | リミット | 単位 |
|---|---|---|---|---|
| $F_{max}$ | 最大受信周波数 | | 6 | Mbit/s |

## 配置

RX8 ユーザーモジュールは、デジタルコミュニケーションブロックの任意の場所に配置することができます。

## パラメータおよびリソース

クロック

RX8 のクロック処理は、16 のソースのうちの一つが行います。グローバル I/O バスは、クロック信号を外部ピンもしくは別の PSoC ブロックによって生成されたクロック関数に接続するために使用することができます。ブロックとして外部デジタルクロックを使う場合、精度を高め、スリープ動作をさせるため、、行入力同期は無効にしてください。48 MHz クロック、CPU_32 kHz クロック、分配されたクロックのひとつ、24V1 または 24V2 別の PSoC ブロック出力を、RX8 クロック入力として指定できます。

クロックレートは、ビット受信レートの 8 倍に設定しなければなりません。

入力

原則として、選択されたバスオプションを通して、同期されていないデータ源に入力されます。グローバルバスを使って、入力は外部ピンのどれかに接続することができます。

ClockSync

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 ソースを提供することができます。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。これの場合、システム クロックに対しスキューが発生します。特にシステムバスに関係する、様々なデータおよびバスの最適化を行うに際し、スキューは CY8C29/27/24/22/21xxx および CY8CLED04/08/16 PSoC 装置の製品群において、より問題になります。このパラメータをクロック スキューの制御に使用することで、PSoC ブロック レジスタ値を正常に読み書きできることを保証します。このパラメータの適切な値は、以下の表より判断してください。

| ClockSync 値 | 使い方 |
|---|---|
| SysClk への同期 | 2 つ以上に分割された、24 MHz (SysClk) 由来のクロック ソースには、この設定を使用してください。例として、VC1、VC2、VC3 (VC3 が SysClk により駆動される場合 )、32KHz 、および SysClk ベースのソースを持つデジタル PSoC ブロックがあります。正しい同期が行われるように、外部で生成されたクロック ソースにもこの値を使用してください。 |
| SysClk*2 への同期 | 最終周波数が 48 MHz でない ( 分割数の積が 1 にならない )、48 MHz (SysClk*2) ベースのクロックに対しこの設定を使用します。 |
| SysClk ダイレクトの使用 | 24 MHz (SysClk/1) クロックが適切な場合に使用します。実際に同期化を行うことはありませんが、システムクロックに対してスキューが小さい値を提供します。選択すると、このオプションは上記の Clock パラメータの設定を上書きします。全ての分割を組み合わせたことにより、最終的に 24 MHz の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。 |
| 非同期 | 48 MHz(SysClk*2) 入力が選択される場合、使用します。<br>非同期入力が適切な場合に使用します。一般に、カウンタの唯一の使途が割り込みの生成の場合にのみ、使用することが推奨されます。 |

RX 出力

このパラメータを通して、入力信号が行バスの一つに伝達されます。この信号はデータクロックアウトと共に、循環重複検査といったデータ検証関数を可能にします。

データクロックアウト

このパラメータを通じて、SPI モード 3 のビットクロックが行バスの一つに伝達されます。ビットクロックは、クロック入力信号を 8 で割ったものです。データクロックアウト信号の立ち上がりエッジは、データが安定化しており、サンプリングを行うタイミングに対応します。この信号は RX 出力信号と共に、巡回冗長検査といったデータ検証関数を可能にします。

RxCmdBuffer

このパラメータは、コマンド処理で用いる、コマンド受信バッファおよびファームウェアを有効にします。コマンドバッファが動作するためには、UART RX 割り込みが有効でなければなりません。

RxBufferSize

このパラメータは、受信バッファにどれだけの RAM 容量が確保されるかを決定します。受信可能な最大のコマンドは、ストリングが 0 で終わらなければならないため、選択したバッファの大きさよりも 1 小さいです。RxCmdBuffer が有効で、かつ UART RX 割り込み信号が有効な場合に限り、このパラメータは有効です。

InvertInput

ユーザーは、このパラメータを通して RX 入力信号を反転できます。

CommandTerminator

このパラメータはコマンドの最後を意味するキャラクタを指定します。受信した場合、コマンド全体が受信されたことを意味するフラグが設定されます。このフラグが設定された場合、cmdReset() 機能が呼び出されるまで、以降のキャラクタは無視されます。

Param_Delimiter

このパラメータは、コマンド受信バッファ内のコマンドおよびパラメータのデリミタを指定します。たとえば、Param_Delimiter がスペース文字 (32) に設定された場合、スペースを通じて分離されたサブストリングは、それぞれパラメータとして扱われます。‘cmd foo bar c‘ というストリングがあった場合、パラメータは ‘cmd‘, ‘foo‘, ‘bar‘ および ‘c‘ になります。szGetParam() が呼び出されると、左から右に向かって、次のサブストリングへのポインタを、ゼロで終了するストリングとして返します。

IgnoreCharsBelow

このパラメータの数値以下のキャラクタは、受信バッファに無視されます。パラメータ以下のキャラクタは受信されますが、受信バッファに追加されません。RxCmdBuffer が有効で、かつ UART RX 割り込み信号が有効な場合に限り、このパラメータは有効です。

## 割り込み生成制御

以下の場合、さらに 2 つのパラメータが存在します。**[Enable interrupt generation control ( 割り込み生成の制御を有効にする )]** PSoC デザイナのチェックボックスがチェックされています。これは以下の状況で使用可能です。**プロジェクト > 設定 > チップ エディタ**。「割り込み生成の制御」は、複数のユーザーモジュールが複数のオーバーレイを通して割り込みを共有している場合に重要です。

- 割り込み API
- IntDispatchMode

InterruptAPI

InterruptAPI パラメータを使うことで、ユーザー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 エントリが生成されません。受信コマンドバッファが使われる場合、InterruptAPI パラメータは “Enable ( 有効 )” に設定されなければなりません。1 つのブロックリソースが異なるオーバーレイにより使用されるような、複数のオーバーレイが存在するプロジェクトでは、特に割り込み API が生成されるかどうかを正しく選択することが推奨されます。割り込み API の生成を必要な場合のみ選択することで、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ーモジュールで共有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、共有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、共有割り込みが要求されるたびに行われます。遅延が発生し、共用割り込み要求の処理という非決定性のプロセスも生じますが、RAM を必要としません。「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、共有割り込み要求のソースを計算します。この計算により、割り込みによる遅延は減少し、共有割り込み要求を処理するプロセスは決定性になりますが、RAM を 1 バイト使用します。

## 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、ユーザー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する定数ともに示します。

**Note** ここでは、全てのユーザー 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 レジスタの値を変更されることができます。A と X の値が呼び出し後に必要な場合は、呼び出し元関数にて 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、効率性の観点からこの「registers are volatile（レジスタの揮発性）」ポリシーが採用されています。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守っていることを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 を変更しないこともありますが、将来も変更しないという保証はありません。

RX8 ユーザーモジュールは、API ルーチンを通じてプログラム上で制御することができます。次の表は、ローレベルおよびハイレベルの RX8supplied API 関数を並べてあります。

Table 2. ローレベル RX8 API

| 関数 | 説明 |
|---|---|
| void RX8_Start(BYTE パリティ) | ユーザーモジュールを有効化してパリティを設定します。 |
| void RX8_Stop(void) | ユーザー モジュールを無効にします。 |
| void RX8_EnableInt(void) | 割り込みを有効にします。 |
| void RX8_DisableInt(void) | 割り込みを無効にします。 |
| BYTE RX8_bReadRxData(void) | キャラクターの状態が正しいかどうかをチェックせずに、RX データレジスタのデータを返します。 |
| BYTE RX8_bReadRxStatus(void) | RX 状態レジスタの状態をチェックします。 |

Table 3. ハイレベル RX8 API

| 関数 | 説明 |
|---|---|
| char RX8_cGetChar(void) | 正しいデータが存在する場合、RX データレジスタキャラクタを返します。キャラクタが受信されるまで、関数は戻り値を返しません。 |
| char RX8_cReadChar(void) | RX データレジスタを直ちに読み取ります。正しいデータが存在しない場合は 0 を返し、存在する場合は 1 と 255 の間の ASCII キャラクターを返します。 |
| int RX8_ iReadChar(void) | RX データレジスタを直ちに読み取ります。正しいデータが存在しない、もしくはエラー条件が存在する場合、MSB にエラー状態を返します。受信されたキャラクタを LSB に返します。 |
| void RX8_CmdReset(void) | Rx コマンドバッファを再設定します。 |
| BYTE RX8_bCmdCheck(void) | 正しいコマンドターミネータが受信された場合、0 でない値を返します。 |

| 関数 | 説明 |
|---|---|
| BYTE RX8_bCmdLength(void) | 現在のコマンドの長さを返します。 |
| char * RX8_szGetParam(void) | RX バッファ内の次のパラメータへのポインタを返します。 |
| char * RX8_szGetRestOfParams(void) | 残りのパラメータストリングへのポインタを返します。 |
| BYTE RX8_bErrCheck(void) | コマンドバッファのエラー状態を返します。 |

## RX8_Start

説明：

コントロールレジスタの Rx 有効ビットを設定することで、RX8 受信機のパリティを設定し、RX8 モジュールを有効にします。

C プロトタイプ：

```
void  RX8_Start(BYTE bParitySetting)
```

アセンブリ：

```
mov   A, RX8_PARITY_NONE
lcall  RX8_Start
```

パラメータ：

bParitySetting: 伝送パリティを指定する一つのバイト 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を、以下の表で示します。

| シンボリック名 | 値 |
|---|---|
| RX8_PARITY_NONE | 0x00 |
| RX8_PARITY_EVEN | 0x02 |
| RX8_PARITY_ODD | 0x06 |

戻り値：

なし

副作用：

A および X レジスタは、今回の、また将来のこの関数の実装により変更される可能性があります。大容量メモリモデル (CY8C29xxx および CY8CLED16) のすべての RAM ページポインタレジスタにおいても同様です。必要な場合には、fastcall16 呼び出し関数にて、呼び出し前の値を保存してください。

## RX8_Stop

説明：

制御レジスタ有効ビットをクリアして RX8 モジュールを無効にします。

C プロトタイプ：

```
void  RX8_Stop(void)
```

アセンブリ：

```
lcall RX8_Stop
```

パラメータ：

なし

戻り値：

なし

副作用：

A および X レジスタは、今回の、また将来のこの関数の実装により変更される可能性があります。大容量メモリモデル (CY8C29xxx および CY8CLED16) のすべての RAM ページポインタレジスタにおいても同様です。必要な場合は、fastcall16 呼び出し関数にて、呼び出し前の値を保存してください。

## RX8_EnableInt

説明：

適切な有効ビットをデジタル PSoC ブロック割り込みマスクレジスタに設定し、受信レジスタフル時の RX8 割り込みを有効にします。

C プロトタイプ：

```
void  RX8_EnableInt(void)
```

アセンブリ：

```
lcall  RX8_EnableInt
```

パラメータ：

なし

戻り値：

なし

副作用：

割り込みが保留状態のときにこの API が呼び出された場合、割り込みが直ちにトリガされます。この呼び出しは、Start() が呼び出される前に行われなければなりません。A および X レジスタは、今回の、また将来のこの関数の実装により変更される可能性があります。大容量メモリモデル (CY8C29xxx および CY8CLED16) のすべての RAM ページポインタレジスタにおいても同様です。必要な場合には、fastcall16 呼び出し関数にて、呼び出し前の値を保存してください。

## RX8_DisableInt

説明：

有効ビットをデジタル PSoC ブロック割り込みマスクレジスタからクリアし、受信レジスタフル時の RX8 割り込みを無効にします。

C プロトタイプ：

```
void  RX8_DisableInt(void)
```

アセンブリ：

```
lcall  RX8_DisableInt
```

パラメータ：

なし

戻り値：

なし

副作用：

A および X レジスタは、今回の、また将来のこの関数の実装により変更される可能性があります。大容量メモリモデル (CY8C29xxx および CY8CLED16) のすべての RAM ページポインタレジスタにおいても同様です。必要な場合には、fastcall16 呼び出し関数にて、呼び出し前の値を保存してください。

## RX8_bReadRxData

説明：

バッファレジスタが受信したデータバイトを読み取ります。

C プロトタイプ：

```
BYTE  RX8_bReadRxData(void)
```

アセンブリ：

```
lcall  RX8_bReadRxData
mov   [bRxData],A
```

パラメータ：

なし

戻り値：

受信したデータをアキュムレーターに返します。

副作用：

A および X レジスタは、今回の、また将来のこの関数の実装により変更される可能性があります。大容量メモリモデル (CY8C29xxx および CY8CLED16) のすべての RAM ページポインタレジスタにおいても同様です。必要な場合には、fastcall16 呼び出し関数にて、呼び出し前の値を保存してください。

## RX8_bReadRxStatus

説明：

コントロールレジスタの状態ビットを読み取り、返します。

C プロトタイプ：

```
BYTE  RX8_bReadRxStatus(void)
```

アセンブリ：

```
call  RX8_bReadRxStatus
mov   [bRxStatus],A
```

パラメータ：

なし

戻り値：

状態バイトの読み取り結果を返します。特定の状態条件にあるかを調べるには、以下に定義されたマスクを使用してください。マスクを OR 処理することで、条件を結合できます。

| RX 状態マスク | 値 |
|---|---|
| RX8_RX_ACTIVE | 0x10 |
| RX8_RX_COMPLETE | 0x08 |
| RX8_RX_PARITY_ERROR | 0x80 |
| RX8_RX_OVERRUN_ERROR | 0x40 |
| RX8_RX_FRAMING_ERROR | 0x20 |
| RX8_RX_ERROR | 0xE0 |

副作用：

このレジスタの読み取った場合、すべての状態ビットはクリアされます。戻り値を破棄する前に、すべての状態条件を確認するよう留意してください。A および X レジスタは、今回の、また将来のこの関数の実装により変更される可能性があります。大容量メモリモデル (CY8C29xxx および CY8CLED16) のすべての RAM ページポインタレジスタにおいても同様です。必要な場合には、fastcall16 呼び出し関数にて、呼び出し前の値を保存してください。

## RX8_cGetChar

説明：

RX8 は正しいキャラクタの受信を待ち、受信した値を返します。

C プロトタイプ：

```
CHAR RX8_cGetChar(void)
```

アセンブラ：

```
lcall RX8_cGetChar          ; lcall function to print single character to
                            ; serial port.
mov  [CharBuffer,]A         ; Store retrieved character in buffer
```

パラメータ：

なし

戻り値：

Char bData: RX8 が受信したキャラクターはアキュムレーターに返されます。

副作用：

キャラクターを受信するまで、もしくはキャラクターを受信しているが読み取っていない間、プログラムフローはこの関数で止まります。この関数を使用する場合は、RX8 割り込みを無効にしなければなりません。A および X レジスタは、今回の、また将来のこの関数の実装により変更される可能性があります。大容量メモリーモデル (CY8C29xxx および CY8CLED16) のすべての RAM ページポインタレジスタにおいても同様です。必要な場合には、fastcall16 呼び出し関数にて、呼び出し前の値を保存してください。

## RX8_cReadChar

説明：

RX8 ポートを直ちに読み取ります。データが存在しない、もしくはエラー条件が存在する場合、0 が返されます。それ以外の場合には、読み取ったキャラクタが返されます。

C プロトタイプ：

```
CHAR RX8_cReadChar(void)
```

アセンブラ：

```
lcall RX8_cReadChar          ; lcall function to read a character
cmp  A,0x00                  ; Check for error
jz   ProcessError            ; If error, Process the error condition
mov  [CharBuffer],A          ; Store retrieved character in buffer
```

パラメータ：

なし

戻り値：

CHAR bData: RX8 ポートから読み取られたキャラクタ 1 ~ 255 までの ASCII キャラクタが有効です。戻り値の 0 は、エラー条件またはデータが存在しないことを意味します。

副作用：

関数は、1 ~ 255 の間のキャラクタのみ有効とみなします。0x00 ( 無効 ) 文字はエラー条件として検知します。A および X レジスタは、今回の、また将来のこの関数の実装により変更される可能性があります。大容量メモリーモデル (CY8C29xxx および CY8CLED16) のすべての RAM ページポインタレジスタにおいても同様です。必要な場合には、fastcall16 呼び出し関数にて、呼び出し前の値を保存してください。

## RX8_iReadChar

説明：

RX8 ポートを直ちに読み取り、受信されたキャラクターとエラー条件を返します。

C プロトタイプ：

```
INT RX8_iReadChar(void)
```

アセンブラ：

```
lcall RX8_iReadChar          ; lcall function to read a character
cmp  X,0x00                  ; Check for error
jnz  ProcessError            ; If error, Process the error condition
mov  [CharBuffer]A           ; Store retrieved character in buffer
```

パラメータ：

なし

戻り値：

unsigned int iData: MSB には状態が保持され、LSB には UART RX データが保持されます。MSB が 0 でないのは、エラーが発生したことを意味します。下の表に MSB が返すことのあるエラーコードが示されています。

| エラーフラグ | 値 | 説明 |
|---|---|---|
| RX8_RX_PARITY_ERROR | 0x80 | パリティエラー |
| RX8_RX_OVERRUN_ERROR | 0x40 | バッファ オーバーラン エラー |
| RX8_RX_FRAMING_ERROR | 0x20 | キャラクタフレーミングエラー |
| RX8_RX_NO_ERROR | 0x0E | エラーなし |
| RX8_RX_NO_DATA | 0x01 | データが存在しない |

副作用：

A および X レジスタは、今回の、また将来のこの関数の実装により変更される可能性があります。大容量メモリモデル (CY8C29xxx および CY8CLED16) のすべての RAM ページポインタレジスタにおいても同様です。必要な場合には、fastcall16 呼び出し関数にて、呼び出し前の値を保存してください。

## RX8_CmdReset

説明：

コマンドバッファとフラグをリセットします。これを通して、次のコマンドのキャラクタを受け付けます。

C プロトタイプ：

```
void RX8_CmdReset(void)
```

アセンブラ：

```
lcall RX8_CmdReset          ; lcall function to reset command buffer.
```

パラメータ：

なし

戻り値：

なし

副作用：

RX8 文字カウントをリセットし、受信バッファをクリアします。バッファに残っているすべてのキャラクタが失われます。A および X レジスタは、今回の、また将来のこの関数の実装により変更される可能性があります。大容量メモリモデル (CY8C29xxx および CY8CLED16) のすべての RAM ページポインタレジスタにおいても同様です。必要な場合には、fastcall16 呼び出し関数にて、呼び出し前の値を保存してください。現状では、CUR_PP ページ のポインタ レジスタのみが変更されます。

## RX8_bCmdCheck

説明：

コマンドターミネータが受信されたかどうか確認します。

C プロトタイプ：

```
BYTE RX8_bCmdCheck(void)
```

アセンブラ：

```
lcall RX8_bCmdCheck      ; lcall function to get command complete status.
cmp  A,0x00              ; Check if command complete
jnz  ProcessCmd          ; Process command buffer
```

パラメータ：

なし

戻り値：

コマンドターミネータを受信した場合、0 以外の値が返されます。

副作用：

A および X レジスタは、今回の、また将来のこの関数の実装により変更される可能性があります。大容量メモリモデル (CY8C29xxx および CY8CLED16) のすべての RAM ページポインタレジスタにおいても同様です。必要な場合には、fastcall16 呼び出し関数にて、呼び出し前の値を保存してください。現状では、CUR_PP ページのポインタ レジスタのみが変更されます。

## RX8_bCmdLength

説明：

コマンドバッファにキャラクタの長さを返します。この命令は、コマンドターミネータを受信したかどうかに関わらず、現在のコマンドの長さを返します。

C プロトタイプ：

```
BYTE RX8_bCmdLength(void)
```

アセンブラ：

```
lcall RX8_bCmdLength      ; lcall function to get current command
                          ; Command length is returned in Accumulator
```

パラメータ：

なし

戻り値：

受信バッファ内の現在のストリングの長さ

副作用：

A および X レジスタは、今回の、また将来のこの関数の実装により変更される可能性があります。大容量メモリーモデル (CY8C29xxx および CY8CLED16) のすべての RAM ページポインタレジスタにおいても同様です。必要な場合には、fastcall16 呼び出し関数にて、呼び出し前の値を保存してください。現状では、CUR_PP ページのポインタ レジスタのみが変更されます。

## RX8_szGetParam

説明：

受信バッファ内の、デバイス エディターで設定されたデリミタ Param_Delimiter で区切られた、次のパラメータを返します。すべてのパラメータが返された後は、以降の呼び出しはヌルポインタ (0) を返します。

C プロトタイプ：

```
char * RX8_szGetParam(void)
```

アセンブラ：

```
lcall RX8_szGetParam       ; lcall function to return pointer to the
                           ; next parameter.
                           ; Pointer is returned in A and X.
```

パラメータ：

なし

戻り値：

char * strPtr: パラメータストリングへのポインタ

副作用：

受信バッファは、szGetParam が呼び出される度に変更されます。ナル (0) が、各パラメータの後に配置されます。A および X レジスタは、今回の、また将来のこの関数の実装により変更される可能性があります。大容量メモリーモデル (CY8C29xxx および CY8CLED16) のすべての RAM ページポインタレジスタにおいても同様です。必要な場合には、fastcall16 呼び出し関数にて、呼び出し前の値を保存してください。現在変更されているのは、CUR_PP および IDX_PP ページのポインタレジスタです。

## RX8_szGetRestOfParams

説明：

szGetParam により返されなかった、残りの受信バッファストリングへのポインタを返します。この関数が szGetParam の前に呼び出された場合、受信バッファ全体へのポインタが返されます。ストリングの終端に到達した場合、ナル (0x00) が返されます。

C プロトタイプ：

```
char * RX8_szGetRestOfParams(void)
```

アセンブラ：

```
lcall RX8_szGetRestOfParams    ; lcall function to return pointer to the
                               ; remainder of the receive buffer.
                               ; Pointer is returned in A and X.
```

パラメータ：

なし

戻り値：

char * strPtr: 残りの受信ストリングへのポインタ

副作用：

A および X レジスタは、今回の、また将来のこの関数の実装により変更される可能性があります。大容量メモリーモデル (CY8C29xxx および CY8CLED16) のすべての RAM ページポインタレジスタにおいても同様です。必要な場合には、fastcall16 呼び出し関数にて、呼び出し前の値を保存してください。現在、CUR_PP ページ ポインタ レジスタのみが変更されています。

## RX8_bErrCheck

説明：

bErrCheck が最後に呼び出された時点以降のコマンドエラーをチェックします。

C プロトタイプ：

```
BYTE RX8_bErrCheck(void)
```

アセンブラ：

```
lcall RX8_bErrCheck             ; lcall function to return error status
cmp  A,0x00                     ; Check for error
jnz  ProcessError               ; If error, Process the error condition
```

パラメータ：

なし

戻り値：

BYTE bErr: MSB は、この関数が最後に呼び出された後で発生したかもしれない、すべてのエラー条件を保持しています。

| エラーフラグ | 値 | 説明 |
|---|---|---|
| RX8_RX_PARITY_ERROR | 0x80 | パリティエラー |
| RX8_RX_OVERRUN_ERROR | 0x40 | バッファ オーバーラン エラー |
| RX8_RX_FRAMING_ERROR | 0x20 | 文字フレーミングエラー |
| RX8_RX_BUF_OVERRUN | 0x10 | ソフトウェア RX バッファオーバーラン |

副作用：

エラー状態がクリアされます。A および X レジスタは、今回の、また将来のこの関数の実装により変更される可能性があります。大容量メモリーモデル (CY8C29xxx および CY8CLED16) のすべての RAM ページポインタレジスタにおいても同じです。必要な場合には、fastcall16 呼び出し関数にて、呼び出し前の値を保存してください。現在、CUR_PP ページ ポインタ レジスタのみが変更されています。

## ファームウェア ソースコードの例

次のサンプルファームウェアは、API 関数を使用して、データバイトを受信するため、RX8 をデータバイトを受信するまで待機させるルーチンを生成するものです。

```
;-------------------------------------------------------------------------
; Calling routine in Assembly:
;
;    call fWaitToReceiveByte    ; status of RX data returned in A
;    jnz  RX_DATA_NO_ERROR      ; no error found - data in bRxData
;    jmp  RX_DATA_WITH_ERROR    ; error found - status flags in bRxData
;
; Description: Waits for a byte of data to be received by an RX8 receiver.
; Byte of data is returned in a common storage location, bRxData. Value of
; bRxData depends on status of received byte. If there was no error on
; received data then bRxData contains received data byte, else bRxData
; contains the status flags from RX8 control register.
;
; Return Value: In A register:
;   TRUE - data successfully received, bRxData contains received data byte.
;   FALSE - data received with error, bRxData contains status flags.
;
;-------------------------------------------------------------------------

include “rx8.inc”
;;;;;;;;;;;;;;;;;;;;;;;;;;
; exports
;;;;;;;;;;;;;;;;;;;;;;;;;;
; routine name
export fWaitToReceiveByte    ; assembly routine label
export _fWaitToReceiveByte   ; C code label

; Rx data storage
export bRxData               ; assembly routine label
export _bRxData              ; C code label

;;;;;;;;;;;;;;;;;;;;;;;;;;
; data storage
;;;;;;;;;;;;;;;;;;;;;;;;;;
area bss(ram,con,rel)
bRxData:                     ; Rx data storage area
_bRxData:
    blk   1

area text(rom,con,rel)

;;;;;;;;;;;;;;;;;;;;;;;;;;
; equates
;;;;;;;;;;;;;;;;;;;;;;;;;;
TRUE:   equ   1
FALSE:  equ   0

;;;;;;;;;;;;;;;;;;;;;;;;;;
; Routine Code
;;;;;;;;;;;;;;;;;;;;;;;;;;
 fWaitToReceiveByte::
```

```
_fWaitToReceiveByte::
; Wait for byte to be received
.WAIT_FOR_RX_COMPLETE:
    call  RX8_bReadRxStatus
    push  A
    and   A, RX8_RX_COMPLETE
    jnz   .CHECK_RX_ERRORS
    pop   A
    jmp   .WAIT_FOR_RX_COMPLETE

; Data completely received now check for errors
.CHECK_RX_ERRORS:
    pop   A                      ; Restore status register state
    and   A, RX8_RX_NO_ERROR     ; mask off non-status bits
    jz    .DATA_RX_WITH_NO_ERRORS ; data is valid - no error detected

; Errors detected in received data - return with error condition
;   1) A is set to FALSE indicating error condition
;   2) bRxData contains the RX status flags for further processing
.RX_ERRORS_FOUND:
    mov   [bRxData], A           ; bRxData contains the status flags
    call  RX8_bReadRxData        ; Read RxData reg to prevent future
                                 ; overrun error
    mov   A, FALSE               ; Set A to FALSE condition
    ret

; No error detected in received data - return with data
;   1) A is set to TRUE indicating NO error condition
;   2) bRxData contains the received data byte
.DATA_RX_WITH_NO_ERRORS:
    call  RX8_bReadRxData        ; get the received data in A
    mov   [bRxData], A           ; bRxData contains received
                                 ; data byte
    mov   A, TRUE                ; set a to NO error condition
    ret

END_fWaitToReceiveByte:
```

このサンプルプロジェクトは C で作成されています。

```
//-------------------------------------------------------------------------
//
// Prototype:
// BOOL fWaitToReceiveByte(void);
// Usage:
// if ( fWaitToReceiveByte() )
// {
// /* data valid - process it here! - data in bRxData */
// }
// Return Value:
//    TRUE -  data successfully received, bRxData contains
//            received data byte.
//    FALSE - data received with error, bRxData contains
//            status flags.
//
//-------------------------------------------------------------------------


#include "psocapi.h"
#include "m8c.h"
#include "RX8.h"


/* Global bRxData - saves code - ptrs are expensive */
BYTE bRxData;

BOOL fWaitToReceiveByte(void)
{
    BYTE bRxStatus;

    /* Wait to receive full byte*/
    while ( !( bRxStatus=RX8_bReadRxStatus() & RX8_RX_COMPLETE ) )
    {
        /* might want to sleep or keep track of time */
    }

    /* data received, now check for errors */
    if (( bRxStatus & RX8_RX_NO_ERROR ) == 0 )
    {
        /* no error detected */
        bRxData = RX8_bReadRxData();
        return( TRUE );
    }
    else
    {
        /* error detected */
        bRxData = bRxStatus;
        return( FALSE );
    }
}
```

## 設定レジスタ

RX8 ユーザーモジュールを構成する時に使用する、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ついての説明が下にあります。パラメータ化された記号のみ説明されています。

Table 4. ブロック RX、レジスタ：関数

| Bit（ビット） | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 値 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 |

このレジスタは、RX8 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定義します。

Table 5. ブロック RX、レジスタ：入力

| Bit（ビット） | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 値 | 入力ソース | | | | クロックソース | | | |

「入力ソース」にて、RX8 の入力ソースを選択します。「クロックソース」にて、受信機のタイミングを駆動するクロックを選択します。

Table 6. ブロック RX、レジスタ：出力

| Bit（ビット） | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 値 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

このレジスタは使用されません。

Table 7. ブロック RX、データシフトレジスタ DR0

| Bit（ビット） | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 値 | RX8 シフトレジスタ | | | | | | | |

RX8 シフトレジスタ : 入力がスタートビットを検知した場合、RX8 ステートマシンハードウェアは、データをこのレジスタに移動するための divide-by-8 ビットクロックを生成します。

Table 8. ブロック RX、データレジスタ : DR1

| Bit（ビット） | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 値 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

このレジスタは使用されません。

Table 9. ブロック RX、データバッファレジスタ : DR2

| Bit（ビット） | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 値 | RX8 バッファレジスタ | | | | | | | |

RX8 バッファレジスタ : ストップビットがサンプリングされた後に、RX8 シフトレジスタからデータが伝送されます。

Table 10. ブロック RX、制御 / 状態レジスタ : CR0

| Bit ( ビット ) | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| 値 | パリティエラー | オーバーランエラー | フラミングエラー | Rx 活性 | Rx レジスタ全体 | パリティタイプ | パリティ有効 | Rx 有効 |

パリティエラーは、受信データバイトのパリティ計算結果を表すフラグです。

オーバーランエラーは、RX バッファレジスタデータが上書きされたことを表わすフラグです。

フレーミングエラーは、ストップビットが適切に受信されたことを表すフラグです。

RxActive(Rx 有効 ) は、データバイトを能動的に受信しているかどうかを表すフラグです。

RxRegFull(Rx レジスタフル ) はデータバイトが全て受信され、データバイトが Rx バッファレジスタに伝送され、エラー条件が有効であることを表すフラグです。

パリティタイプは、計算するパリティのタイプです。このビットは “don’t care if Parity Enable bit is not set( パリティ有効ビットを設定しない場合でも関係ない )” です。

パリティ有効は、受信したパリティビットの計算を有効または無効にします。パリティは、パリティタイプビットを設定することで選択されます。

Rx 有効は、RX8 受信機を有効または無効にします。

## バージョン ヒストリー

| バージョン | 提供元 | 説明 |
|---|---|---|
| 3.4 | DHA | 大容量メモリモデルチップとの互換性が追加されました。<br><br>外付けの水晶振動子を使わない限り、ユーザーに 32 kHz オプションが使用できないことを通知する DRC が追加されました。 |
| 3.50 | DHA | デバイス CY8C21x12 がサポートが追加されました。 |

**Note** PSoC Designer 5.1 では、すべてのユーザ ーモジュールの データシートに対しバージョン ヒストリーを導入しました。このセクションは、ユーザー モジュールにおける過去と現在のバージョンの、違いの概略が書かれています。



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.