8-Bit シリアルトランスミッタ データシート TX8 V 3.50

Copyright © 2001-2011 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（各外部 I/O およびクロックにつき）
	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, CYWUSB6953, CY8C23x33, CY8CLED02/04/08/16, CY8CLED0xD, CY8CLED0xG, CY8CTST110, CY8CTMG110, CY8CTST120, CY8CTMG120, CY8CTMA120, CY8C21x45, CY8C22x45, CY8CTMA30xx, CY8C28x45, CY8CPLC20, CY8CLED16P01, CY8C28xxx, CY8C21x12						
	1	0	0	193	0	1
CY8C26/25xxx	1	0	0	203	0	1

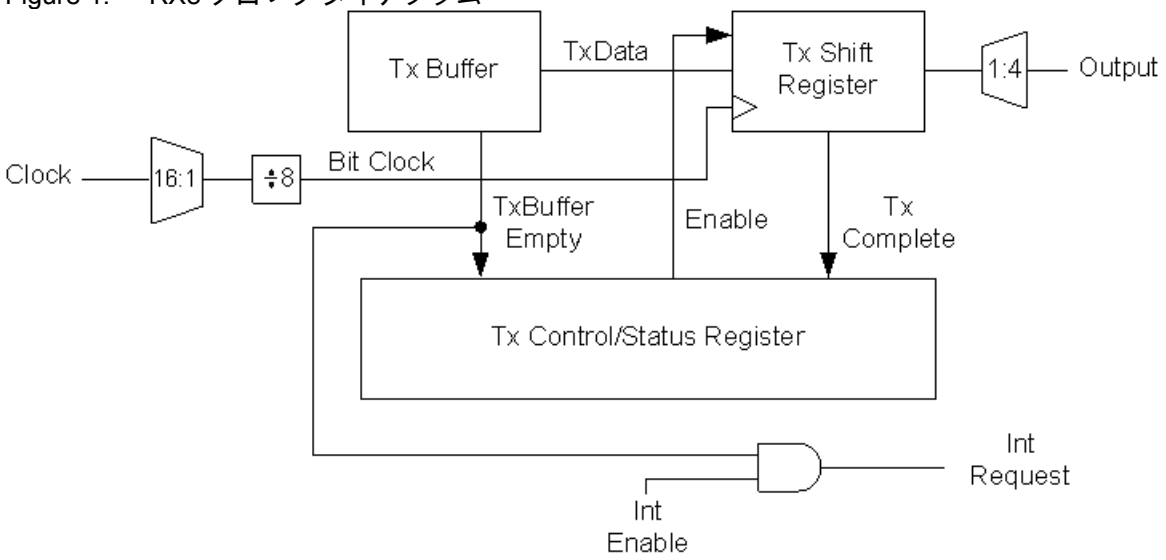
このユーザ モジュールを使用するいくつかの設定を利用するサンプルプログラムについては、www.cypress.com/psocexampleprojects を参照してください。

特性および概要

- クロック動作が 48 MHz まで選択可能な 8-bit シリアルトランスミッタで、最大 6 Mbit データ転送速度を実現
- スタート、パリティオプション、ストップビットで構成されるデータフレーミング
- 偶数や奇数、またはパリティのない RS-232 直列データ準拠フォーマット
- 送信バッファが空の条件における割り込み（オプション）

TX8 ユーザーモジュールは、プログラミングが可能なクロッキングと選択可能な割り込み、またはポーリングコントロール作業が行える 8-bitRS-232 データフォーマット準拠の直列トランスミッタです。送信されたデータは、最初のスタートビット（オプションのパリティビット）およびストップビットとともにフレーミングされます。トランスミッタファームウェアは、初期化、開始、停止、ステータスの読み出し、TX8 へのデータの書き込みに使用されます。

Figure 1. RX8 ブロック ダイアグラム



機能説明

TX8 ユーザ モジュールは、シリアルトランスミッタを実装します。これは、デジタルコミュニケーション用の PSoC ブロックに対するバッファ、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、TX8 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。制御レジスタでイネーブル ビットがセットされている場合、内部の divide-by-eight ビットクロックが生成されます。

送信するデータバイトは、API ルーチンによってバッファレジスタに書き込まれ、制御レジスタでバッファ Empty（空）状態ビットをクリアします。この状態ビットは、送信オーバーランエラーを検知・防止するために使用されます。

次のビットクロックの立ち上がりエッジで、シフト レジスタにデータが送信され、制御レジスタのバッファ Empty（空）状態ビットがセットされます。割り込みは、割り込みイネーブルマスクが有効になっている場合にトリガされます。この割り込みによって、次のバイトキューの送信が有効となるため、現在のデータバイトの送信が完了すると、次の利用可能な送信クロックで新しいバイトが送信されます。

スタートビットは、データバイトがバッファレジスタからシフト レジスタに転送されるのと同時に送信されます。連続したビットクロックによって、シリアルビットストリームが出力にシフトされます。ストリームは、データバイトの各ビット（最下位ビットから）、オプションのパリティビット、最後のストップビットによって構成されます。ストップビットの送信が完了すると、制御レジスタの送信完了状態ビットがセットされます。このビットは、読み出しまで有効になっています。新しいデータバイトがバッファレジスタに書き込まれると、データバイトはシフト レジスタに転送され、ビットクロックの次の立ち上がりエッジでデータの送信が始まります。

[+] Feedback

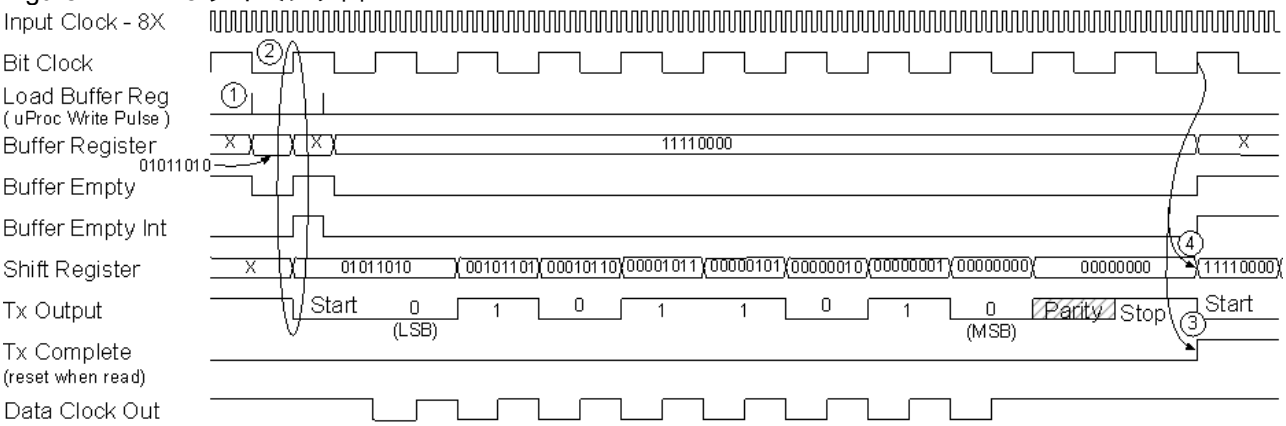
タイミング

クロックレートは、希望するビットレートの 8 倍に設定しなければなりません。

有効になっている場合、送信バッファ Empty（空）イベントで TX8 割り込みが発生します。これは、送信されるデータバイトがバッファレジスタからシフト レジスタに送信されるときに発生します。割り込み信号のイネーブルおよびディスエーブル作業は API を通じてコントロールされます。

次の TX8 タイミング・ダイアグラムでは TX8 ユーザーモジュール作動方法が説明されています。

Figure 2. RX8 タイミング図



コミュニケーション・システムの正確度

PSoC デバイスの SLIMO モードが使われる場合、PSoC システムクロック (SysClk) は正常な UART コミュニケーションを保証するのに十分な精度をもっていません。なお、PSoC が USB につながらない場合、PSoC CY8C24x94 製品群にある装置の SysClk は正常な UART コミュニケーションを保証するのに十分な精度をもっていません。UART コミュニケーションが適切に作動するためには、システムエラー、つまり通信の両端で、エラー合計は 5% より小さく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考ください。

DC 電気的特性と AC 電気的特性

Table 1. TX8 の 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}	最大送信周波数	--	6	Mbits

配置

TX8 ユーザ モジュールは、「TX」と指定されている単一ブロックを使用します。これは、任意のデジタル通信 PSoC ブロックに自由にマッピングできます。

[+] Feedback

パラメータおよびリソース

クロック

TX8 は 16 種類のソースのうち、ひとつのクロックを選択できます。このパラメータは、PSoC Designer で Device Editor を使用してセットされます。グローバル I/O バスは、、クロック入力をクロック出力につないだり、クロック信号を別の PSoC ブロックによって生成されたクロック機能につなげることができます。外部デジタルクロックを使う場合、最高の精度とスリープ動作のために、raw input の同期機能を off にしなければなりません。48 MHz クロック、CPU_32 kHz クロック、分配されたクロックのうち、ひとつ 24V1 または 24V2 また別の PSoC ブロック出力 TX8 クロック入力で指定できます。

クロックレートは、希望するビット送信率の 8 倍に設定しなければなりません。 10 クロック入力毎に、1 データビットが送信されます。

出力

トランスミッタの出力は、グローバル出力バスにルーティングできます。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。

TX 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。割込みサービス ルーチンでは、TX_CONTROL_REG を読み込んで次の割り込みを有効にします。

ClockSync (クロック同期)

PSoC デバイスでは、システム クロックに加えて、デジタル ブロックはクロック ソースを提供できます。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。これで、システム クロックに関するスキューが発生することになります。様々なデータ最適化作業、とりわけシステムバスに適用された最適化作業のため、このようなスキューは CY8C29/27/24/22/21xxx および CY8CLED04/08/16 PSoC 装置の製品群に取って特に重要です。このパラメータは、制御クロック スキューに使用され、PSoC ブロック レジスタ値を読み書きする場合に、正しい動作を保証します。このパラメータの適切な値は、以下の表から決定してください。

ClockSync 値	使用
SysClk への同期	2 以上で除算される 24 MHz (SysClk) からの 派生クロック ソースでこの設定を使用します。例には、VC1、VC2、VC3 (VC3 が SysClk によって駆動される場合)、32KHz、SysClk ベースのソースを持つデジタル PSoC ブロックが含まれます。正しい同期が行われるよう、外部で生成されたクロック ソースもこの値を使用してください。
SysClk*2 への同期	結果の周波数が 48 MHz でない限り (言い換えると、すべての除算結果が 1 になる場合)、48 MHz (SysClk*2) ベースのクロックでは、この設定を使用します。
SysClk Direct の使用	24 MHz (SysClk/1) クロックが適切な場合に使用します。これは、同期を実際には実行しませんが、システム クロック自体への低スキュー アクセスを提供します。選択すると、このオプションは、上の Clock パラメータの設定を上書きします。組み合わせた全除算値の最終結果が 24 MHz 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。
Unsynchronized (非同期)	48 MHz (SysClk*2) 入力が選択される場合に使用します。 非同期入力が適切な場合に使用します。一般に、割り込み生成がカウンタの単独の用途である場合にのみ使用することを推奨します。

データクロックアウト

データクロック出力信号は、8 分したクロック入力に相当します。このクロックは、送信のデータビット中のみアクティブで、その他のときは常に 「High」を維持します。クロックの立ち上がりエッジは、データが安定化しサンプリングされる時間と一致します。データクロックアウト信号は、巡回冗長検査などのデータ検証機能を容易にします。

割り込み生成制御

次の 2 つのパラメータ、InterruptAPI と IntDispatchMode は、イネーブルをセットすることによってのみアクセス可能です。**PSoC デザイナ中の割り込み生成制御** チェックボックス これは **[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。

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。受信命令バッファが使われる場合、InterruptAPI パラメータが “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。必要な場合にのみ Interrupt API 生成を選択して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、オーバーヘッドが軽減できます。

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 の全治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。

API ルーチンを通じて TX8 ユーザーモジュールに対するプログラムで成業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 TX8 による API 機能を一覧表示しています。

Table 2. ローレベル TX8 API

関数	説明
void TX8_Start(BYTE bParity)	ユーザーモジュールをイネーブルしてパリティを設定します。
void TX8_Stop(void)	ユーザ モジュールを無効にします。
void TX8_EnableInt(void)	TX 割り込みを有効にします。
void TX8_DisableInt(void)	TX 割り込みを無効にします。
void TX8_SetTxIntMode(BYTE bTxIntMode)	TX 割り込みのソースを設定します。
void TX8_SendData(BYTE bTxData)	TX ステータスを確認せずにバイトを送信します。
BYTE TX8_bReadTxStatus(void)	TX 状態レジスタにステータスを返します。

Table 3. ハイレベル TX8 API

関数	説明
void TX8_PutString(char * szStr)	TX8 ポートからヌル終端ストリングを送信します。
void TX8_CPutString(const char * azStr)	TX8 ポートからヌル終端定数 (ROM) ストリングを送信します。
void TX8_PutChar(char bData)	TX レジスタが空のとき、TX8 に文字を送信します。TX データレジスタがデータオーバーランエラーを起こさずに書き込めるようになるまで、関数は戻りません。
void TX8_Write(char * aStr, BYTE bCnt)	bCnt バイトを aStr アレイから TX8 ポートに送信します。

関数	説明
void TX8_CWrite(const char * aStr, int iCnt)	iCnt バイトと定数 aStr アレイから TX8 ポートに送信します。
void TX8_PutSHexByte(BYTE bValue)	16 進法表記による bValue の 2 文字を TX8 ポートに送信します。
void TX8_PutSHexInt(int iValue)	16 進法表記による iValue の 4 文字を TX8 ポートに送信します。
void TX8_PutCRLF(void)	キャリッジ·リターン（0x0D）とラインフィード（0x0A）を TX8 ポートに送信します。

TX8_Start

説明

コントロールレジスタの Tx 有効ビットを設定することで TX8 トランスミッタのパリティを設定し、TX8 モジュールを有効にします。いったん有効になると、バッファレジスタの書き込み後、次のビットクロックで送信が行われます。

C プロトタイプ :

```
void  TX8_Start(BYTE bParitySetting)
```

アセンブラ :

```
mov   A, TX8_PARITY_NONE
lcall  TX8_Start
```

パラメータ :

bParitySetting: 送信パリティのバイト 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は、以下の表で示されます。

記号名	値
TX8_PARITY_NONE	0x00
TX8_PARITY_EVEN	0x02
TX8_PARITY_ODD	0x06

戻り値 :

なし

特殊作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリーモデル (CY8C27x66、CY8C29xxx、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。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

TX8_Stop

説明

制御レジスタの Tx イネーブルビットをクリアして TX8 モジュールを無効にします。

C プロトタイプ :

```
void  TX8_Stop(void)
```

アセンブラ :

```
lcall  TX8_Stop
```

パラメータ :

なし

戻り値 :

なし

特殊作用 :

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリーモデル (CY8C27x66、CY8C29xxx、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。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

TX8_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、送信バッファ Empty (空) の条件で TX8 割り込みを有効にします。TX8 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。詳細は、選択したコンポーネントの PSoC データシートを参照してください。

C プロトタイプ :

```
void  TX8_EnableInt(void)
```

アセンブラ :

```
lcall  TX8_EnableInt
```

パラメータ :

なし

戻り値 :

なし

特殊作用 :

割り込み信号が保留され、この API が呼び出される場合、割り込み信号が直ちにトリンガされます。Start() が呼び出される前に、この呼び出しが遂行されなければなりません。A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリーモデル (CY8C27x66、CY8C29xxx、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。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

TX8_DisableInt

説明

デジタル PSoC ブロック割り込みマスクレジスタに適切な有効ビットをクリアし、送信バッファ Empty（空）の条件で TX8 の割り込みを無効にします。

C プロトタイプ：

```
void  TX8_DisableInt(void)
```

アセンブラ：

```
lcall  TX8_DisableInt
```

パラメータ：

なし

戻り値：

なし

特殊作用：

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリーモデル (CY8C27x66、CY8C29xxx、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。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

TX8_SetTxIntMode

説明

割り込みのソースを設定します。

C プロトタイプ：

```
void  TX8_SetTxIntMode(BYTE bTxIntMode)
```

アセンブラ：

```
lcall  TX8_SetTxIntMode
```

パラメータ：

bTxIntMode: 割り込みモードを指定する 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。

TX 割り込みモード	値
TX8_INT_MODE_TX_REG_EMPTY	0x00
TX8_INT_MODE_TX_COMPLETE	0x01

戻り値：

なし

特殊作用：

A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリーモデル (CY8C27x66、CY8C29xxx、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。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

TX8_SendData

説明

特定の送信データをバッファレジスタに読み込むことによって、データ送信を開始します。制御レジスタの送信完了状態ビットは、送信が開始したことを確認するために、モニタしなければなりません。

C プロトタイプ :

```
void  TX8_SendData(BYTE bTxData)
```

アセンブラ :

```
mov   A, bTxData
lcall  TX8_SendData
```

パラメータ :

bTxData: データは送信バッファレジスタに読み込まれます。アキュムレーターを通過します。

戻り値 :

なし

特殊作用 :

送信バッファ Empty（空）の条件による割り込みが設定されている場合、bTxData がバッファレジスタからシフト レジスタへ送信されたときに割り込みが生成されます。A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリーモデル (CY8C27x66、CY8C29xxx、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。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

TX8_bReadTxStatus

説明

制御レジスタの内容を返します。

C プロトタイプ :

```
BYTE TX8_bReadTxStatus(void)
```

アセンブラ :

```
lcall TX8_bReadTxStatus
and  A, TX8_TX_COMPLETE
jnz  TxIsComplete
```

パラメータ :

なし

戻り値 :

ステータスバイトの読み取り結果を変換します。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、複数のコンディションを確認できます。

RX 状態マスク	値
TX8_TX_COMPLETE	0x20
TX8_TX_BUFFER_EMPTY	0x10

特殊作用：

読み出し実行後に状態ビットがクリアされます。A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリーモデル (CY8C27x66、CY8C29xxx、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。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

TX8_PutString

説明

ヌル終端（RAM）ストリングを TX8 ポートに送信します。

C プロトタイプ：

```
void TX8_PutString(char * szRamString)
```

アセンブラ：

```
mov  A,>szRamString     ; Load MSB part of pointer to RAM based null
                        ; terminated string.
mov  X,<szRamString     ; Load LSB part of pointer to RAM based null
                        ; terminated string.
lcall TX8_PutString      ; Call function to send string out TX8 port
```

パラメータ：

char * aRamString: TX8 ポートに送信されるストリングのポインタ。MSB はアキュムレーターを通り、LSB は X レジスタを通ります。

戻り値：

なし

特殊作用：

最後の文字が TX8 送信バッファに読み込まれるまで、プログラムフローは、この関数内にとどまります。A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリーモデル (CY8C27x66、CY8C29xxx、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。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。現時点では , IDX_PP ページ ポインタ レジスタのみが変更されています。

[+] Feedback

TX8_CPutString

説明

定数（ROM）、ヌル終端ストリングを TX8 ポートから送信します。

C プロトタイプ：

```
void TX8_CPutString(const char * szROMString)
```

アセンブラ：

```
mov  A,>szRomString   ; Load MSB part of pointer to ROM based null
                      ; terminated string.
mov  X,<szRomString   ; Load LSB part of pointer to ROM based null
                      ; terminated string.
lcall TX8_CPutString   ; Call function to send constant string out
                      ; TX8 port
```

パラメータ：

const char * szROMString: TX8 ポートに送信されるストリングのポインタ。ストリングポインタの MSB は累算器を通り、ポインタの LSB は X レジスタを通ります。

戻り値：

なし

副作用

最後の文字が TX8 送信バッファに読み込まれるまで、プログラムフローは、この関数内にとどまります。A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリーモデル (CY8C27x66、CY8C29xxx、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。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

TX8_PutChar

説明

ポートバッファが空のときに、TX8 ポートに単一の文字を書き込みます。

C プロトタイプ：

```
void TX8_PutChar(CHAR cData)
```

アセンブラ：

```
mov  A,0x33            ; Load ASCII character “3” in A
lcall TX8_PutChar       ; Call function to send single character to
                       ; TX8 port.
```

パラメータ：

CHAR cData: TX8 ポートに送信する文字。データは累算器を通過します。

戻り値：

なし

特殊作用：

データを TX8 バッファに書き込めるようになるまで、プログラムフローはこの関数内にとどまります。A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリーモデル (CY8C27x66、CY8C29xxx、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。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

TX8_Write

説明

「n」文字（RAM）を TX8 ポートに送信します。

C プロトタイプ：

```
void TX8_Write(char * szRamString, BYTE bCount)
```

アセンブラ：

```
mov   A,20              ; Load string/array count
push  A
mov   A,>szRamString    ; Load MSB part of pointer to RAM string
push  A
mov   A,<szRamString    ; Load LSB part of pointer to RAM string
push  A
mov   X,SP              ; Set X register to point to variables
dec   X
lcall  TX8_Write         ; Make call to function
add   SP,253            ; Reset stack pointer to original position
```

パラメータ：

CHAR * szRamString: TX8 ポートに送信されるストリングのポインタ。

BYTE bCount: TX8 に送信される文字数。

戻り値：

なし

特殊作用：

最後の文字が TX8 バッファに読み込まれるまで、プログラムフローは、この関数内にとどまります。A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリーモデル (CY8C27x66、CY8C29xxx、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。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。現時点では , IDX_PP ページ ポインタ レジスタのみが変更されています。

TX8_CWrite

説明

「n」文字（ROM）を TX8 ポートに送信します。

C プロトタイプ：

```
void TX8_CWrite(const char * szRomString, INT iCount)
```

アセンブラ：

```
mov   A,0               ; Load MSB of string/array count
push  A
mov   A,20              ; Load LSB of string/array count
push  A
mov   A,>szRomString    ; Load MSB part of pointer to ROM string
push  A
```

```
mov   A,<szRomString     ; Load LSB part of pointer to ROM string
push  A
mov   X,SP               ; Set X register to point to variables
dec   X
lcall  TX8_CWrite         ; Make call to function
add   SP,252             ; Reset stack pointer to original position
```

パラメータ :

const char * szRomString: TX8 ポートに送信されるストリングのポインタ。

int iCount: TX8 に送信される文字数。

戻り値 :

なし

特殊作用 :

最後の文字が TX8 バッファに読み込まれるまで、プログラムフローは、この関数内にとどまります。A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリーモデル (CY8C27x66、CY8C29xxx、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。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

TX8_PutSHexByte

説明

16 進法表記による 2 バイトの ASCII 文字を TX8 ポートに送信します。

C プロトタイプ :

```
void TX8_PutSHexByte(BYTE bData)
```

アセンブラ :

```
mov  A,0x33               ; Load data to be sent to TX8
lcall TX8_PutSHexByte      ; Call function to output hex
                          ; representation of data.  The output
                          ; for this value would be "33".
```

パラメータ :

BYTE bData: ASCII 文字列に変換されるバイト。

戻り値 :

なし

特殊作用 :

最後の文字が TX8 バッファに読み込まれるまで、プログラムフローは、この関数内にとどまります。A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリーモデル (CY8C27x66、CY8C29xxx、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。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

TX8_PutSHexInt

説明

16 進法表記による 4 バイトの ASCII 文字を TX8 ポートに送信します。

C プロトタイプ :

```
void TX8_PutSHexInt(INT iData)
```

アセンブラ :

```
mov  A,0x34           ; Load LSB in A
mov  X,0x12           ; Load MSB in X

lcall TX8_PutSHexInt   ; Call function to output hex
                      ; representation of data.  The output
                      ; for this value would be “1234”.
```

パラメータ :

int iData: ASCII 文字列に変換される整数。

戻り値 :

なし

特殊作用 :

最後の文字が TX8 バッファに読み込まれるまで、プログラムフローは、この関数内にとどまります。A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリーモデル (CY8C27x66、CY8C29xxx、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。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

TX8_PutCRLF

説明

キャリッジリターン（0x0D）とラインフィード（0x0A）を TX8 ポートに送信する関数。

C プロトタイプ :

```
void TX8_PutCRLF(void)
```

アセンブラ :

```
lcall TX8_PutCRLF          ; Send a carriage return and line feed out TX
```

パラメータ :

なし

戻り値 :

なし

特殊作用 :

すべての文字が TX8 バッファに送信されるまで、プログラムの実行はこの関数内にとどまります。A および X レジスタは、今回、または今後、この関数を実装することによって変更される可能性があります。大容量メモリーモデル (CY8C27x66、CY8C29xxx、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。必要に応じて、fastcall16 関数の呼び出しでこれらの値を保存してください。

ファームウェア ソースコードの例

次のコードは、TX8 モジュールを用いて、RAM に存在するゼロ終端ストリングを送信する関数の例です。コードのサンプルをアセンブリ言語で示すと次のようになります。

```
;*****************************************************************
;  Name:   TxZeroTerminatedRamString
;
;  Description:
;    Transmits a zero terminated RAM-based string.
;
;  Parameters:
;    BYTE * pbStrPtr - pointer to the string - passed in
;    the X register.
;
;  Return:
;    None
;
;  Note:
;     TX8_Start should be called prior to calling this function.
;
;*****************************************************************
include "psocapi.inc"
export TxZeroTerminatedRamString

TxZeroTerminatedRamString:
.TxNextByte:
; check to see if end has been reached.
    mov   A, [X]
    jz    .AllDone

; transmit the next data byte
    call  TX8_SendData

.WaitForTxStart:
    ; wait for data to start transmitting
    call  TX8_bReadTxStatus
    and   A, TX8_TX_BUFFER_EMPTY
    jz    .WaitForTxStart
    ; increment the pointer to the next byte in the string
    inc   X
    ; data byte transmitted - send the next one
    jmp   .TxNextByte

.AllDone:
    ; string completely transmitted!
    ret
```

C で書かれた同じコードは以下のようになります。

```
//
// TX8_Start() should be called prior to calling this function.
//
#include "psocapi.h"
#include "m8c.h"


void TxZeroTerminatedRamString( BYTE *pbStrPtr )
{
    /* check for the end condition, before sending the next byte */
    while( *pbStrPtr != 0 )
    {
        /* send the next byte */
        TX8_SendData( *pbStrPtr );

        /* Wait for the data to start transmitting */
        while( !( TX8_bReadTxStatus() & TX8_TX_BUFFER_EMPTY ) );

        pbStrPtr++;
    }
}
```

コンフィグレーション レジスタ

TX8 ユーザーモジュールを構成するの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は下に示されています。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 TX、レジスタ : 関数

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	1	1	0	1

このレジスタは、TX8 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 5. ブロック TX、レジスタ : 入力

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	クロック			

クロックは、トランスミッタのタイミングを管理するために選択されたクロックです。これは、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 TX、レジスタ : 出力

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	TX 出力バス		

出力イネーブルと選択は、TX8 の出力を指定します。これは、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 TX、データシフトレジスタ : DR0

ビット	7	6	5	4	3	2	1	0
値	TX8 シフトレジスタ							

TX8 シフト レジスタは、TX8 状態マシンハードウェアによって制御され、コンテンツをシフトし、最下位ビットを送信します。データは、TX8 状態マシンハードウェアによって、DR1 データレジスタからこのレジスタに読み込まれます。

Table 8. ブロック TX、データバッファレジスタ : DR1

ビット	7	6	5	4	3	2	1	0
値	TX8 バッファレジスタ							

TX8 バッファレジスタは、ユーザ モジュール API によってこのレジスタに書き込まれるデータの送信に使用します。このレジスタに読み込まれたデータは、TX 状態マシンによって TX シフト レジスタに転送されます。

Table 9. ブロック TX、データレジスタ : DR2

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 10. ブロック TX、制御 / 状態レジスタ : CR0

ビット	7	6	5	4	3	2	1	0
値	予約	予約	TX 完了	TX レジスタ Empty (空)	予約	パリティタイプ	パリティイネーブル	TX イネーブル

TX イネーブルは、データバイトが送信処理中であるか否かを示すフラグです。このビットは、レジスタが読み込まれるとリセットされます。これは API 関数の一つによってアクセスできます。Tx Reg Empty は、バッファレジスタが空であることを示すフラグです。これは API 関数の一つによって評価できます。Parity Type は計算するパリティタイプです。このビットは、「パリティのイネーブル ビットがセットされていなくてもかまわない」というビットです。API 関数の 1 つによって、またはデバイス エディタから設定できます。Parity Enable は、データタイプを用いたパリティビットの計算と送信を有効· 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。API 関数の 1 つによって、またはデバイス エディタから設定できます。Tx Enable は TX トランスミッタを有効· 無効にします。これは API 関数の一つによって設定できます。

バージョン ヒストリー

バージョン	著者	説明
3.3	TDU	クロックの説明に関する更新：ブロックで外部デジタル クロックを使用している場合は、最高の精度およびスリープ動作を得るため、未処理の入力同期がオフになります。
3.4	DHA	外部クリスタルが使われない場合、ユーザーに 32 kHz オプションを使用してはいけないことを通報するために追加された DRC。
3.50	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簡単な解説を掲載しています。

Copyright © 2001-2011 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.