8-Bit UART データシート UART V 5.3

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	
CY8C29/27/24/22/21xxx、CY8C23x33、CY7C64215、CYWUSB6953、CY8CLED02/04/08/16、CY8CLED0xD、CY8CLED0xG、CY8CTST110、CY8CTMG110、CY8CTST120、CY8CTMG120、CY8CTMA120、CY8C21x45、CY8C22x45、CY8CTMA30xx、CY8C28x45、CY8CPLC20、CY8CLED16P01、CY8C28x43、CY8C28x52						
ローレベル API	2			311	0	2
ハイレベル API	2			506	3 + バッファ	2
CY8C26/25xxx						
ローレベル API	2			351	0	2
ハイレベル API	2			546	3 + バッファ	2

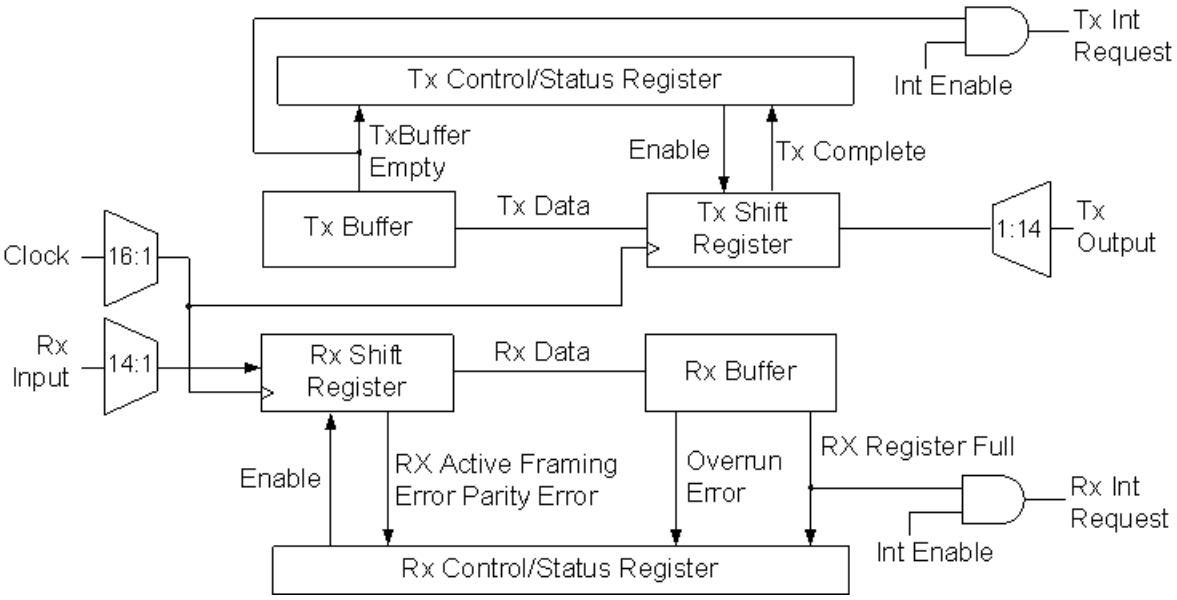
このユーザ モジュールを使用していくつかの設定をするサンプルプログラムは、以下で利用できます。
www.cypress.com/psocexampleprojects

特長および概要

- 非同期のレシーバとトランスミッタ
- R-232 シリアルデータ形式に準拠したデータ形式
- 最大 6 Mbit/ 秒のバースト伝送速度
- スタートビット、オプションパリティビット、ストップビットで構成されるデータフレーミング
- 受信レジスタがフル、送信バッファが空のときにオプションで割り込み
- パリティ、オーバラン、フレーミングエラーの検出
- ハイレベルの送信および受信関数

UART ユーザ モジュールは、二重 RS-232-compliant データ形式シリアル通信を 2 線でサポートする、8-bit ユニバーサル非同期レシーバ トランスミッタです。受信および送信されるデータ形式には、スタートビット、オプションのパリティ、ストップビットが含まれます。プログラマブルなクロックと、選択可能な割り込みやポーリング方式の操作をサポートしています。UART の初期化、構成、操作を行うための、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 ルーチンが提供されています。バックグラウンドのコマンド受信と文字列プリントをサポートする、その他のハイレベル API も用意されています。

Figure 1. UART ブロック ダイアグラム



機能説明

UART ユーザ モジュールはシリアル トランスミッタおよびレシーバを実装します。UART は PSoC Designer デバイス エディタで、2 つの PSoC ブロックを TX および RX ブロック にマッピングします。TX PSoC ブロックはトランスミッタの機能を、また RX PSoC ブロックはレシーバの機能を持ちます。

RX および TX は、それぞれ個別に動作します。またそれぞれが、独自の制御およびステータスレジスタ、プログラマブルな割り込み、I/O、バッファ レジスタ、シフト レジスタを持ちます。同じイネーブル、クロック、データ形式を共有します。

RX 制御および TX 制御レジスタでイネーブル ビットを設定すると、UART の操作が有効になります。有効または無効にする操作は、API の関数を使って行います。

UART ユーザ モジュールのクロックは、RX と TX の両コンポーネントで共有されます。選択したクロック周波数は、必要なデータ ビット レートの周波数の 8 倍になる必要があります。受信および送信したデータ ビットには、8 入力クロック サイクルが必要です。クロックは、PSoC Designer デバイス エディタを使って設定されます。

受信および送信されたデータは、スタートビット、8 つのデータビット、オプションのパリティビット、ストップビットから成るビットストリームです。パリティは、PSoC Designer デバイス エディタまたは UART API のいずれかを使って、なし、偶数、奇数に設定できます。RX および TX はともに、同じパリティに設定されます。

TX - UART トランスミッタ

トランスミッタは、デジタルコミュニケーションタイプの PSoC ブロックの TX バッファ、TX シフト、TX 制御レジスタを使用します。

TX 制御レジスタは、UART ユーザ モジュールのファームウェア API ルーチンを使って、初期化および構成されます。TX 制御レジスタのイネーブル ビットが設定されると、内部の 8 分周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 ルーチンによって送信バッファ レジスタに書き込まれ、TX 制御レジスタで、送信バッファの空ステータス ビットがクリアされます。このステータスビットは、送信オーバラン 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は、シフト レジスタにデータを送信し、TX 制御レジスタで送信バッファ空ビットをセットします。割り込みイネーブル マスクが有効になっている場合は、割り込みがトリガされます。この割り込みにより、送信する次のバイトの待機が可能になるとともに、現在のデータ バイトの送信が完了すると、次に利用可能な送信クロックで新しいデータ バイトが送信されます。

スタートビットは、データ バイトが送信バッファ レジスタから TX シフト レジスタに送信されるときに送信されます。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、TX 制御レジスタで TX 完了ステータス ビットが設定されます。このビットは、読み取りまで有効になります。新しいデータ バイトが送信バッファ レジスタに書き込まれると、データ バイトが TX シフト レジスタに送信され、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開始されます。

RX - UART レシーバ

レシーバは、デジタル通信タイプの PSoC ブロックの受信バッファ、RX シフト、RX コントロールレジスタを使用します。

RX 制御レジスタは、UART ユーザ モジュールのファームウェア API ルーチンを使って、初期化および構成されます。RX の初期化は、UART パリティの設定、Rx レジスタ フル条件での割り込みイネーブル (オプション)、UART のイネーブルの順で行われます。

スタートビットが RX 入力で検出されると、8 分周ビットクロックが開始され、受信ビット中央のデータをサンプリングするために同期化されます。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、RX シフト レジスタに移動します。パリティが有効な場合は、次のビットクロックが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ると、受信したデータ バイトを受信バッファ レジスタに送信し、以下のイベントがトリガされます。

- RX 制御レジスタで Rx レジスタ フルのビットが設定され、RX への割り込みが有効な場合は、関連付けられた割り込みがトリガされます。
- データストリーム内の予想ビット位置でストップビットが検出されない場合は、RX 制御レジスタ内でフレーミング エラー ビットが設定されます。
- 現在受信されたデータのストップビットより前にバッファ レジスタが読み取られなかった場合は、RX 制御レジスタでオーバラン エラー ビットが設定されます。
- パリティエラーが検出されると、RX 制御レジスタでパリティ エラー ビットが設定されます。

完全に受信されたデータ バイトをポーリング検出するには、RX 制御レジスタの Rx レジスタ フル ビットをモニタする必要があります。オーバラン エラーを防ぐには、次のバイトが完全に受信される前に、受信バッファ レジスタからデータを読み取らなければなりません。

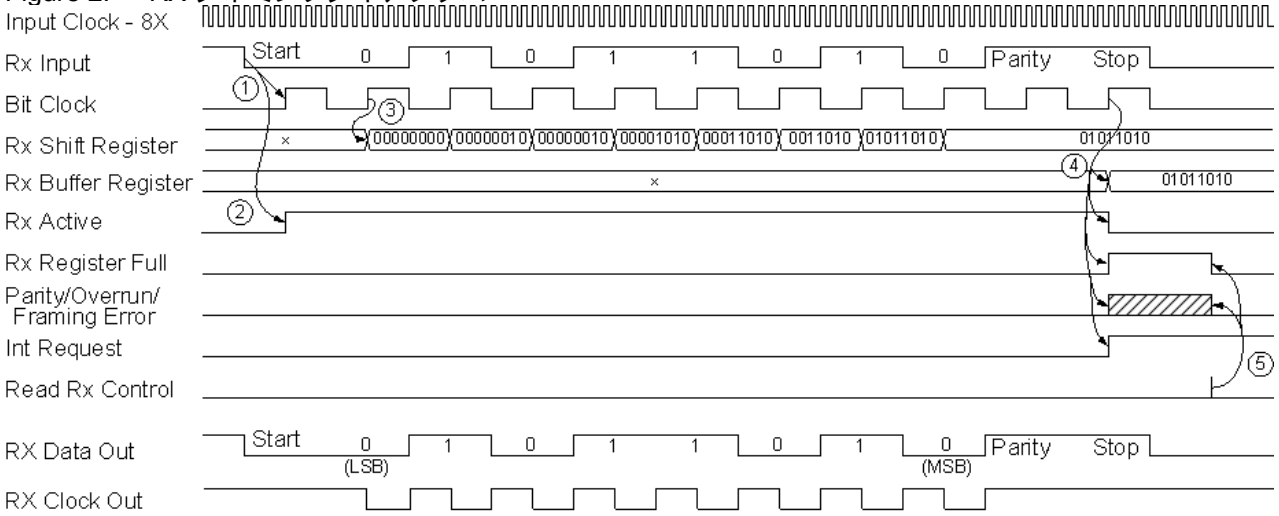
コミュニケーション システムの精度

PSoC デバイスの SLIMO モードを使用する場合は、PSoC システムクロック (SysClk) は、有効な UART 通信を保証できるほど正確ではなくなります。また、PSoC が USB に接続されていない場合は、PSoC CY8C24x94 ファミリ デバイスの SysClk は、有効な UART 通信を保証できるほど正確ではなくなります。UART 通信が正しく動作するためには、システムエラー、つまり通信リンクの両側のエラー合計が 2% 未満になる必要があります。SysClk の精度に関する詳しい情報は、デバイスのデータシートを参照してください。

タイミング

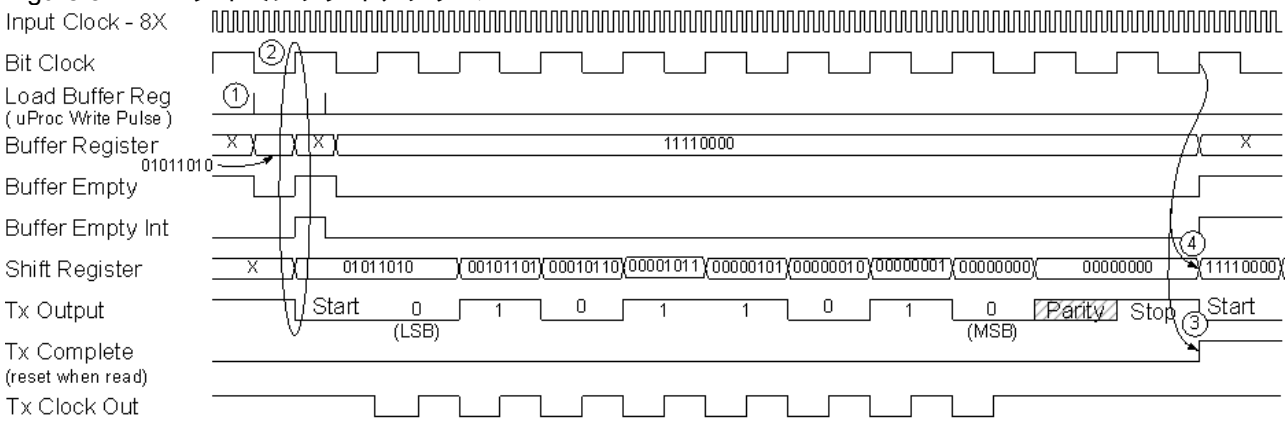
次の RX タイミング図は、UART ユーザ モジュールの RX コンポーネントの動作を示しています。

Figure 2. RX タイミングダイアグラム



次の TX タイミング図は、UART ユーザ モジュールの TX コンポーネントの動作を示しています。

Figure 3. TX タイミングダイアグラム



DC 電気的特性と AC 電気的特性

Table 1. UART 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}	最大文字送信周波数	--	6	Mbits

配置

RX ユーザ モジュールは、どの 2 つのデジタルコミュニケーションブロックにでも配置できます。レシーバおよびトランスミッタの両コンポーネントで、同じクロック源が使用されることに注意してください。

パラメータおよびリソース

クロック

UART は 16 個のクロック源のいずれかを使用します。外部のピンまたは異なる PSoC ブロックが生成するクロック関数にクロック入力を接続する場合は、グローバル I/O バスを使用することが可能です。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。48 MHz クロック、CPU_32 kHz クロック、分周クロックのいずれか、24V1 または 24V2、別の PSoC ブロックの出力を、クロック入力として指定できます。

クロックレートは、希望するビット レートの 8 倍に設定する必要があります。1 データ ビットは、8 クロック サイクルごとに受信または送信されます。正しく機能するには、クロックの許容値が ± 2% となる必要があります。

RX 入力

レシーバの入力は、下位、上位、近接の PSoC ブロック、アナログ コンパレータ出力バス、グローバル バスのいずれかに接続できます。グローバル バスを使って、入力を外部ピンのいずれかに接続できます。

反転 RX 入力

このパラメータを使うと、RX 入力信号を反転できます。

TX 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が生成されるタイミングを決定します。データ レジスタからシフト レジスタにデータが送信されるとすぐに、「TxRegEmpty」オプションは、割り込みを生成します。2 つ目のオプション 「TxComplete」を選ぶと、最後のビットがシフト レジスタからシフトされるまで、割り込みが遅延されます。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限に引き上げるのに最適です。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、デジタル ブロックは、システムクロック以外のクロック源となります。デジタル クロック源は、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源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)、SysClk ベースのソースを持つデジタル PSoC ブロックが挙げられます。正しい同期が行われるよう、外部で生成されたクロック源で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 指示の使用	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 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。
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。

RX 出力

このパラメータを使うと、入力信号をローバスのいずれかにルーティングできます。データクロックアウトと一緒に、この信号は循環重複検査といったデータ検証機能を容易にします。

RX クロックアウト

このパラメータを使うと、RX ブロックからのビットクロックを、ローバスのいずれかにルーティングできます。このビットクロックは 8 で除算したクロック入力信号です。データクロックアウト信号の立ち上がりエッジは、データが安定しサンプリングされる時間と一致します。RX 出力に伴う信号は、巡回冗長検査回路などのデータ検証機能を実行する際に使用することができます。

TX クロックアウト

このパラメータを使うと、TX ブロックからのビットクロックを、ローバスのいずれかにルーティングできます。このビットクロックは 8 で除算したクロック入力信号です。データクロックアウト信号の立ち上がりエッジは、データが安定しサンプリングされる時間と一致します。TX 出力に伴う信号は、巡回冗長検査回路などのデータ検証機能を実行する際に使用することができます。

RxCmdBuffer

このパラメータは、コマンド処理に使用される受信コマンド バッファおよびファームウェアを有効にします。UART RX 割り込みは、操作するコマンド バッファで有効にする必要があります。

RxBufferSize

このパラメータを使うと、受信バッファでいくつの RAM 位置を予約するかを指定できます。受信可能な最大コマンド数は、文字列をすべてヌルで終端する必要があるため、選択したバッファ サイズより 1 小さくなります。RxCmdBuffer が有効で、UART RX 割り込みが有効な場合にのみ、このパラメータが有効になります。

CommandTerminator

このパラメータは、コマンドの最後の部分を示す文字を選択します。受信されると、コマンド全体が受信されたことを示すフラグが設定されます。このフラグが設定されると、cmdReset() 関数が呼び出されるまで、追加の文字はそれ以上受け付けられません。

Param_Delimiter

このパラメータは、コマンド レシーバ バッファでコマンドおよびパラメータを区切るのに使用される文字を選択します。例えば、Param_Delimiter をスペース文字 (32) に設定した場合、スペースで区切られた部分文字列がそれぞれ 1 つのパラメータになります。このとき文字列 「cmd foo bar c」では、「cmd」、「foo」、「bar」、「c」がそれぞれパラメータになります。szGetParam() が呼び出されるたびに、ヌル終端文字列として、左から右の順に、次の部分文字列へのポインタを返します。

IgnoreCharsBelow

このパラメータは、受信バッファが設定値以下の文字列を無視するよう設定します。この文字は受信されますが、受信バッファには追加されません。このパラメータは、RxCmdBuffer が有効で、UART RX 割り込み信号が有効な場合にのみ有効です。

Enable_BackSpace

このパラメータは、バックスペースまたは削除文字で UART 受信バッファの最後の文字を削除するよう設定します。「Disable (無効)」、「Backspace (バックスペース)」、「Delete (削除)」のいずれかを設定できます。このパラメータは、RxCmdBuffer が有効で、UART RX 割り込み信号が有効な場合にのみ有効です。

割り込み生成制御

InterruptAPI および IntDispatchMode の 2 つのパラメータには、PSoC Designer の **[Enable interrupt generation control (割り込み生成の制御を有効にする)]** チェックボックスを設定することでのみアクセスできます。これは **[Project (プロジェクト)] > > [Settings (設定)] > > [Chip Editor (チップ エディタ)] >** で利用できます。

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。受信コマンド バッファを使用する場合は、InterruptAPI パラメータを 「Enable (有効)」に設定する必要があります。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。必要な場合にのみ InterruptAPI の生成を選択すると、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、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、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機能に対するインタフェースおよび定数を示します。

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。これらの値が呼び出し後に必要となる場合は、呼び出す関数を使って、呼び出し前に A および X の値を維持してください。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。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。

以下の表に、UART によるローレベル API 関数を示します。

Table 2. ローレベル UART API

関数	説明
無効な UART_Start(BYTE bParity)	ユーザ モジュールを有効にして、パリティを設定します。
void UART_Stop(void)	ユーザ モジュールを無効にします。
void UART_EnableInt(void)	RX および TX 割り込みの両方を有効にします。
void UART_DisableInt(void)	RX および TX 割り込みの両方を無効にします。
void UART_SetTxIntMode(BYTE bTxIntMode)	Tx 割り込みのソースを設定します。
void UART_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 UART_bReadTxStatus(void)	TX 状態レジスタにステータスを返します。
BYTE UART_bReadRxData(void)	文字状態が有効かどうかをチェックせず、RX データレジスタのデータを変換します。
BYTE UART_bReadRxStatus(void)	RX 状態レジスタの状態をチェックします。

UART_Start

説明

パリティを設定し、UART レシーバおよびトランスミッタを有効にします。有効になると、データを受信および送信できるようになります。

C プロトタイプ :

```
void  UART_Start(BYTE bParitySetting)
```

アセンブリ :

```
mov   A, UART_PARITY_NONE
lcall  UART_Start
```

パラメータ :

bParitySetting: 伝送パリティを指定する 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を以下の表に示します。

TX パリティ	値
UART_PARITY_NONE	0x00
UART_PARITY_EVEN	0x02
UART_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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_Stop

説明

UART モジュールを無効にします。

C プロトタイプ :

```
void  UART_Stop(void)
```

アセンブリ :

```
lcall  UART_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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_EnableInt

説明

適切な割り込みイネーブル ビットを設定することで、両方の UART 割り込みを設定し、有効にします。

C プロトタイプ :

```
void  UART_EnableInt(void)
```

アセンブリ :

```
lcall  UART_EnableInt
```

パラメータ :

なし

戻り値 :

なし

特殊作用 :

割り込みが保留されているときにこの API が呼び出されると、割り込みが直ちにトリガされます。Start() を呼び出す前に、これを呼び出す必要があります。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_DisableInt

説明

すべての UART 割り込みを無効にします。

C プロトタイプ :

```
void  UART_DisableInt(void)
```

アセンブリ :

```
lcall  UART_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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_SetTxIntMode

説明

Tx 割り込みのソースを設定します。

C プロトタイプ :

```
void  UART_SetTxIntMode(BYTE bTxIntMode)
```

アセンブリ :

```
lcall  UART_SetTxIntMode
```

パラメータ :

bTxIntMode: Tx ブロックの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を以下の表に示します。

TX 割り込みモード	値
UART_INT_MODE_TX_REG_EMPTY	0x00
UART_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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_SendData

説明

送信するよう指定されたデータとともに送信バッファ レジスタをロードして、データの送信を開始します。送信が開始されたことを確認するため、TX 制御レジスタで TX 完了ステータス ビットをモニタしてください。

C プロトタイプ :

```
void  UART_SendData(BYTE bTxData)
```

アセンブリ :

```
mov   A, bTxData
lcall  UART_SendData
```

パラメータ :

bTxData: データが送信バッファ レジスタにロードされ、アキュムレーターに渡されます。

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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_bReadTxStatus

説明

TX 制御レジスタの内容を返します。

C プロトタイプ :

```
BYTE  UART_bReadTxStatus(void)
```

アセンブリ :

```
lcall  UART_bReadTxStatus
and   A, UART_TX_COMPLETE
```

```
jnz   TxIsComplete
```

パラメータ :

なし

戻り値 :

TX ステータスバイトを返します。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、結合条件を確認できます。

RX ステータス マスク	値
UART_TX_COMPLETE	0x20
UART_TX_BUFFER_EMPTY	0x10

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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_bReadRxData

説明

受信バッファ レジスタから、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  UART_bReadRxData(void)
```

アセンブリ :

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

パラメータ :

なし

戻り値 :

受信し、累算器で渡されたデータ バイト。

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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_bReadRxStatus

説明

RX 制御レジスタを読み取って返します。

C プロトタイプ :

```
BYTE  UART_bReadRxStatus(void)
```

アセンブリ :

```
lcall  UART_bReadRxStatus
push   A
and    A, UART_RX_COMPLETE
pop    A
jz     RxNotCompleted
cmp    A, UART_RX_ERROR              ; determine if RX was without error
jz     RxCompleteWithoutError        ; receive was completed
```

パラメータ :

なし

戻り値 :

RX ステータス バイトを返します。定義された以下のマスクを使用し、特定のステータス条件をテストしてください。マスクを OR 処理することで、結合条件をチェックできます。

RX ステータス マスク	値	説明 (ビットが設定された場合)
UART_RX_REG_FULL	0x08	データ レジスタには、読み取られていないデータが含まれます。
UART_RX_PARITY_ERROR	0x80	ステータスが最後に読み取られた時点以降のパリティ エラー。
UART_RX_OVERRUN_ERROR	0x40	データがバッファ レジスタに書き込まれました。
UART_RX_FRAMING_ERROR	0x20	ストップビットが下位でした。
UART_RX_ERROR	0xE0	OR で結合されたすべてのエラー ビット。

特殊作用 :

このレジスタを読み取ると、ステータス ビットがすべてクリアされます。ただし UART_RX_REG_FULL は例外で、データ レジスタが直接読み取られた場合、または API 関数を使って読み取られた場合にのみクリアされます。戻り値を破棄する前に、該当するすべてのステータス条件をチェックする際には十分注意してください。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

ハイレベル API

ハイレベル API は、基本関数に加えて、文字レベルではなくコマンドと文字列レベルの関数を提供するよう、ファームウェアを追加します。デバイス エディタを使うと、レシーバのコマンド バッファ、コマンド終端文字、パラメータ区切り文字、レシーバが文字を無視する値の下限を設定できます。送信関数は文字列へのポインタを受け入れるため、C または ASM コードで行を追加するのではなく、1 つの関数呼び出しを使って文字列全体をプリントすることができます。送信関数は、UART 送信割り込みを有効にする必要はありません。

ハイレベルの受信関数を使うには、デバイス エディタのウィンドウを開き、UART を選択してから「RxCmdBuffer」パラメータの「Enable (有効)」オプションを選択してください。次に、最大コマンド + 1 を保持するのに十分な大きさの 「RxBufferSize」を選択します。コマンド終端文字「CommandTerminator」を選択します。これは、復帰文字 (13) または改行文字 (10) のいずれかに設定するのが普通です。コマンドに複数のパラメータが含まれている場合は、パラメータ区切り文字「Param_Delimiter.」を選択してください。通常、スペース (32) またはコンマ (44) 文字になります。選択された値以下の制御文字を無視するよう設定することも可能です。ほとんどの制御文字は 0 ー 31 の範囲内にあります。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。このパラメータを 「1」に設定すると、すべての文字が有効になります。コマンド終端文字 (CommandTerminator) として選択した文字は、「IgnoreCharsBelow」オプションには影響されません。以下のフローチャートに、コマンド バッファ関数の基本操作の正しい順番を示します。

コマンド終端文字が受信されるまで、UART RX 割り込みサービス ルーチンでコマンド バッファが動作し、バッファ内の文字が収集されます。この時点で、コマンド バッファを読み取る準備ができたことを示すフラグが設定されます。アレイ INSTANCE_NAME_aRxBuffer を読み取るか、szGetParam() または szGetRestOfParams() 関数のいずれかを使用して、受信バッファに直接アクセスすることができます。buffer_size -1 を超える文字が受信された場合、それに続く文字は無視されます。

コマンド バッファは、バッファが完全に埋まるか、コマンド終端文字が検出されるまで文字を収集します。この 2 つの条件のいずれかが満たされると、CmdReset コマンドが実行されるまで、以降に受信されたすべての文字が無視されます。CmdReset コマンドが実行されると、RX ISR ファームウェアが文字を再度収集し始めます。

Figure 4. コマンド バッファ フロー

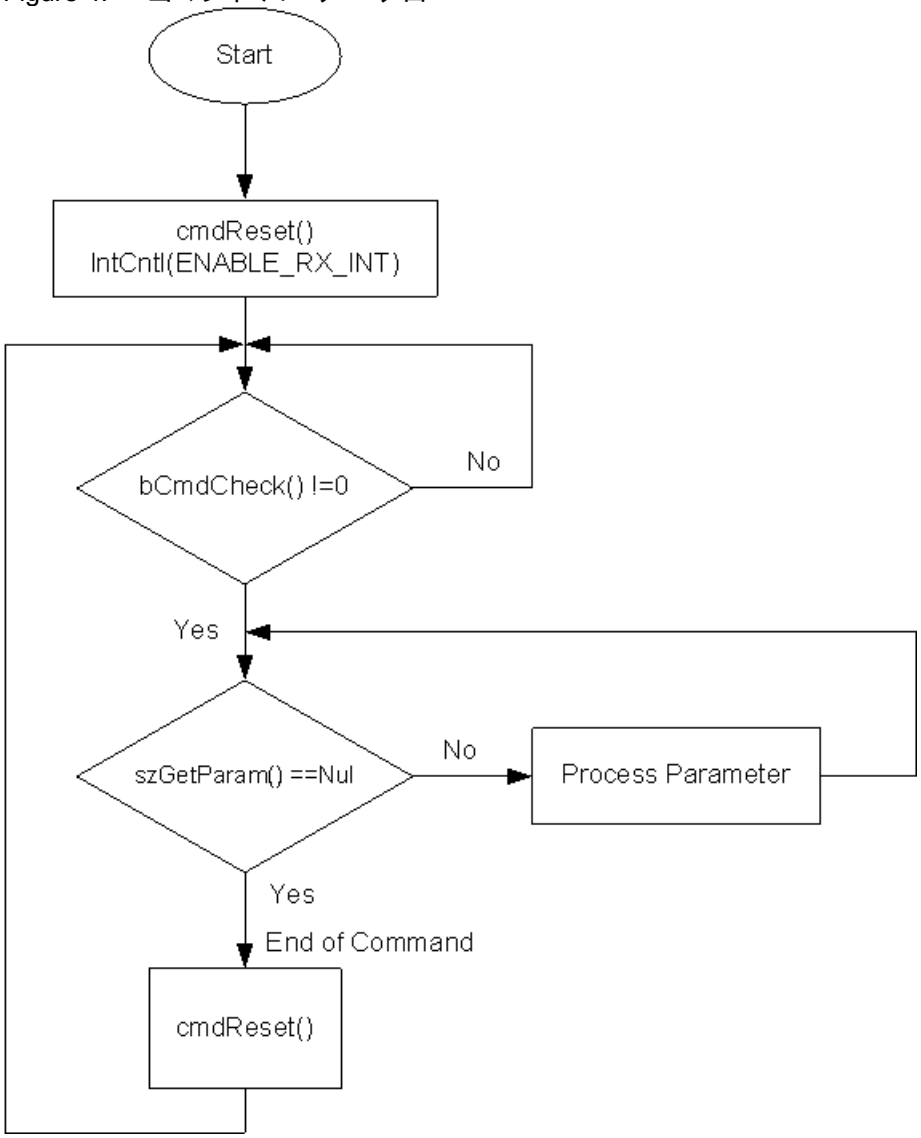


Table 3. ハイレベル UART API

関数	説明
void UART_IntCntl(BYTE bMask)	RX および TX の割り込みを有効または無効にします。
void UART_PutString(char * szStr)	TX ポートに NULL 終端された文字列を送信します。
void UART_CPutString(const char * azStr)	TX ポートに NULL 終端された定数 (ROM) 文字列を送信します。
void UART_PutChar(char bData)	TX レジスタが空の場合に、TX ポートに文字を送信します。データ オーバラン エラーが発生せずに TX データ レジスタへの書き込みが可能になるまで、関数は何も返しません。
char UART_cGetChar(void)	有効なデータがある場合は、RX データ レジスタからの文字を返します。文字が受信されるまで、関数は何も返しません。

関数	説明
void UART_Write(char * aStr, BYTE bCnt)	TX ポートに aStr アレイからの bCnt バイトを送信します。
void UART_CWrite(const char * aStr, int iCnt)	TX ポートに定数 aStr アレイからの iCnt バイトを送信します。
char UART_cReadChar(void)	RX データ レジスタを直ちに読み取ります。有効なデータがない場合は 0 を返し、データがある場合は 1 ～ 255 の ASCII 文字を返します。
int UART_ iReadChar(void)	Rx データ レジスタを直ちに読み取ります。データがない場合やエラー条件が存在する場合は、エラー状態を MSB で返します。受信した文字は LSB で返されます。
void UART_PutSHexByte(BYTE bValue)	TX ポートに bValue の 16 進数値を表す 2 文字を送信します。
void UART_PutSHexInt(int iValue)	TX ポートに iValue の 16 進数値を表す 4 文字を送信します。
void UART_PutCRLF(void)	TX ポートに、復帰文字 (0x0D) および改行文字 (0x0A) を送信します。
void UART_CmdReset(void)	Rx コマンド バッファをリセットします。
BYTE UART_bCmdCheck(void)	有効なコマンド終端文字が受信された場合、非ゼロ値を返します。
BYTE UART_bCmdLength(void)	現在のコマンドの長さを返します。
char * UART_szGetParam(void)	受信バッファの次のパラメータへのポインタを返します。
char * UART_szGetRestOfParams(void)	残りのパラメータ文字列へのポインタを返します。
BYTE UART_bErrCheck(void)	コマンド バッファ エラー ステータスを返します。

UART_IntCntl

説明

RX および TX 割り込みを別々に有効または無効にします。

C プロトタイプ :

```
void  UART_IntCntl(BYTE bMask)
```

アセンブリ :

```
mov   A,(UART_ENABLE_RX_INT|UART_ENABLE_TX_INT)
lcall  UART_IntCntl
```

パラメータ :

BYTE bMask：TX および RX ブロックを有効または無効にするマスク。

マスク	説明
UART_ENABLE_RX_INT	レシーバを有効にします。
UART_ENABLE_TX_INT	トランスミッタを有効にします。
UART_DISABLE_RX_INT	レシーバを無効にします。
UART_DISABLE_TX_INT	トランスミッタを無効にします。

戻り値 :

なし

特殊作用 :

TX または RX に保留中の割り込みがあり、割り込みが有効になっている場合は、割り込みが直ちに発生します。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_PutString

説明

UART TX に NULL で終端された (RAM) 文字列を送信します。

C プロトタイプ :

```
void UART_PutString(char * szRamString)
```

アセンブリ :

```
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 UART_PutString     ; Call function to send string out UART TX port
```

パラメータ :

char * aRamString: UART TX に送信される文字列へのポインタ。MSB は累算器で、LSB は X レジスタで渡されます。

戻り値 :

なし

特殊作用 :

プログラムのフローは、最後の文字が UART 送信バッファにロードされるまで、この関数に残ります。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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ます。

UART_CPutString

説明

UART TX ポートに、定数 (ROM)、NULL で終端された文字列を送信します。

C プロトタイプ :

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

アセンブリ :

```
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 UART_CPutString  ; Call function to send constant string out
                      ; UART TX
```

パラメータ :

const char * szROMString: UART に送信される文字列へのポインタ。文字列ポインタの MSB は累算器で、ポインタの LSB は X レジスタで渡されます。

戻り値 :

なし

特殊作用 :

プログラムのフローは、最後の文字が UART 送信バッファにロードされるまで、この関数に残ります。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_PutChar

説明

ポート バッファが空の場合に、UART TX ポートに 1 文字を書き込みます。

C プロトタイプ :

```
void UART_PutChar(CHAR cData)
```

アセンブリ :

```
mov  A,0x33            ; Load ASCII character "3" in A
lcall UART_PutChar      ; Call function to send single character to
                       ; UART TX port.
```

パラメータ :

CHAR cData: UART TX ポートに送信される文字。データは累算器で渡されます。

戻り値 :

なし

特殊作用 :

プログラム フローは、UART 送信バッファへのデータ書き込みが可能になるまで、この関数に残ります。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_cGetChar

説明

UART RX の有効文字を待ち、その値を返します。

C プロトタイプ :

```
CHAR UART_cGetChar(void)
```

アセンブリ :

```
lcall UART_cGetChar         ; Call 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: UART RX から読み取られた文字は、累算器に返されます。

特殊作用 :

プログラム フローは、文字が受信されるか、または受信されたが読み取られていない状態になるまで、この関数に残ります。この関数を使用する場合は、UART RX 割り込みを無効にする必要があります。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_Write

説明

UART TX ポートに 「n」文字 (RAM) を送信します。

C プロトタイプ :

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

アセンブリ :

```
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  UART_Write        ; Make call to function
add   SP,253            ; Reset stack pointer to original position
```

パラメータ :

CHAR * szRamString: UART TX に送信される文字列へのポインタ。

BYTE bCount: UART TX に送信される文字数。

戻り値 :

なし

特殊作用 :

プログラム フローは、最後の文字が UART 送信バッファにロードされるまで、この関数に残ります。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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ます。

UART_CWrite

説明

UART TX ポートに 「n」文字 (ROM) を送信します。

C プロトタイプ :

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

アセンブリ :

```
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  UART_CWrite       ; Make call to function
add   SP,252            ; Reset stack pointer to original position
```

パラメータ :

const char * szRomString: UART TX ポートに送信される文字列へのポインタ。

int iCount: UART TX ポートに送信される文字数。

戻り値 :

なし

特殊作用 :

プログラム フローは、最後の文字が UART 送信バッファにロードされるまで、この関数に残ります。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_cReadChar

説明

データがない場合やエラー条件が存在する場合、ゼロが返された場合は、UART RX ポートを直ちに読み取ってください。直ちに読み取らないと、文字が読み取られて返されます。

C プロトタイプ :

```
CHAR UART_cReadChar(void)
```

アセンブリ :

```
lcall UART_cReadChar          ; Call 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: UART RX ポートから読み取られた文字。1 ～ 255 までの ASCII 文字が有効です。ゼロが返された場合は、エラー条件またはデータがないことを示します。

特殊作用 :

関数は、1 ～ 255 の文字のみを有効な文字として受け入れます。0x00 (NULL) 文字は、エラー条件として検出されます。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_iReadChar

説明

UART RX ポートを直ちに読み取り、受信された文字とエラー条件を返します。

C プロトタイプ :

```
INT UART_iReadChar(void)
```

アセンブリ :

```
lcall UART_iReadChar          ; Call 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 にはステータスが、LSB には UART RX データが含まれます。MSB がゼロでない場合は、エラーが発生したことを意味します。この表に、MSB で返される可能性のあるエラーコードを示します。

エラー フラグ	値	説明
UART_RX_PARITY_ERROR	0x80	パリティ エラー
UART_RX_OVERRUN_ERROR	0x40	バッファ オーバラン エラー
UART_RX_FRAMING_ERROR	0x20	文字フレーミング エラー
UART_RX_NO_DATA	0x10	受信バッファフル

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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_PutSHexByte

説明

データを ASCII 16 進数で表した 2 バイトを、UART TX ポートに送信します。

C プロトタイプ :

```
void UART_PutSHexByte(BYTE bData)
```

アセンブリ :

```
mov  A,0x33               ; Load data to be sent to UART TX
lcall UART_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 :

プログラムのフローは、最後の文字が UART 送信バッファにロードされるまで、この関数に残ります。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_PutSHexInt

説明

データを ASCII 16 進数で表した 4 バイトを、UART TX ポートに送信します。

C プロトタイプ :

```
void UART_PutSHexInt(INT iData)
```

アセンブリ :

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

lcall UART_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 :

プログラムのフローは、最後の文字が UART 送信バッファにロードされるまで、この関数に残ります。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_PutCRLF

説明

UART TX ポートに、復帰文字 (0x0D) および改行文字 (0x0A) を送信する関数。

C プロトタイプ :

```
void UART_PutCRLF(void)
```

アセンブリ :

```
lcall UART_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 :

プログラムの実行は、すべての文字が UART 送信バッファに書き込まれるまで、この関数に残ります。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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。

UART_CmdReset

説明

コマンド バッファおよびフラグをリセットします。これにより、次のコマンドの文字を受け入れることが可能になります。

C プロトタイプ :

```
void UART_CmdReset(void)
```

アセンブリ :

```
lcall UART_CmdReset         ; Call function to reset command buffer.
```

パラメータ :

なし

戻り値 :

なし

特殊作用 :

UART 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文字は失われます。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM

ページ ポインタでも同じです。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、CUR_PP ページ ポインタ レジスタのみが変更されます。

UART_bCmdCheck

説明

コマンド終端文字が受信されたかどうかをチェックします。

C プロトタイプ :

```
BYTE UART_bCmdCheck(void)
```

アセンブリ :

```
lcall UART_bCmdCheck      ; Call 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 :

コマンド終端文字が受信された場合、非ゼロ値を返します。

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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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、CUR_PP ページ ポインタ レジスタのみが変更されます。

UART_bCmdLength

説明

コマンド バッファの文字の長さを返します。このコマンドは、コマンド終端文字が受信されたかどうかに関係なく現在のコマンドの長さを返します。

C プロトタイプ :

```
BYTE UART_bCmdLength(void)
```

アセンブリ :

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

パラメータ :

なし

戻り値 :

受信バッファにある現在の文字列の長さ。

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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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、CUR_PP ページ ポインタ レジスタのみが変更されます。

UART_szGetParam

説明

デバイス エディタで設定した Param_Delimiter によって区切られた、受信バッファの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ると、以降のすべての呼び出しは NULL ポインタ (ゼロ) を返します。

C プロトタイプ :

```
char * UART_szGetParam(void)
```

アセンブリ :

```
lcall UART_szGetParam       ; Call 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: パラメータ文字列へのポインタ。

特殊作用 :

szGetParam が呼び出されるたびに、受信バッファが変更されます。NULL は各パラメータの後ろに配置されます。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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、CUR_PP および IDX_PP ページ ポインタ レジスタが変更されます。

UART_szGetRestOfParams

説明

szGetParam で返されていない、受信バッファ文字列の残りを指すポインタを返します。この関数が szGetParam の前に呼び出されると、受信バッファ全体へのポインタが返されます。文字列の最後に達すると、NULL (0x00) が返されます。

C プロトタイプ :

```
char * UART_szGetRestOfParams(void)
```

アセンブリ :

```
lcall UART_szGetRestOfParams    ; Call 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: 受信文字列の残りへのポインタ。

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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも

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、CUR_PP ページ ポインタ レジスタのみが変更されます。

UART_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 UART_bErrCheck(void)
```

アセンブリ :

```
lcall UART_bErrCheck             ; Call 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 には、この関数が最後に呼び出された時点以降に発生した可能性のある、すべてのエラー条件のステータスが含まれます。

エラー フラグ	値	説明
UART_RX_PARITY_ERROR	0x80	パリティ エラー
UART_RX_OVERRUN_ERROR	0x40	バッファ オーバラン エラー
UART_RX_FRAMING_ERROR	0x20	文字フレーミング エラー
UART_RX_BUF_OVERRUN	0x10	ソフトウェア受信バッファのオーバラン

特殊作用 :

エラー ステータスがクリアされます。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 モデル (CY8C29xxx および CY8CLED16) の全 RAM ページ ポインタでも同じです。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、CUR_PP ページ ポインタ レジスタのみが変更されます。

ファームウェア ソースコードの例

次に、UART ユーザ モジュールのアセンブリ言語コード サンプルを示します。

```
;*****************************************************************
;  LoopBack:
;
;  This code sample illustrates how to setup a UART loopback.
;  Data received is echoed back to the receiver.  If an error
;  is received, then a NAK is returned.
;
;  This sample is written to be performed in the non-interrupt
;  processing.  This code could easily be written to be
;  interrupt driven.
;  In this example a Counter8 user module is used to generate
;  this clock.  The following are the parameter settings for the
;  Counter8 User Module used for this example.
;  Counter8  (placed in DBA03)
;  Clock          48MHz
;  Enable         High
;  Output         None
;  Period         155
;  CompareValue   78
;  CompareType    Less Than Or Equal
;  InterruptType Terminal Count
;
;  Using the settings above the UART baud rate can be calculated.
;  baud rate = 48MHz / ((Period+1) * (over Sample rate))
;  = 48MHz / ((155+1) * 8)
;  = 38.4 kBaud
;
;  The settings for the UART are as follows:
;
;  UART  (Placement RX => DAC06, TX => DCA07)
;  Clock              DBA03 Broadcast
;  RX Input           Global_IN_5
;  TX Output          Global_OUT_7
;  TX InterruptMode   TXComplete
;  RxCmdBuffer        Enable
;  RxBufferSize       16 Bytes
;  CommandTerminator  13      (CR)
;  Param_Delimiter    32      (space)
;  IgnoreCharsBelow   32      (Ignore all chars below space, except ;  CR)
;
;
;  First the UART and Counter8 user modules are initialized.  The
;  period and compare values for the Counter8 User Module are set to
;  generate the 8X clock.  If the period and compare values for
;  the Counter8 User Module are set in the parameter window, it is
;  not required to set them in the user program.
;
;*****************************************************************
include "m8c.inc"       ; part specific constants and macros
include "PSoCAPI.inc"   ; PSoC API definitions for all User Modules

export  _main
```

```
NAK_RESPONSE:   equ   00
_main:

    mov   A,UART_PARITY_NONE  ; No parity
    call  UART_Start
    call  Counter8_Start
.WaitForData:               ; wait for data to be received
    call  UART_bReadRxStatus
    and   A, UART_RX_COMPLETE
    jz    .WaitForData

    and   A, UART_RX_ERROR ; data received - see if data is valid
    jz    .GetData

    mov   A, NAK_RESPONSE  ; error detected setup to send a NAK
    jmp   .TxData
.GetData:

    call  UART_bReadRxData  ; read the data from the receiver
.TxData:

    call  UART_SendData     ; transmit the response data
    jmp   .WaitForData      ; go wait for next byte
```

次に、UART ユーザ モジュールの C コード サンプルを示します。

```
//-------------------------------------------------------------------
// Description:
//    This example program shows how easy it is to use the PSoC
//    serial port with this user module.
//
//    The UART requires a clock that is 8 times the desired Baud Rate.
//    In this example a Counter8 user module is used to generate
//    this clock.  The following are the parameter settings for the
//    Counter8 User Module used for this example.
//
//    Counter8  (placed in DBA03)
//      Clock         48MHz
//      Enable        High
//      Output        None
//      Period        155
//      CompareValue  78
//      CompareType   Less Than Or Equal
//      InterruptType Terminal Count
//
//    Using the settings above the UART baud rate can be calculated.
//    baud rate = 48MHz / ((Period+1) * (over Sample rate))
//              = 48MHz / ((155+1) * 8)
//              = 38.4 kBaud
//
//    The settings for the UART are as follows:
//
//    UART  (Placement RX => DAC06, TX => DCA07)
//      Clock              DBA03 Broadcast
```

```
//      RX Input           Global_IN_5
//      TX Output          Global_OUT_7
//      TX InterruptMode   TXComplete
//      RxCmdBuffer        Enable
//      RxBufferSize       16 Bytes
//      CommandTerminator  13      (CR)
//      Param_Delimiter    32      (space)
//      IgnoreCharsBelow   32      (Ignore all chars below space, except CR)
//
//
//    First the UART and Counter8 user modules are initialized.  The
//    period and compare values for the Counter8 User Module are set to
//    generate the 8X clock.  If the period and compare values for
//    the Counter8 User Module are set in the parameter window, it is
//    not required to set them in the user program, but are set here
//    to make it clear how the UART clock is generated. Next a
//    sample string is printed to welcome you to the program.  The
//    program then sits and waits for characters until a command
//    terminator is received (CR).  Once the command terminator is received
//    it parses the parameters and outputs them to the UART TX port.
//    The command buffer is then reset and waits for the next command.
//
//  Example:
//  Input:
//             "foobar aa bbb cc"(CR)
//
//  Output:
//
//    Welcome to PSoC UARTlus test program. V1.1
//    Found valid command
//    Command =>foobar<
//    Parameters:
//      <aa>
//      <bbb>
//      <c>
//
//-------------------------------------------------------------------
#include <m8c.h>
#include "PSoCAPI.h"

void main(void)
{

    char * strPtr;                      // Parameter pointer

    UART_CmdReset();                    // Initialize receiver/cmd
                                        // buffer
    UART_IntCntl(UART_ENABLE_RX_INT);   // Enable RX interrupts

    Counter8_WritePeriod(155);          // Set up baud rate generator
    Counter8_WriteCompareValue(77);
    Counter8_Start();                   // Turn on baud rate generator

    UART_Start(UART_PARITY_NONE);       // Enable UART
```

```
    M8C_EnableGInt ;                        // Turn on interrupts

    UART_CPutString("\r\nWelcome to PSoC UART test program. V1.1 \r\n");

    while(1) {
       if(UART_bCmdCheck()) {                     // Wait for command
          if(strPtr = UART_szGetParam()) {        // More than delimiter"
             UART_CPutString("Found valid command\r\nCommand =>");
             UART_PutString(strPtr);              // Print out command
             UART_CPutString("<\r\nParamaters:\r\n");
             while(strPtr = UART_szGetParam()) { // loop on each parameter
                UART_CPutString("   <");
                UART_PutString(strPtr);          // Print each parameter
                UART_CPutString(">\r\n");
             }
          }
       UART_CmdReset();                           // Reset command buffer
       }
    }
}
```

コンフィグレーション レジスタ

ここでは、UART ユーザ モジュールによって使用または変更される、PSoC ブロックとその他のシステム レジスタについて説明します。パラメータ化された記号のみ説明します。

Table 4. ブロック RX、データシフト レジスタ : DR0

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

RX シフト レジスタ : 入力でスタートビットが検出されると、RX 状態マシンのハードウェアが、このレジスタにデータをシフトする 8 分周ビットクロックを生成します。

Table 5. ブロック RX、データレジスタ : DR1

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 6. ブロック RX、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 : ストップビットがサンプリングされた後、RX シフト レジスタからデータが転送されます。

Table 7. ブロック RX、制御レジスタ : CR0

ビット	7	6	5	4	3	2	1	0
値	パリティ エラー	オーバラン エラー	フレーミング エラー	Rx アクティブ	Rx レジスタに空きなし	パリティのタイプ	パリティ イネーブル	Rx イネーブル

パリティ エラーは、受信したデータ バイトのパリティ計算結果を表すフラグです。オーバラン エラーは、受信バッファ レジスタ データが上書きされたことを表すフラグです。フレーミング エラーは、ストップビットが正しく受信されたことを表すフラグです。Rx アクティブは、データ バイトがアクティブに受信されたかどうかを表すフラグです。Rx レジスタに空きなしは、データ バイトが完全に受信され、データ バイトが受信バッファ レジスタに送信され、かつエラー条件が有効であることを表すフラグです。パリティのタイプは、計算するパリティのタイプです。このビットは、パリティ イネーブル ビットが設定されているかどうかには関係しません。パリティ イネーブルは、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、パリティのタイプビットを設定することで選択されます。Rx イネーブルは、RX8 レシーバを有効または無効にします。

Rx 制御レジスタの読み取りは、ステータス データをデータ バスに入力し、すべてのステータス ビットをクリアします。パリティ エラーまたはオーバフロー エラーが検出された場合、ファームウェアはどのような動作も実行する必要がないはずです。フレーミング エラーが検出されると、フレーマが次のデータ バイトを直ちに探し始めます。ロジック 0 に RX 入力が長時間留まる可能性のあるシステムでは、レシーバを再度有効にする前に、レシーバを停止し、ラインがロジック 1 に戻るようポーリングする必要があります。ロジック 0 で RX 入力が止まった場合、ラインはロジック 0 にあり、スタートビットがあるべき場所でロジック 0 が検出されるとフレーミング エラーを引き起こしますが、これによって「誤った」スタートビットの繰り返し検出を回避します。

Table 8. ブロック RX、レジスタ：関数

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、レシーバになるこのデジタル通信タイプ「A」ブロックの特性を定義します。

Table 9. ブロック RX、レジスタ：入力

ビット	7	6	5	4	3	2	1	0
値	RX 入力				クロック源			

RX 入力は、RX 入力ソースを選択します。クロック源は、レシーバのタイミングを駆動するクロックを選択します。

Table 10. ブロック RX、レジスタ：出力

ビット	7	6	5	4	3	2	1	0
値	0	0	0	0	0	0	0	0

Table 11. 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	TX シフト レジスタ							

TX シフト レジスタは、内容をシフトし最下位ビットを送信するよう、TX ステートマシン ハードウェアによって制御されます。データは、TX 状態マシン ハードウェアにより、DR1 データ レジスタからこのレジスタにロードされます。

Table 12. 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	送信バッファ レジスタ							

送信バッファ 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。このレジスタにロードされたデータは、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 13. 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 14. 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 レジスタに空きあり	予約	パリティのタイプ	パリティ イネーブル	TX イネーブル

TX 完了は、データ バイトが送信処理中であるかどう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。TX レジスタに空きありは、バッファ レジスタが空であることを示すフラグです。パリティのタイプは、計算するパリティ タイプです。このビットは、パリティ イネーブル ビットが設定されているかどうかには関係しません。パリティ イネーブルは、データ バイトを用いたパリティ ビットの計算と送信を、有効または無効にします。パリティは、パリティのタイプビットを設定することで選択されます。Tx イネーブルは TX トランスミッタを有効または無効にします。

Table 15. 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、トランスミッタになるこのデジタル通信タイプ 「A」ブロックの特性を定義します。

Table 16. 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	クロック源			

クロック源は、トランスミッタのタイミングを駆動するために選択されたクロックです。

Table 17. 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 出力イネーブルと選択		

TX 出力イネーブルと選択は、TX8 の出力を指定します。

バージョン ヒストリー

バージョン	著者	説明
5.3	DHA	大容量メモリ モデルのチップの互換性が追加されました。

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。