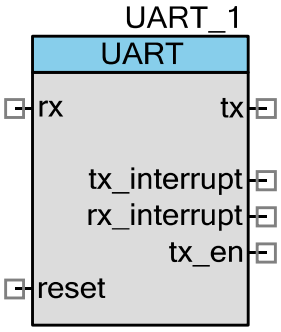
ユニバーサル非同期レシーバ トランスミッタ (UART)

2.10

特長



- ハードウェアアドレス検出付き 9 ビットアドレスモード
- ボーレート 110～921600 bps または任意最大 4 Mbps
- RX および TX バッファ = 4～65535
- フレーミング、パリティ、およびオーバーランエラーの検出
- 全二重、半二重、TX 専用、および RX 専用に最適化されたハードウェア
- ビット毎の 2/3 多数決機能
- ブレーク信号の生成と検出
- 8x または 16x のオーバーサンプリング

概要説明

UART では、一般に RS232 または RS485 と呼ばれている非同期通信が、提供されます。UART コンポーネントは、全二重、半二重、TX 専用、または RX 専用バージョンに設定できます。これらのバージョンはすべて、同じ基本機能を提供します。相違しているのは、使用するリソースの量のみです。

UART がデータを受信および送信する処理の補助として、独立したサイズの設定可能なバッファが提供されています。SRAM 中の独立した受信リングバッファと伝送用バッファ、及びハードウェア FIFO を使用すると、データの欠落がなくなります。これにより CPU は、UART へのサービスではなく重要なリアルタイムのタスクに、より多くの時間を使うことができます。

使用するほとんどの場合、ボーレート、パリティ、データビット数、スタートビット数を選択することにより、UART を簡単に設定できます。多くの場合、RS232 の一般的な設定は、「8N1」として表示されます。これは、8 データビットの省略表記で、パリティなし、1 ストップビットです。これは UART コンポーネントの初期設定です。これにより、ほとんどのアプリケーションで設定する必要があるのは、ボーレートのみです。UART で 2 番目によく使用されるのは、マルチドロップ RS485 ネットワークでの使用です。UART コンポーネントは、ハードウェアアドレス検出付き 9 ビットアドレス指定モード、および送信時の TX トランシーバをイネーブルにする TX 出力イネーブル信号を、サポートします。

UART は長期間にわたって使用されており、その間に物理層およびプロトコル層の多くの種類があります。これには RS423、DMX512、MIDI、LIN バス、従来型端子プロトコル、IrDa があり、さらにこれら以外もあります。通

常使用される種類の UART をサポートするために、コンポーネントによって、データビットの数、ストップビット、パリティ、ハードウェア フロー制御、およびパリティの生成と検出の、設定サポートが提供されます。

ハードウェアによってコンパイルされるオプションとして、クロックおよびシリアルデータストリームを出力するように選択できます。このデータ ストリームは、クロックの立ち上がりエッジで UART データビットのみを出力します。独立したクロックとデータ出力が、TX と RX の両方に提供されます。これらの出力の目的は、CRC コンポーネントを UART に接続することによって、データ CRC の自動計算を可能にすることです。

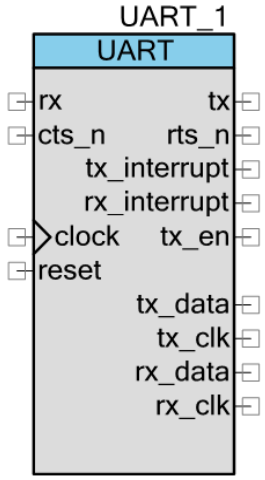
UART を使用する時期

互換性のある非同期通信インターフェースが必要な場合(特に RS232、RS485、その他の種類)はいつでも、URAT を使用します。UART を使用して、DMX512、LIN、IrDa、顧客や業界の資産となっているような、より高度な非同期ベースのプロトコルを作成することができます。

特定のコンポーネントが既にプロトコルをりようするように作成されている場合は、UART を使用しないでください。たとえば DMX512、LIN、または IrDa コンポーネントが提供されている場合、これによって、物理層およびプロトコル層の両方の機能を提供する実装が行われます。この場合、UART は必要ありません(コンポーネントの空き状況によります)。

入出力接続

このセクションでは、UART への種々の入力および出力の接続を、説明します。一部の I/O のシンボルは、その I/O の説明に示されている条件下で非表示になることがあります。



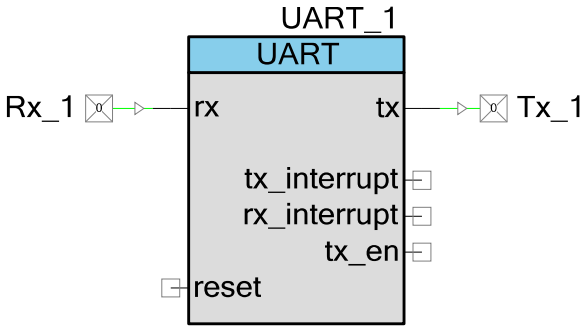
入力	非表示の場合	説明
クロック	Y	クロック入力により、シリアル通信のボーレート(ビットレート)が定義されます。ボーレートは、**Oversampling Rate**パラメータによって異なりますが、入力クロック周波数の1/8または1/16です。この入力は、**Clock Selection**パラメータが**External Clock**に選択されている場合、表示されます。内部クロックが選択されている場合、必要なボーレートを設定時に定義する必要があり、PSoC Creatorによって必要なクロック周波数が解析されます。
リセット	N	リセット入力により、UARTステートマシン(RXおよびTX)がアイドル状態にリセットされます。これにより、現在送信中または受信中のデータはすべて取り消されます。この入力は同期リセットであり、少なくとも1つのクロックの立ち上がりエッジが必要です。リセット入力は、外部接続なしでフローティング状態のままにすることができます。リセット ラインになにも接続されていない場合、コンポーネントによって定数0が割り当てられます。
rx	Y	rx入力には、シリアルバスの他のデバイスからの入力シリアルデータが含まれます。**Mode**パラメータが**RX Only**、**Half Duplex**、または**Full UART (RX + TX)**に設定されている場合、この入力は表示されており、接続する必要があります。
cts_n	Y	cts_n入力は、他のデバイスでデータ受信準備が完了していることを、示します。これはアクティブLOW (_n)入力です。この入力は、**Flow Control** パラメータが **Hardware** に設定されている場合、表示されます。

出力	非表示の場合	説明
tx	Y	tx出力には、シリアルバスの他のデバイスへの出力シリアルデータが含まれます。この出力は、**Mode**パラメータが **TX Only**、**Half Duplex**、または**Full UART (RX + TX)**に設定されている場合、表示されます。システムリセットがアクティブなときの予期しないLowレベルのパルスからレシーバを保護するため、外部にプルアップ抵抗を使用することを、サイプレスは推奨します。
rts_n	Y	rts出力は他のデバイスに、対して、データを受信する準備が完了していることを、伝えます。この出力はアクティブLOW (_n)です。RTS信号は、RXバッファサイズパラメータ(4より大きい場合)によって割り当てられた内部FIFOおよびRXバッファがフルの場合、HIGHになります。この出力は、**Flow Control**パラメータが**Hardware**に設定されている場合、表示されます。
tx_en	Y	tx_en出力は、おもにRS485通信に使用されて、使用しているデバイスがバスで送信中であることを、示します。この出力は、送信が開始する前にHIGHに、送信が完了するとLOWになります。これにより、このバス上の他のデバイスに対してビジー状態のバスが、表示されます。この出力は、**Hardware TX Enable**パラメータが選択されている場合、表示されます。
tx_interrupt	Y	tx_interrupt出力は、可能な割り込みソースのグループの論理ORです。この信号は、任意のイネーブルな割り込みソースがTRUEの場合、HIGHになります。この出力は、**Mode**パラメータが**TX Only**または**Full UART (RX + TX)**に設定されている場合、表示されます。
rx_interrupt	Y	rx_interrupt出力は、可能な割り込みソースのグループの、論理ORです。この信号は、任意のイネーブルな割り込みソースがTRUEの場合、HIGHになります。この出力は、**Mode**パラメータが **RX Only**、**Half Duplex**、または**Full UART (RX + TX)**に設定されている場合、表示されます。

tx_data	Y	tx_data出力は、TXデータをCRCコンポーネントまたは他のロジックへシフトアウトするために使用します。この出力は、**Enable CRC outputs**パラメータが選択されているときに、表示されます。
tx_clk	Y	tx_clk出力により、TXデータをCRCコンポーネントまたは他のロジックにシフトアウトするためのクロックエッジが提供されます。この出力は、**Enable CRC outputs**パラメータが選択されているときに、表示されます。
rx_data	Y	rx_data出力は、RXデータをCRCコンポーネントまたは他のロジックにシフトアウトするために使用します。この出力は、**Enable CRC outputs**パラメータが選択されているときに、表示されます。
rx_clk	Y	rx_clk出力により、RXデータをCRCコンポーネントまたは他のロジックにシフトアウトするために使用するクロック エッジが、提供されます。この出力は、**Enable CRC outputs**パラメータが選択されているときに、表示されます。

回路図マクロ情報

コンポーネント カタログのデフォルトの UART は、デフォルトの設定で UART コンポーネントを使用する回路図マクロです。デジタル入力および出力のピン接続部に接続されます。



コンポーネント パラメータ

UART コンポーネントを、使用するデザインまでドラッグしてダブルクリックし、**Configure** ダイアログを開きます。

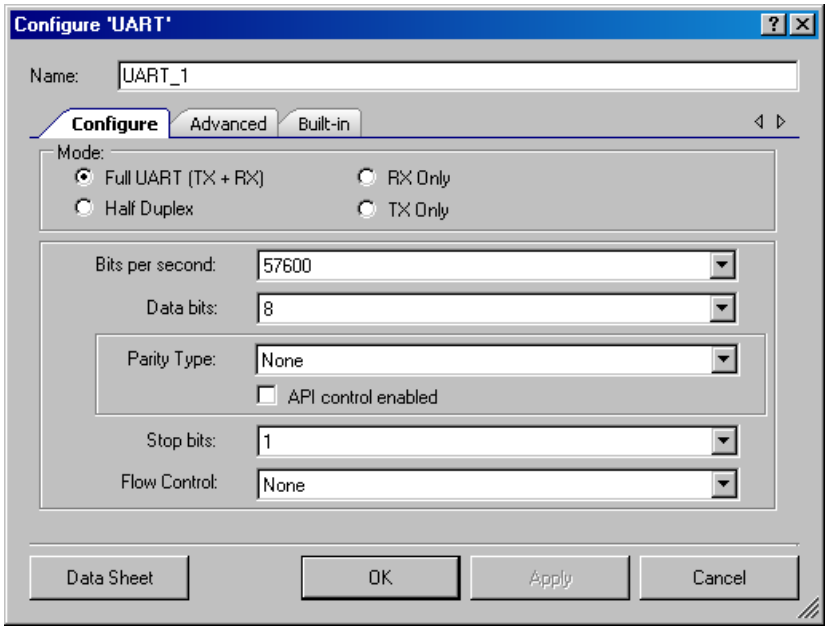
ハードウェア対ソフトウェア オプション

ハードウェアコンフィギュレーションオプションは、ハードウェア中でコンポーネントが合成・設置される方法を変えます。これらのオプションを変更した場合、ハードウェアをリビルドしなければなりません。ソフトウェアコンフィギュレーション オプションは、合成や配置に影響を及ぼしません。これらのパラメータのビルド前の設定時、いつでも API の提供によって変更可能な値の初期値を設定します。

以下のセクションでは、UART パラメータについて、およびダイアログを使用してこれを設定する方法を、説明します。オプションがハードウェアであるかソフトウェアであるかについても、示されています。

Configure (設定) タブ

このダイアログは、シリアルバスの両端の設定を間違わないように、ハイパーターミナルのコンフィギュレーションウィンドウと同じようになっています。この理由は、ハイパーターミナルを使用した PC がきわめて多くの場合、バスのもう一方の側だからです。



これらのオプションのすべては、ハードウェア構成のオプションです。

Mode (モード)

このパラメータにより、UART に含める機能コンポーネントが定義されます。これにより、双方向の **Full UART (TX + RX)** (デフォルト)、**Half Duplex** UART(リソースの半分を使用)、RS232 レシーバ (**RX Only**)または トランスミッタ(**TX Only**)として、セットアップできます。

ビット/秒

このパラメータにより、クロック生成用ハードウェアのボーレートまたはビット幅の設定が、定義されます。デフォルトは **57600** です。

内部クロック(**Clock Selection** パラメータによって設定)を使用する場合、このボレートを達成するために必要なクロックが、PSoC® Creator によって生成されます。

データビット

このパラメータにより、単一の UART トランザクションの開始と停止の間に送信されるデータビットの数が、定義されます。オプションは、**5**、**6**、**7**、**8** (初期設定)、または **9** です。

- 8 データビットが初期設定であり、1 転送あたり 1 バイトを送信します。
- 9 ビットモードは 9 データビットを送信しません。9 番目のビットは、マーク/スペース パリティを使用するアドレスの指示子として、パリティビットの代わりをします。9 データビットモードを使用している場合、マーク/スペース パリティを選択する必要があります。

パリティのタイプ

このパラメータにより、転送でのパリティビットの位置の機能が、定義されます。これは、 **None** (初期設定)、**Odd**、**Even**、または **Mark/Space** に設定できます。9 データビットを選択する場合、**Parity Type** として **Mark/Space** を選択します。

API 制御が有効

このチェックボックスは、コントロール レジスタおよび UART_WriteControlRegister()関数を使用してパリティを変更するために、使用します。このオプションを選択した場合、このパリティタイプを、UART の動作を邪魔することなく、バイト間で動的に変更できますが、より多くのリソースがコンポーネントによって使用されます。

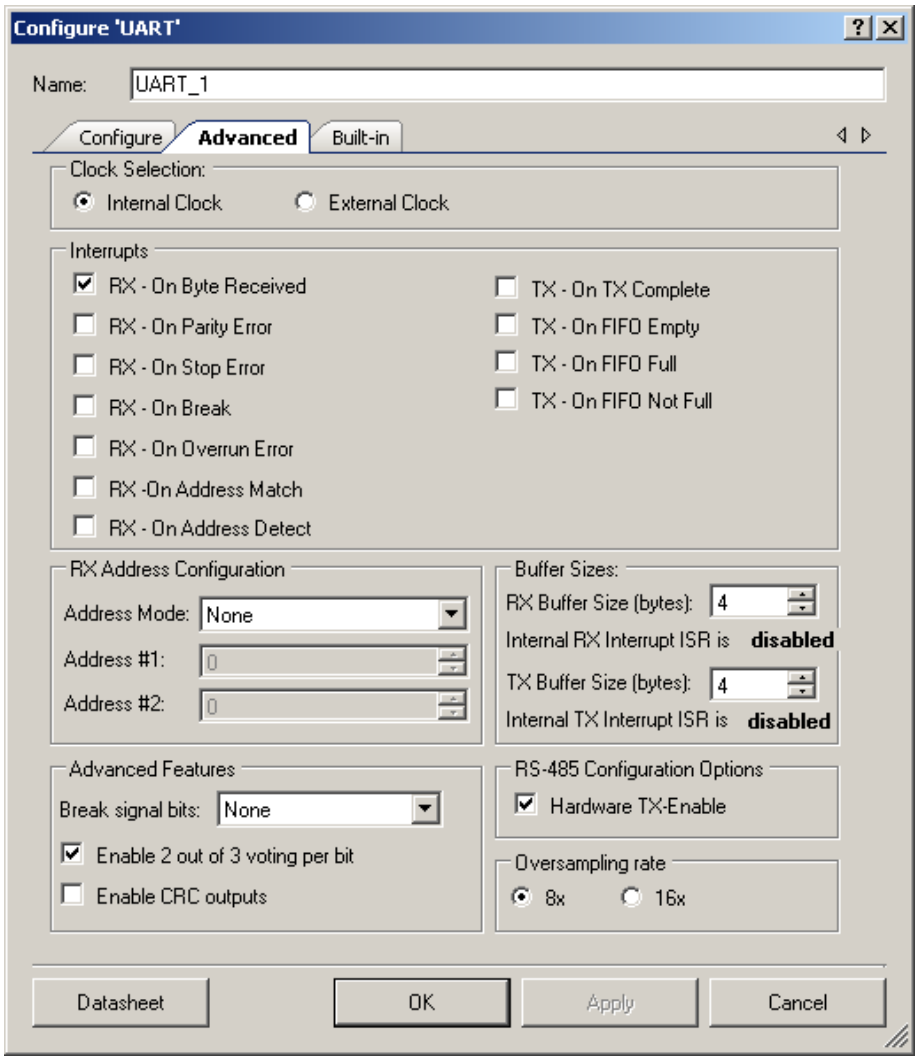
ストップビット

このパラメータにより、トランスミッタに実装されるストップビットの数が、定義されます。このパラメータは、**1** (初期設定)または **2** データビットに設定できます。

フロー制御

このパラメータにより、**Hardware** または **None** (初期設定)を選択できます。このパラメータを **Hardware** に設定すると、CTS および RTS 信号がシンボル上に現れます。

Advanced Tab（詳細設定タブ）



ハードウェア設定オプション

Clock Select (クロック選択)

このパラメータにより、ボーレートの生成に、内部設定クロック、外部設定クロック、または I/O を選択できます。**Internal Clock** に設定すると、必要なクロック周波数は PSoC Creator によって計算および設定されます。**External Clock** モードでは、ボーレートはコンポーネントによってコントロールすることはできませんが、、ユーザは予想されるボーレートを計算することができます。

このパラメータを **Internal Clock** に設定すると、クロック入力のシンボルは表示されません。

Address モード

このパラメータにより、ハードウェアとソフトウェアが対話して、デバイスのアドレスおよびデータバイトをハンドルする方法が、定義されます。このパラメータは、以下のタイプに設定できます。

- **Software Byte by Byte** – ハードウェアにより、受信したすべてのバイトに対して、検出したアドレスバイト(UART_RX_STS_MRKSPC ステータス)が示されます。ソフトウェアは、このバイトを読み取って、このアドレスが **Address #1** または **Address #2** パラメータに定義されたデバイス アドレス、または他の追加アドレスに一致するか、判断する必要があります。
- **Software Detect to Buffer** – ハードウェアが、アドレスバイト(UART_RX_STS_MRKSPC ステータス)を検出したことを示します。RX ISR に組み込まれたソフトウェアがバイトを読み取り、このアドレスが **Address #1** または **Address #2** パラメータ(UART_RX_STS_ADDR_MATCH ステータスを使用)に定義されたデバイス アドレスに一致するか判断します。次に、すべてのアドレス指定されたデータが、アドレスバイトとともに、**RX Buffer Size** パラメータで定義された RX バッファにコピーされます。**RX Buffer Size** は、手動で 4 より大きい値に設定する必要があります。アドレス指定されていないデータは、FIFO から読み取られますが、バッファには書き込まれません。
- **Hardware Byte By Byte** – ハードウェアにより、アドレス指定されたバイトが検出され、すべてのデータをアドレスとともに、ハードウェア FIFO から **RX Buffer Size** によって定義されたデータ バッファに移動するため割り込み(RX-オン バイト受信)が要求されます。ハードウェアは、アドレス指定されていないバイトは FIFO に保存せず、これに対する割り込みも生成しません。
- **Hardware Detect to Buffer** – ハードウェアにより、アドレス指定されたバイトが検出され、データ(アドレスバイトは含まれません)のみをハードウェア FIFO から **RX Buffer Size** によって定義されたデータ バッファに移動するため、割り込み(RX-オン バイト受信)が要求されます。ハードウェアは、アドレス指定されていないバイトは FIFO に保存せず、これに対する割り込みも生成しません。
- **None** – RX アドレス検出は組み込まれていません。

RX アドレス#1/#2

RX Address パラメータは、UART が所有することがあるデバイス アドレスを、最大 2 つまで示します。これらのパラメータは、**Address Mode** パラメータに説明されているハードウェアアドレス検出モード用のハードウェアに保存されます。**RX Address #2** 用のハードウェアは、**Half Duplex** モードでは組み込まれません。このパラメータは、ソフトウェア アドレスモード用ファームウェアに使用できます。

高度機能

- **Break signal bits** – Break signal bits パラメータにより、ブレーク信号の生成と検出がイネーブルになり、送信するロジック 0 のビット数が定義されます。このオプションを **None** に設定すると、リソースが節約されます。

- **Enable 2 out of 3 voting per bit** – **Enable 2 out of 3 voting per bit** パラメータにより、エラー補正アルゴリズムがイネーブルまたはディスエーブルになります。このオプションをデイスエーブルにすると、リソースが節約されます。詳細については、このデータシートの「 機能説明 」セクションを参照してください。
- **Enable CRC outputs** – **Enable CRC outputs** パラメータにより、tx_data, tx_clk、rx_data および rx_clk 出力がイネーブルまたはディスエーブルになります。これらは、クロックと、クロックの立ち上がりエッジで UART データビットのみを出力するシリアルデータストリームを出力するために使用します。これらの出力の目的は、データ CRC の自動計算を可能にすることです。このオプションをデイスエーブルにすると、リソースが節約されます。

Hardware TX Enable

このパラメータにより、TX UART の TX-イネーブル出力の使用が、イネーブルまたはディスエーブルになります。この信号は、RS485 通信で使用されます。このハードウェアにより、この出力の機能が、バッファの状態に基づいて、自動的に提供されます。

Oversampling Rate

このパラメータにより、ボーレート生成用クロック分周器を選択できます。

ソフトウェアの設定オプション

割り込み

Interrupt On パラメータにより、割り込みソースを設定できます。これらの値は他の **Interrupt On** パラメータのすべてと OR されており、割り込みをトリガできるイベントの最終グループを与えます。ソフトウェアはこれらのモードを、いつでも変更できます。これらのパラメータは、初期設定を定義します。

- **RX - On Byte Received** (UART_RX_STS_FIFO_NOTEMPTY)
- **RX - On Parity Error** (UART_RX_STS_PAR_ERROR)
- **RX - On Stop Error** (UART_RX_STS_STOP_ERROR)
- **RX - On Break** (UART_RX_STS_BREAK)
- **RX - On Overrun Error** (UART_RX_STS_OVERRUN)
- **RX - On Address Match** (UART_RX_STS_ADDR_MATCH)
- **RX - On Address Detect** (UART_RX_STS_MRKSPC)
- **TX - On TX Complete** (UART_TX_STS_COMPLETE)
- **TX - On FIFO Empty** (UART_TX_STS_FIFO_EMPTY)
- **TX - On FIFO Full** (UART_TX_STS_FIFO_FULL)
- **TX - On FIFO Not Full** (UART_TX_STS_FIFO_NOT_FULL)

ISR を、tx_interrupt または rx_interrupt 出力に接続された外部割り込みコンポーネントを使用して、ハンドルできます。割り込み出力ピンは、選択した **Mode** パラメータに応じて表示されます。同じ信号が、選択したステータス割り込みに基づく内部割り込みに出力されます。

これらの出力は、割り込みとは無関係に RX または TX バッファから DMA への DMA 要求ソースとして、または希望する機能に対応した他の割り込みとして、使用できます。

RX Buffer Size (bytes)

このパラメータにより、RX バッファに割り当てる RAM のバイト数が定義されます。データが受信レジスタからこのバッファへ移動します。

選択したバッファ サイズが 4 バイトのとき、ハードウェア FIFO の 4 バイトが、バッファとして使用されます。4 バイトより大きいバッファ サイズには、データの受信 FIFO からこのバッファへの移動をハンドルするために、割り込みを使用する必要があります。UART_GetChar()または UART_ReadRXData()関数により、使用しているトップレベルのファームウェアを変更することなく、正しいソースからデータが取得されます。

RX バッファのサイズが 4 バイトより大きい場合、**Internal RX Interrupt ISR** が自動的にイネーブルになり、**RX – On Byte Received** 割り込みソースは、正しくないハンドラ機能の原因となるため、選択されて使用できなくなります。

TX Buffer Size (bytes)

このパラメータにより、TX バッファに RAM の何バイトを割り当てるかが、定義されます。データがこのバッファに、UART_PutChar() および UART_PutArray() API コマンドを使用して書き込まれます。

選択したバッファサイズが 4 バイトの場合、ハードウェア FIFO の 4 バイトが、バッファとして使用されます。これ以外の場合、RAM バッファが割り当てられます。4 バイトより大きいバッファ サイズには、使用しているトップレベルのファームウェアを変更することなく、データの送信バッファからハードウェア FIFO への移動をハンドルするために、割り込みを使用する必要があります。

TX バッファが 4 バイトより大きい場合、**Internal TX Interrupt ISR** が自動的にイネーブルになり、**TX – On FIFO EMPTY** 割り込みソースは、正しくないハンドラ機能の原因になるため、選択されて使用できなくなります。

TX 割り込みは、**Half Duplex** モードでは使用できません。このため、**Half Duplex** モードを選択したとき、**TX Buffer Size** は 4 バイトに制限されます。

Internal RX Interrupt ISR

UART の RX 部分のコンポーネントによって提供される ISR をイネーブルにします。このパラメータは、**RX Buffer Size** パラメータに対応して、自動的に設定されます。その理由は、データの FIFO から RX バッファへの転送のハンドルに、内部 ISR が必要だからです。

Internal TX Interrupt ISR

UART の TX 部分のコンポーネントによって提供される ISR をイネーブルにします。このパラメータは、**TX Buffer Size** パラメータに対応して自動的に設定されます。その理由は、データの TX バッファから FIFO への転送のハンドルに、内部 ISR が必要だからです。

Clock Select (クロック選択)

内部クロック構成が選択されている場合、PSoC Creator により、必要な周波数およびクロック ソースが計算され、実装に必要なリソースが生成されます。そうでない場合、ユーザーがクロックを提供して、入力クロック周波数の 1/8 または 1/16 でボーレートを計算する必要があります。

クロックの許容誤差は、最大±2 パーセントでなくてはなりません。クロックをこの制限内で生成できない場合、警告が生成されます。その場合、DWR のマスタ クロックを変更するか、外部の水晶を元にしたクロックを使用する必要があります。

配置

UART コンポーネントは UDB アレイ全体に配置され、配置情報はすべて *cyfitter.h* ファイルを使用して API に提供されます。

リソース

リソース	リソースのタイプ					API メモリ(バイト)		ピン (外部入出力ごと)
	データパス セル	PLD	ステータス セル	Control/ Count7 セル	割り込み	フラッシュ	RAM	
フルUART	3	27	2	2	2	1695	24	12
フルUART*	2	28	2	3	2	1703	24	12
単純なUART	3	6	2	1	0	551	5	4
半二重	1	7	1	2	0	577	4	3
RX専用	1	3	1	1	0	223	3	2
TX専用	2	3	1	0	0	383	4	2
TX専用*	1	3	1	1	0	431	4	2

*パラメータTxBitClkGenDP = false (切り替えるには、AdvancedタブのExpression Viewに移動します)。

アプリケーション・プログラミング・インターフェース

アプリケーション・プログラミング・インターフェース (API) ルーチンにより、ソフトウェアを使用してコンポーネントを設定できます。次の表は、各関数へのインターフェースとその説明を示しています。後続のセクションでは、各機能を詳細に説明します。

初期設定では、PSoC Creator により、所定の設計でのコンポーネントの最初のインスタンスに、インスタンス名「UART_1」が、割り当てられます。インスタンス名は、識別子の構文ルールに従った独自の名称に変更できます。インスタンス名は、すべてのグローバル関数名、変数名、定数名のプリフィックスになります。理解しやすいように、以下の表で使用されるインスタンス名を「UART」とします。

機能	説明
UART_Start()	UART の動作を初期化し、イネーブルにします
UART_Stop()	UART の動作をディスエーブルにします
UART_ReadControlRegister()	コントロールレジスタの現在の値を返します
UART_WriteControlRegister()	8ビット値をコントロールレジスタに書き込みます
UART_EnableRxInt()	内部割り込みIRQをイネーブルにします
UART_DisableRxInt()	内部割り込みIRQをディスエーブルにします
UART_SetRxInterruptMode()	有効なRX割り込みソースを設定します
UART_ReadRxData()	RXデータレジスタのデータを返します
UART_ReadRxStatus()	ステータスレジスタの現在の状態を返します
UART_GetChar()	受信データの次のバイトを返します
UART_GetByte()	UART RXバッファを直ちに読み取り、受信した文字とエラーコンディションを返します
UART_GetRxBufferSize()	RXバッファに残っている受信したバイトの数を返し、カウント数をバイト単位で返します
UART_ClearRxBuffer()	受信データすべてのメモリ アレイをクリアします
UART_SetRxAddressMode()	UARTのRX部分で使用されるソフトウェアによってコントロールするアドレス指定モードを、設定します
UART_SetRxAddress1()	2 つのハードウェア検出可能アドレスのうち1番目のものを設定します
UART_SetRxAddress2()	ハードウェアによる検出が可能な2つのアドレスのうち2番目のものを設定します
UART_EnableTxInt()	内部割り込みIRQをイネーブルにします
UART_DisableTxInt()	内部割り込みIRQをディスエーブルにします
UART_SetTxInterruptMode()	有効なTX割り込みソースを設定します
UART_WriteTxData()	バッファの余裕およびステータスを確認せずに、1バイトを送信します
UART_ReadTxStatus()	UARTのTX部分のステータスレジスタを読み取ります

機能	説明
UART_PutChar()	1バイトのデータを、バスを使用できるときに送信する送信バッファに入れます
UART_PutString()	文字列データ送信用メモリ バッファに入れます
UART_PutArray()	配列データを送信用メモリ バッファに入れます
UART_PutCRLF()	1バイトのデータの後にキャリッジ リターンとライン フィードをつけたものを送信バッファに書き込みます
UART_GetTxBufferSize()	TXバッファで使用するバイト数を決定します。バッファが空のときは、0を返します
UART_ClearTxBuffer()	TXバッファからすべてのデータをクリアします
UART_SendBreak()	バスのブレーク信号を送信します
UART_SetTxAddressMode ()	次のバイトがアドレスなのか、データなのかを示すよう、トランスミッタを設定します
UART_LoadRxConfig()	レシーバ設定をロードします。Half Duplex UARTのバイト受信準備が完了します
UART_LoadTxConfig()	トランスミッタ設定をロードします。Half Duplex UARTのバイト送信準備が完了します
UART_Sleep()	UARTの動作を停止し、ユーザー設定を保存します
UART_Wakeup()	ユーザ設定を復元し、有効にします
UART_Init()	カスタマイザにで提供された初期設定に初期化します
UART_Enable()	UARTブロック動作をイネーブルにします
UART_SaveConfig()	現在のユーザ設定を保存します
UART_RestoreConfig()	ユーザ設定を復元します

グローバル変数

変数	説明
UART_initVar	UARTが初期化済みか否かが示されます。変数は初期化で0になり、UART_Start()が初めて呼び出されたときに、1に設定されます。これによりコンポーネントを、UART_Start()ルーチンへの最初の呼び出し後に、再初期化することなく再起動できます。 コンポーネントを正しい動作のためには、SendまたはPutコマンドを実行する前に、UARTを初期化する必要があります。このため、送信データを書き込むすべてのAPIで、コンポーネントがこの変数を使用して初期化されていることを、確認する必要があります。 コンポーネントの再初期化が必要な場合、UART_Start()またはUART_Enable()関数の前に、UART_Init()関数を呼び出します。
UART_rxBuffer	これは、ユーザが定義する長さの、RAMに割り当てられたRXバッファです。このバッファは、**RX Buffer Size**パラメータが4より大きく設定されているときに受信データを保管するために、割り込みによって使用されます。UART_ReadRxData()およびUART_GetChar()によって、データをユーザ レベルのファームウェアに転送するためにも、使用されます。

変数	説明
UART_rxBufferWrite	この変数は、UART_rxBufferにデータを書き込むための巡回インデックスとして、RX割り込みによって使用されます。この変数は、UART_ReadRxData()およびUART_GetChar()関数によって、新しいデータを特定するためにも使用されます。UART_ClearRxBuffer()関数によってゼロにクリアされます。
UART_rxBufferRead	この変数は、UART_rxBufferからデータを読み取るための巡回インデックスとして、UART_ReadRxData()およびUART_GetChar()関数で使用されます。UART_ClearRxBuffer()関数によってゼロにクリアされます。
UART_rxBufferLoopDetect	UART_rxBufferWriteインデックスがUART_rxBufferReadインデックスに追いついたとき、この変数はRX割り込みで1に設定されます。これは次のバイトを受信したとき、UART_rxBufferOverflowになる、過負荷直前の状態です。UART_ReadRxData()またはUART_GetChar()関数が呼び出されたとき、ゼロに設定されます。UART_ClearRxBuffer()関数によってゼロにクリアされます。
UART_rxBufferOverflow	この変数は、過負荷状態を示すために使用されます。UART_rxBufferに新しいデータを書き込む空きスペースがない場合、RX割り込みで1に設定されます。この状態は、UART_ReadRxStatus()関数によってUART_RX_STS_SOFT_BUFF_OVERビットとして、RXステータスレジスタ ビットとともに返され、ゼロにクリアされます。UART_ClearRxBuffer()関数によってゼロにクリアされます。
UART_txBuffer	これは、ユーザが定義する長さの、RAM-割り当てTXバッファです。このバッファは、**TX Buffer Size**パラメータが4より大きい値に設定されたとき、APIを送信することによって、送信用データを保管するために使用されます。TX割り込みにより、データをハードウェアFIFOに移動するためにも、使用されます。
UART_txBufferWrite	この変数は、UART_WriteTxData()、UART_PutChar()、UART_PutString()、UART_PutArray()、およびUART_PutCRLF()関数によって、UART_txBufferにデータを書き込むための巡回インデックスとして使用されます。この変数は、TX割り込みによって、送信用の新しいデータを識別するためにも使用されます。UART_ClearTxBuffer()関数によって、ゼロにクリアされます。
UART_txBufferRead	この変数はUART_txBufferからデータを読み取るための巡回インデックスとしてTX割り込みによって、使用されます。UART_ClearRxBuffer()関数によってゼロにクリアされます。

void UART_Start(void)

説明: これは、コンポーネントの動作を開始する際に推奨される方法です。UART_Start()により、initVar変数が設定され、UART_Init()関数が呼び出され、その後UART_Enable()関数が呼び出されます。

パラメータ: 無効

返り値: 無効

副作用: initVar変数がすでに設定されている場合、この機能はUART_Enable()関数のみを呼び出します。

void UART_Stop(void)

説明: UART動作がディスエーブルになります。

パラメータ: 無効

返り値: 無効

副作用: なし

uint8 UART_ReadControlRegister(void)

説明: コントロールレジスタの現在の値を返します。

パラメータ: 無効

返り値: uint8: コントロールレジスタの内容　以下の定義は、返された値を解釈するために、使用できます。詳細については、このデータシートの終わり近くにある制御レジスタの説明を、参照してください。

値	説明
UART_CTRL_HD_SEND	半二重 UART (イネーブルの場合)がRXモード(0)かTX モード(1)かを設定します。
UART_CTRL_HD_SEND_BREAK	バスのブレーク信号を送信するように、設定します。このビットはUART_SendBreak()関数によって、書き込まれます。
UART_CTRL_MARK	次のトランザクションの間のパリティビット(Mark/Spaceパリティモード時)が1か0かを、設定します。
UART_CTRL_PARITY_TYPE_MASK	ソフトウェア設定が可能な場合、次の転送のパリティを設定する、2ビット幅のフィールド。以下の定義は、UART_CTRL_PARITY_TYPE0_SHIFTによって左にシフトして、パリティのタイプを特定するために、使用できます:
UART__B_UART__NONE_REVB	パリティなし
UART__B_UART__EVEN_REVB	偶数パリティ
UART__B_UART__ODD_REVB	奇数パリティ
UART__B_UART__MARK_SPACE_REVB	マーク/スペース パリティ

UART_CTRL_RXADDR_MODE_MASK	UARTレシーバに要求されるハードウェアアドレス指定動作を設定する、3ビット幅のフィールド。以下の定義は、UART_CTRL_RXADDR_MODE0_SHIFTによって左にシフトして、アドレスモードを特定するために、使用できます:
UART__B_UART__AM_SW_BYTE_BYTE	Software Byte-by-Byteアドレス検出
UART__B_UART__AM_SW_DETECT_TO_BUFFER	Software Detect to Bufferアドレス検出
UART__B_UART__AM_HW_BYTE_BY_BYTE	Hardware Byte-by-Byteアドレス検出
UART__B_UART__AM_HW_DETECT_TO_BUFFER	Hardware Detect to Bufferアドレス検出
UART__B_UART__AM_NONE	アドレス検出なし

副作用: なし

void UART_WriteControlRegister(uint8 control)

説明: 8ビット値をコントロールレジスタに書き込みます

パラメータ: uint8 control: コントロールレジスタの値

値	説明
UART_ CTRL_HD_SEND	半二重UART(有効な場合)がRXモード(0)かTXモード(1)かを設定します。UART_LoadTxConfig()およびUART_LoadRxConfig()関数を使用して、設定およびクリアできます。
UART_ CTRL_HD_SEND_BREAK	バスのブレーク信号を送信するように、設定します。このビットは、UART_SendBreak()関数を使用して書き込まれるのが最適です。
UART_CTRL_MARK	次のトランザクションの間のパリティビット(Mark/Spaceパリティモード時)が1か0かを、設定します。
UART_CTRL_PARITY_TYPE_MASK	ソフトウェア設定可能な場合に次の転送のパリティを設定する、2ビット幅のフィールド。以下の定義は、UART_CTRL_PARITY_TYPE0_SHIFTによって左にシフトして、パリティのタイプを設定するために、使用できます:
UART__B_UART__NONE_REVB	パリティなし

UART__B_UART__EVEN_REVB	偶数パリティ
UART__B_UART__ODD_REVB	奇数パリティ
UART__B_UART__MARK_SPACE_REVB	マーク/スペース パリティ
UART_CTRL_RXADDR_MODE_MASK	UARTレシーバに対して要求されるハードウェアアドレス指定動作を設定する、3ビット幅のフィールド。以下の定義は、UART_CTRL_RXADDR_MODE0_SHIFTによって左にシフトして、アドレスモードを設定するために、使用できます:
UART__B_UART__AM_SW_BYTE_BYTE	Software Byte-by-Byteアドレス検出
UART__B_UART__AM_SW_DETECT_TO_BUFFER	Software Detect to Bufferアドレス検出
UART__B_UART__AM_HW_BYTE_BY_BYTE	Hardware Byte-by-Byteアドレス検出
UART__B_UART__AM_HW_DETECT_TO_BUFFER	Hardware Detect to Bufferアドレス検出
UART__B_UART__AM_NONE	アドレス検出なし

返り値: 無効

副作用: なし

void UART_EnableRxInt(void)

説明: 内部レシーバの割り込みを有効にします。

パラメータ: 無効

返り値: 無効

副作用: RX内部割り込みの実装がUARTで選択されている場合のみ、使用可能

void UART_DisableRxInt(void)

説明: 内部レシーバの割り込みを無効にします。

パラメータ: 無効

返り値: 無効

副作用: RX内部割り込みの実装がUARTで選択されている場合のみ、使用可能

void UART_SetRxInterruptMode(uint8 intSrc)

説明: 有効なRX割り込みソースを設定します。

パラメータ: uint8 intSrc: イネーブルにするRX割り込みを含むビット フィールド。ステータスレジスタのビットフィールド構成に基づきます。この値は、以下に示すステータスレジスタのビットマスクの組み合わせである必要があります。

値	説明
UART_RX_STS_FIFO_NOTEMPTY	バイト受信割込み。
UART_RX_STS_PAR_ERROR	パリティエラー割り込み。
UART_RX_STS_STOP_ERROR	ストップエラー割り込み。
UART_RX_STS_BREAK	ブレーク割込み。
UART_RX_STS_OVERRUN	オーバーランエラー割り込み。
UART_RX_STS_ADDR_MATCH	アドレス一致割り込み。
UART_RX_STS_MRKSPC	アドレス検出割り込み。

返り値: 無効

副作用: なし

uint8 UART_ReadRxData(void)

説明: 受信データの次のバイトを返します。この関数は、ステータスをチェックせずにデータを返します。ステータスは、個別に確認する必要があります。

パラメータ: 無効

返り値: uint8: RXレジスタからの受信データ

副作用: なし

uint8 UART_ReadRxStatus(void)

説明: レシーバステータスレジスタの現在の状態、およびソフトウェアバッファのオーバーフローステータスを返します。

パラメータ: 無効

返り値: uint8: 現在の RXステータスレジスタ値

値	説明
UART_RX_STS_FIFO_NOTEMPTY	セットされている場合、FIFOに使用可能なデータがあることを示します。

UART_RX_STS_PAR_ERROR	セットされている時、パリティエラーが検出されたことを示します。
UART_RX_STS_STOP_ERROR	セットされている場合、フレーミングエラーが検出されたことを示します。UARTハードウェアが、ストップビットがあるべき場所(ロジック1)にロジック0を見つけると、フレーミングエラーが発生します。
UART_RX_STS_BREAK	セットされている場合、ブレークが検出されたことを示します。
UART_RX_STS_OVERRUN	セットされている場合、FIFOバッファにオーバーランが発生したことを示します。
UART_RX_STS_ADDR_MATCH	受信したバイトが、ハードウェアのアドレス検出に使用できる2つのアドレスの1つに一致することを示します。**Address Mode** が **None** に設定されている場合、実装されません。**Half Duplex**モードでは、この検出用に**Address #1**のみが実装されます。
UART_RX_STS_MRKSPC	マーク/スペースパリティビットのステータス。このビットは、マークまたはスペースのどちらが、転送のパリティビット位置に見出されたかを示します。**Address Mode** が **None** に設定されている場合、実装されません。
UART_RX_STS_SOFT_BUFF_OVER	セットされている時、RXバッファにオーバーランが発生したことを示します。

副作用: すべてのステータスレジスタ ビットは、UART_RX_STS_FIFO_NOTEMPTYを除いて、クリア・オン・リードです。

UART_RX_STS_FIFO_NOTEMPTYは、RXデータレジスタの読み取り後すぐに、クリアされます。

このデータシートの「レジスタ」セクションを、後でご覧ください。

uint8 UART_GetChar(void)

説明: データの最後に受信したバイトを返します。UART_GetChar()は ASCII文字用に設計されており、uint8を返します。ここで、1～255は有効な文字の値であり、0はエラーの発生またはデータが存在しないことを示します。

パラメータ: 無効

返り値: uint8: UART RXバッファから読み取った文字。ASCII文字の値1～255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。

副作用: なし

uint16 UART_GetByte(void)

説明: 直ちにUART RXバッファを読み取り、受信した文字およびエラーの状態を返します。

パラメータ: 無効

返り値: uint16: 上位バイトにはステータスが、下位バイトには UART RX データが含まれます。MSBがゼロ以外の場合、エラーが発生しています。

副作用: なし

uint8/uint16 UART_GetRxBufferSize(void)

説明: RXバッファに残っている受信バイトの数を返します。

パラメータ: 無効

返り値: uint8/uint16: RXバッファに残っているバイトの数の、整数カウント数。タイプは、RX Buffer Sizeパラメータによって異なります。

副作用: なし

void UART_ClearRxBuffer(void)

説明: レシーバ メモリ バッファおよびハードウェアRX FIFOから、受信したデータをすべてクリアします。

パラメータ: 無効

返り値: 無効

副作用: なし

void UART_SetRxAddressMode(uint8 addressMode)

説明: UARTのRX部分で使用される、ソフトウェアでコントロールされるアドレス指定モードを、設定します。

パラメータ: uint8 addressMode: 実装するRXアドレス指定のモードを示す、列挙型の値

値	説明
UART__B_UART__AM_SW_BYTE_BYTE	Software Byte-by-Byteアドレス検出
UART__B_UART__AM_SW_DETECT_TO_BUFFER	Software Detect to Bufferアドレス検出
UART__B_UART__AM_HW_BYTE_BY_BYTE	Hardware Byte-by-Byteアドレス検出
UART__B_UART__AM_HW_DETECT_TO_BUFFER	Hardware Detect to Bufferアドレス検出

UART__B_UART__AM_NONE	アドレス検出なし

返り値: 無効

副作用: なし

void UART_SetRxAddress1(uint8 address)

説明: ハードウェアで検出可能な2つのレシーバアドレスのうち最初のものを、設定します。

パラメータ: uint8アドレス: ハードウェアアドレス検出のアドレス#1

返り値: 無効

副作用: なし

void UART_SetRxAddress2(uint8 address)

説明: ハードウェアで検出可能な2つのレシーバアドレスのうち2番目を設定します。

パラメータ: uint8アドレス: ハードウェアアドレス検出のアドレス#2

返り値: 無効

副作用: なし

void UART_EnableTxInt(void)

説明: 内部トランスミッタ割り込みをイネーブルにします。

パラメータ: 無効

返り値: 無効

副作用: UART構成でTX内部割り込みの実装が選択されている場合のみ、使用できます。

void UART_DisableTxInt(void)

説明: 内部トランスミッタ割り込みをディスエーブルにします。

パラメータ: 無効

返り値: 無効

副作用: UART構成でTX内部割り込みの実装が選択されている場合のみ、使用できます。

void UART_SetTxInterruptMode(uint8 intSrc)

説明: イネーブルにするTX割り込みソース(割り込みはイネーブルにされません)を設定します。

パラメータ: uint8 intSrc: イネーブルにするTX割り込みソースを含むビット フィールド

値	説明
UART_TX_STS_COMPLETE	TXバイト完了時に割り込み
UART_TX_STS_FIFO_EMPTY	ITX FIFOが空のときに割り込み
UART_TX_STS_FIFO_FULL	TX FIFOがフルのときに割り込み
UART_TX_STS_FIFO_NOT_FULL	TX FIFOがフルではないときに割り込み

返り値: 無効

副作用: なし

void UART_WriteTxData(uint8 txDataByte)

説明: バスが使用できるときに、TXステータスレジスタを確認せずに、1バイトのデータを送信対象の送信バッファに入れます。ステータスは別途確認する必要があります。

パラメータ: uint8 txDataByte:データのバイト

返り値: 無効

副作用: なし

uint8 UART_ReadTxStatus(void)

説明: UARTのTX部分のステータスレジスタを読み取ります。

パラメータ: 無効

返り値: uint8: TXステータスレジスタの内容

値	説明
UART_TX_STS_COMPLETE	セットされている時、バイトが正常に送信されたことを、示します
UART_TX_STS_FIFO_EMPTY	セットされている時、TX FIFOが空であることを示します
UART_TX_STS_FIFO_FULL	セットされている時、TX FIFOがフルであることを示します
UART_TX_STS_FIFO_NOT_FULL	セットされている時、FIFOがフルではないことを示します

副作用: この関数は、TXのステータスレジスタを読み取ります。ステータスレジスタは読み取りによってクリアされます。

void UART_PutChar(uint8 txDataByte)

説明: バスが使用できるときに、1バイトのデータを送信対象の送信バッファに入れます。これはブロックAPIであり、TXバッファにデータを保持する余裕ができるまで待ちます。

パラメータ: uint8 txDataByte: 送信するデータを含むバイト

返り値: 無効

副作用: なし

void UART_PutString(char* string)

説明: NULL終端文字列を、送信用バッファに送信します。

パラメータ: char* 文字列: RAMまたはROMに存在するNULL終端文字列アレイのポインタ

返り値: 無効

副作用: TXバッファに文字列全体に対する十分なメモリがない場合、この機能は、文字列の最後の文字がTXバッファにロードされるまでブロックします。

void UART_PutArray(uint8* string, uint8/uint16 byteCount)

説明: Nバイトのデータを、メモリ アレイから送信用 TXバッファに入れます。

パラメータ: uint8* 文字列: RAMまたはROMに存在するメモリ アレイのアドレス
uint8/uint16 byteCount: 送信するバイトの数 タイプは、**TX Buffer Size**パラメータによって異なります。

返り値: 無効

副作用: TXバッファに、アレイ全体に対する十分なメモリがない場合、この機能は、アレイの最後のバイトがTXバッファにロードされるまでブロックします。

void UART_PutCRLF(uint8 txDataByte)

説明: 1バイトのデータとその後に続くキャリッジ リターン(0x0D)およびライン フィード(0x0A)を送信バッファに書き込みます。

パラメータ: uint8 txDataByte: キャリッジ リターンおよびライン フィードの前に送信するデータ バイト

返り値: 無効

副作用: TXバッファに3バイトすべてに対する十分なメモリがない場合、この機能は、3バイトの最後がTXバッファにロードされるまでブロックします。

uint8/uint16 UART_GetTxBufferSize(void)

説明: TXバッファで使用するバイト数を決定します。空のバッファは0を返します。

パラメータ: 無効

返り値: uint8/uint16: TXバッファで使用されるバイトの数。タイプは**TX Buffer Size**パラメータによって異なります。

副作用: なし

void UART_ClearTxBuffer(void)

説明: TXバッファおよびハードウェアTX FIFOのすべてのデータをクリアします。

パラメータ: 無効

返り値: 無効

副作用: 送信バッファで待機しているデータは、送信されません。現在送信中のバイトの送信は終了します。

void UART_SendBreak(uint8 retMode)

説明: バス上にブレーク信号を送信します。

パラメータ: uint8 retMode: Send Breakリターン モード。オプションについては以下の表を参照してください。

オプション	説明
UART_SEND_BREAK	レジスタをブレーク用に初期化し、ブレーク信号を送信して、直ちに戻ります
UART_WAIT_FOR_COMPLETE_REINIT	ブレーク送信が完了するまで待ち、レジスタを通常の送信モードに再初期化して、戻ります
UART_REINIT	レジスタを通常の送信モードに再初期化して、戻ります
UART_SEND_WAIT_REINIT	両方のオプションを実行します: UART_SEND_BREAK および UART_WAIT_FOR_COMPLETE_REINIT。このオプションは、ほとんどのケースで使用を推奨します

返り値: 無効

副作用: UART_SendBreak()関数はレジスタを初期化して、ブレーク信号を送信します。ブレーク信号の長さは、ブレーク信号ビットの設定によって異なります。レジスタの設定は、通常の8ビット通信を継続する前に、再初期化する必要があります。

void UART_SetTxAddressMode(uint8 addressMode)

説明: 次のバイトがアドレスかデータかを示すように、トランスミッタを設定します。

パラメータ: uint8 addressMode:

オプション	説明
UART_SET_SPACE	次のバイトをデータとして送信するように、トランスミッタを設定します。
UART_SET_MARK	次のバイトをアドレスとして送信するように、トランスミッタを設定します。

返り値: 無効

副作用: この関数は、コントロールレジスタのUART_CTRL_MARKビットのセットやクリアを行います。

void UART_LoadRxConfig(void)

説明: 半二重モード用のレシーバ設定をロードします。この関数を呼び出した後、UARTではデータを受信する準備が完了します。

パラメータ: 無効

返り値: 無効

副作用: 半二重モードでのみ有効です。前のトランザクションが完了し、トランスミッタの設定をアンロードしても安全であることを、確認する必要があります。

void UART_LoadTxConfig(void)

説明: 半二重モードでのトランスミッタ設定を、ロードします。この機能を呼び出すと、UARTでデータを送信する準備が完了します。

パラメータ: 無効

返り値: 無効

副作用: 半二重モードでのみ有効です。前のトランザクションが完了し、レシーバの設定をアンロードしても安全なことを、確認する必要があります。

void UART_Sleep(void)

説明: これが、コンポーネントにスリープの準備をする推奨APIです。UART_Sleep() APIは、現在のコンポーネントの状態を保存します。その後UART_Stop()関数が呼び出され、UART_SaveConfig()が呼び出されてハードウェア設定が保存されます。

CCyPmSleep()またはCyPmHibernate()関数を呼び出す前に、UART_Sleep()関数を呼び出します。パワー マネージメント機能の詳細については、PSoC Creator の『 *System Reference Guide* 』を参照してください。

パラメータ: 無効

返り値: 無効

副作用: なし

void UART_Wakeup(void)

説明: これは、コンポーネントをUART_Sleep()が呼び出されたときの状態に復元する推奨APIです。UART_Wakeup()関数は、UART_RestoreConfig()関数を呼び出して設定を復元します。UART_Sleep()関数が呼び出される前にコンポーネントがイネーブルされた場合、UART_Wakeup()関数もコンポーネントを再イネーブルします。

パラメータ: 無効

返り値: 無効

副作用: この関数はRXとTXソフトウェア バッファおよびハードウェアFIFOをクリアしますが、ハードウェア ステート マシンはリセットされません。最初にUART_Sleep()またはUART_SaveConfig()関数を呼び出さないで、UART_Wakeup()関数を呼び出すと、予想しない動作が発生することがあります。

void UART_Init(void)

説明: カスタマイザのConfigureダイアログの設定に従って、コンポーネントを初期化または復元します。UART_Init()を呼び出す必要はありません。その理由は、UART_Start() APIがこの関数を呼び出し、これがコンポーネントの操作を始める推奨法だからです。

パラメータ: なし

返り値: なし

副作用: 全レジスタは、カスタマイザのConfigureダイアログの設定に従って、値が設定されます。

void UART_Enable(void)

説明: ハードウェアの使用を開始し、コンポーネントの動作を開始します。UART_Enable()を呼び出す必要はありません。その理由は、UART_Start() APIがこの関数を呼び出し、これがコンポーネントの操作を始める推奨法だからです。

パラメータ: なし

返り値: なし

副作用: なし

void UART_SaveConfig(void)

説明: この関数はコンポーネントの設定および非保持レジスタが保存します。現在のコンポーネント パラメータ値も、Configureダイアログに定義されているものや、または適切なAPIによって変更されているものと同様に保存されます。この機能は、UART_Sleep()関数によって呼び出されます。

パラメータ: なし

返り値: なし

副作用: FIFOを除くすべての非保持型レジスタは、RAMに保存されます。

void UART_RestoreConfig(void)

説明: 非保持型レジスタのユーザ設定を復元します。

パラメータ: なし

返り値: なし

副作用: FIFOを除く、RAMからロードされたすべての非保持型レジスタ。この関数が呼び出されるのはUART_SaveConfig()が呼び出された後のみとする必要があります。さもないと、間違ったデータがレジスタにロードされます。

定義

以下の定義は、参照のみを目的として、提供されています。定義の値は、コンポーネント カスタマイザ設定によって決定されます。

定義	説明
UART_INIT_RX_INTERRUPTS_MASK	設定GUIで選択する割り込みソースの初期設定を、定義します。これは、ステータスレジスタ中のビットのうち、コンフィギュレーション時にRX割り込みのソースとしてイネーブルされたもののマスクデータです。

UART_INIT_TX_INTERRUPTS_MASK	設定GUIで選択する割り込みソースの初期設定を、定義します。これは、ステータスレジスタ中のビットのうち、コンフィギュレーション時にTX割込み用のソースとしたもののマスクです。
UART_TXBUFFERSIZE	TXメモリ アレイ バッファに割り当てるメモリの量を、定義します。これには、FIFOに含まれている4バイトは含まれていません。
UART_RXBUFFERSIZE	RXメモリ アレイ バッファに割り当てるメモリの量を、定義します。これには、FIFOに含まれている4バイトは含まれていません。
UART_NUMBER_OF_DATA_BITS	1データ転送当たりのビット数を定義します。この値は、ビットクロック発生器およびビット カウンタのコンフィギュレーションレジスタの計算に使用されます。
UART_BIT_CENTER	データビットの数に基づいて、この値はRXビットクロック発生器の中心点の計算に使用されます。この値は、UARTの起動時にコンフィギュレーションレジスタにロードされます。
UART_RXHWADDRESS1	設定GUIで選択された初期アドレスを、定義します。このアドレスは、UARTの起動時に、対応するハードウェアレジスタにロードされます。
UART_RXHWADDRESS2	設定GUIで選択された初期アドレスを、定義します。このアドレスは、UARTの起動時に、対応するハードウェアレジスタにロードされます。

ファームウェアソースコードの例

PSoC Creator は Find Example Project ダイアログの中で回路図およびコードのサンプルが含まれている、多くのサンプルプロジェクトを提供しています。コンポーネント特有のサンプルを見るには、Component Catalog または回路図に置いたコンポーネントインスタンスからダイアログを開きます。一般例については、[Start Page (スタート ページ)] または **[File (ファイル)]** メニューからダイアログを開きます。必要に応じてダイアログにある **Filter Options** を使用し、選択できるプロジェクトのリストを絞り込みます。

詳細については、PSoC Creator のヘルプの「Find Example Project」トピックを参照してください。

機能説明

UART コンポーネントにより、一般に RS232 または RS485 と言われる同期通信が、提供されます。UART は、全二重、半二重、RX 専用、または TX 専用の動作に、設定できます。以下のセクションで、UART コンポーネントの使用方法の概要を、説明します。

初期設定

UART の初期設定は、フローコントロールなし、パリティなし、ボーレート 57.6 Kbp 動作の 8 ビット UART です

UART モード: フル UART (RX+TX)

このモードでは、非同期のレシーバおよびトランスミッタで構成される全二重 UART が実装されます。このモードでは、レシーバとトランスミッタの両方にボーレートを定義するために、単一のクロックが必要です。

UART モード: Half Duplex

このモードではフル UAR が実装されますが、フル UART 構成の半分のリソースが使用されます。この構成では、UART を RX モードと TX モードの間で切り替えるように設定できますが、RX 動作と TX 動作を同時に実行することはできません。RX または TX 構成は、UART_LoadRxConfig()または UART_LoadTxConfig()関数を呼び出すことによって、ロードできます。

このモードでは、**TX – On FIFO Not Full** ステータスを使用できませんが、その代わりに **TX – On FIFO Full** ステータスを使用できます。このモードでは TX 割り込みを使用できないため、TX バッファのサイズは 4 バイトに制限されています。

Half Duplex モードでは、Address2 パラメータは、ハードウェアアドレス一致ステータス (UART_RX_STS_ADDR_MATCH)で機能しませんが、ソフトウェアでは使用できます。

[Half Duplex (半二重)] モードの例 :

- このサンプルでは、コンポーネントが UART_1 という名前でデザイン中に配置されていると仮定しています。
- UART を **Mode**: Half Duplex、**Bits per second**: 115200、**Data bits**: 8、**Parity Type**: None、**Rx Buffer Size**: 4、**Tx Buffer Size**:4 に設定します。

```
#include <device.h>

void main()
{

    uint8 recByte;
    uint8 tmpStat;

    CyGlobalIntEnable;                  /* Enable interrupts */

    UART_1_Start();                    /* Start UART */
    UART_1_LoadTxConfig();              /* Configure UART for transmitting */
    UART_1_PutString("Half Duplex Test");   /* Send message */
    /* make sure that data has been transmitted */
    CyDelay(30);    /* Appropriate delay could be used */
                    /* Alternatively, check TX_STS_COMPLETE status bit */
    UART_1_LoadRxConfig(); /* Configure UART for receiving */
    while(1)
    {
        recByte = UART_1_GetChar();     /* Check for receive byte */
        if(recByte > 0)                 /* If byte received */
        {
            UART_1_LoadTxConfig();      /* Configure UART for transmitting */
            UART_1_PutChar(recByte);    /* Send received byte back */
```

```
            do                          /* wait until transmission complete */
            {   /* Read Status register */
                tmpStat = UART_1_ReadTxStatus();
                /* Check the TX_STS_COMPLETE status bit */
            }while(~tmpStat & UART_1_TX_STS_COMPLETE);
            UART_1_LoadRxConfig();      /* Configure UART for receiving */
        }
    }
}
```

UART モード: RX Only

このモードでは、UART のレシーバの部分のみが実装されます。このモードでは、レシーバのボーレートを定義するために、シングルクロックが必要です。

UART モード: TX Only

このモードでは、UART のトランスミッタの部分のみが実装されます。このモードでは、トランスミッタのボーレートを定義するために、シングルクロックが必要です。

UART Flow Control: None, Hardware

UART でのフローコントロールにより、個別の RX および TX ステータス表示ラインが、既存のバスに提供されます。ハードウェアのフローコントロールがイネーブルな場合、この UART と別の UART の間に、「Request to Send」(RTS)ラインと「Clear to Send」(CTS)ラインを使用できます。CTS ラインは UART への入力であり、システム内の他の UART が、このバスでのデータ送信が OK の場合セットされます。RTS ラインは UART の出力であり、この UART がバス上の他の UART に対して、データ受信準備が完了していることを、通知します。1 つの UART の RTS ラインは他の UART の CTS ラインに、およびその逆に、接続されます。これらのラインは、送信が開始される前のみイネーブルです。転送が開始された後で信号を設定またはクリアすると、その変更が影響するのは次の転送のみです。

UART Parity: なし

このモードには、パリティビットはありません。データ フローは「Start、Data、Stop」です。

UART Parity: Odd

奇数パリティは、パリティビット 1 から始まります。データストリームで 1 が見つかるごとに、パリティビットがトグルされます。データ送信の終りに、パリティビットの状態が送信されます。奇数パリティにより、UART バスに常に移行があることが、確認されます。すべてのデータがゼロの場合、送信されるパリティビットは 1 です。データ フローは「Start、Data、Parity、Stop」です。奇数パリティは、使用される最も一般的なパリティ タイプです。

UART Parity: Even

偶数パリティは、パリティビット 0 から始まります。データストリームで 1 が見つかるごとに、パリティビットがトグルされます。データ送信の終りに、パリティビットの状態が送信されます。データ フローは「Start、Data、Parity、Stop」です。

UART Parity: Mark/Space、データビット: 9

マーク/スペースパリティは、送信されたデータがアドレス データであったか、標準データであったかを定義するために、最も通常的に使用されます。パリティビットのマーク(1)はデータが送信されたことを示し、パリティビットのスペース(0)はアドレスが送信されたことを示します。マークまたはスペースは、データ送信のパリティビットの位置で、送信されます。データフローは、他のパリティモードと同様に、「Start、Data、Parity、Stop」ですが、このビットは、データビット値に基づいて計算されるのではなく、転送前にソフトウェアによって設定されます。このパリティは、RS485 および同様のプロトコルに対して使用できます。

TX 利用モデル

トランスミッタをパケットの最初のアドレスバイトに対して設定するために、ファームウェアで UART_SetTxAddressMode API を、UART_SET_MARK パラメータによって使用する必要があります。この API では、コントロールレジスタで UART_CTRL_MARK ビットをセットします。MARK パリティを設定した後、送信される最初のバイトはアドレスで、残りのバイトは SPACE パリティ付きデータとして送信されます。トランスミッタにより、最初のアドレスバイトの後に、データバイトが自動的に送信されます。他のパケットを送信する前に、コントロールレジスタの UART_CTRL_MARK ビットは、1 クロック以上の期間、クリアされる必要があります。これは、UART_SET_SPACE パラメータを使用して UART_SetTxAddressMode API を呼び出すことによって、実行できます。これは、以下のサンプルコードに示されています。

アドレス指定で送信されたパケットの例:

- このサンプルでは、コンポーネントは、UART_TX の名前で設計に配置されていると仮定しています。
- UART を**データビット**: 9、**パリティタイプ**: Mark/Space に設定します。

```
#include <device.h>

void main()
{
    UART_TX_Start();
    /*Set UART_CTRL_MARK bit in Control register*/
    UART_TX_SetTxAddressMode(UART_TX_SET_MARK);
    /*Send data packet with the address in first byte*/
    /*The address byte is character '1', which is equal to 0x31 in hex format*/
    UART_TX_PutString("1UART TEST\r");

    /*Clear UART_CTRL_MARK bit in Control register*/
    UART_TX_SetTxAddressMode(UART_TX_SET_SPACE);
}
```

RX 使用モデル

レシーバには 4 つの異なるモードがあります。

1. ソフトウェアバイト単位

このモードは、カスタムコードが必要なときに使用します。

ステータスレジスタの UART_RX_STS_MRKSPC ビットは、アドレスまたはデータバイトがレシーバに到着したことを示します。

アドレス指定で受信したパケットの例：

- このサンプルでは、コンポーネントは名前 UART_RX で設計に配置されていると仮定します。
- UART を**データビット**: 9、**パリティタイプ**: Mark/Space、**割り込み**: RX - On Byte Received、**アドレスモード**: Software Byte by Byte, **Address#1**: 31。
- 外部の ISR を rx_interrupt ピンに、名前「isr_rx”」で接続します。

```
#include <device.h>

#define STR_LEN_MAX     60u
char rx_buffer[STR_LEN_MAX];
uint8 packet_receivedRX = 0u;

void main()
{
    CyGlobalIntEnable;          /* Enable interrupts */
    isr_rx_Start();
    UART_RX_Start();

    if(packet_receivedRX == 1u)
    {
        /* add analyze here */
        packet_receivedRX = 0u;
    }
}
```

ISR ルーチンのソースコードサンプル

```
uint8 rec_status = 0u;
uint8 rec_data = 0;
static uint8 pointerRX = 0u;
static uint8 address_detected = 0u;

rec_status = UART_RX_RXSTATUS_REG;
if(rec_status & UART_RX_RX_STS_FIFO_NOTEMPTY)
{
    rec_data = UART_RX_RXDATA_REG;
    if(rec_status & UART_RX_RX_STS_MRKSPC)
    {
        if (rec_data == UART_RX_RXHWADDRESS1) /* Use any other address */
```

```
            {
                address_detected = 1;
            }
            else
            {
                address_detected = 0;
            }
        }
        else
        {
            if(address_detected)
            {
                if(pointerRX >= STR_LEN_MAX)
                {
                    pointerRX = 0u;
                }
                /* Detect end of packet */
                if(rec_data == '\r')
                {   /* write null terminated string */
                    rx_buffer[pointerRX++] = 0u;
                    pointerRX = 0u;
                    paket_receivedRX = 1u;
                }
                else
                {
                    rx_buffer[pointerRX++] = rec_data;
                }
            }
        }
    }
```

2. バッファに対するソフトウェア検出

必要なコードはすべて、このモードの RX ISR に実装されます。

- UART を**データビット**: 9、**パリティタイプ**: Mark/Space、**RX バッファサイズ**: 20、**アドレスモード**: Software Detect to Buffer、**Address#1**: 31。

アドレス指定で受信したパケットの例：

```
void main()
{
    uint8 rec_data = 0u;

    CyGlobalIntEnable;          /* Enable interrupts */
    UART_RX_Start();
    for(;;)
    {
        rec_data = UART_RX_GetChar();
        if(rec_data > 0u)
        {
            /* add analyze here */
        }
    }
}
```

3. Hardware Byte By Byte

ハードウェアによって、アドレス指定されていないパケットが除外されます。このモードのメインコードは、前のサンプルと同様です。

- UART を以下のように設定。**データビット**: 9、**パリティタイプ**: Mark/Space、**RX バッファサイズ**: 20、**アドレスモード**: Hardware Byte By Byte、**Address#1**: 31。

4. バッファに対するハードウェア検出

これは、アドレスバイトを必要としないプロジェクトに対する推奨モードです。ハードウェアにより、アドレスバイト内のアドレス指定されていないパケットが除外されます。メインコードは、アドレス指定されたデータのみのバイトを、受信します。

- UART を**データビット**: 9、**パリティタイプ**: Mark/Space、**RX バッファサイズ**: 20、**アドレスモード**: バッファに対するハードウェア検出、**Address#1**: 31。

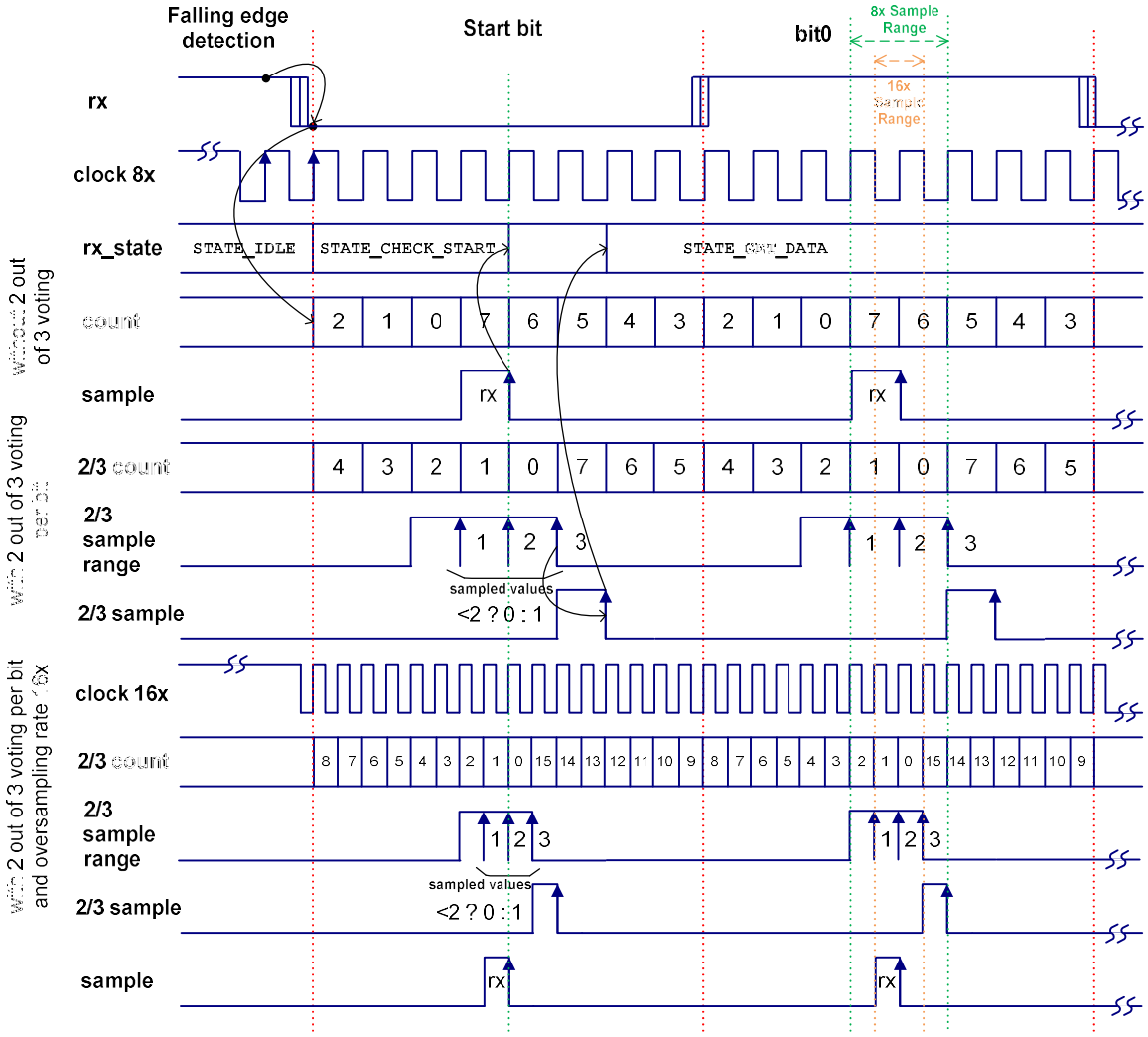
UART ストップビット: 1、2

ストップビットの数は、同期化メカニズムとして使用できます。低速のシステムでは、次のデータが送信される前に受信側でデータを処理できるように、ストップ コマンドが 2 ビット時間を占有することが必要になることがあります。トランスミッタが 2 ビット幅のストップ信号を送信することにより、レシーバはデータバイトとパリティを解釈する猶予期間を与えられます。、2 番目のストップビットは、レシーバでフレーミングエラーを確認されません。データフローは、同じ「Start、Data、[Parity]、Stop」です。ストップビットの時間は、1 ビット幅または 2 ビット幅に設定できます。

2 out of 3 ボーティング

2/3 ボーティング機能により、エラー補正アルゴリズムがイネーブルになります。このアルゴリズムでは、基本的に各ビットの中央が 3 回オーバーサンプリングされ、多数決でそのビットが 0 か 1 かを判断します。2/3 ボーティングがイネーブルではない場合、各ビットの中央がサンプリングされるのは 1 回のみです。

イネーブルの場合、このパラメータには、3 オーバーサンプリングクロックサイクル用の RX 入力に基づく 3 ビットカウンタを実装するために、追加のハードウェアリソースが必要です。以下の図に、2/3 ボーティングありおよびなしの、8 ビットおよび 16 ビットオーバーサンプリングの実装が、示されています。



立下りエッジ検出は、スタートビットを識別するために、実装されます。これが検出されると、カウンタは半ビット長から 0 までダウンカウントを開始し、レシーバは CHECK_START 状態に切り替わります。カウンタが 0 になると、RX ラインは 3 回サンプリングされます。RX ラインが LOW (たとえば 3 ビットのうち 2 ビット以上が 0)と検証された場合、レシーバは GET_DATA 状態になります。そうでない場合、レシーバは IDLE 状態に戻ります。スタートビット検出シーケンスは、8x または 16x オーバーサンプリングレートと同じです。

レシーバが GET_DATA 状態になっていると、RX 入力がカウンタサイクル 4～6 (3 サイクル)でイネーブルになるカウンタへの入力になります。このカウンタは、RX 入力に表示される 1 の数をカウントします。カウンタの値が 2 以上の場合、このカウンタの出力は 1 です。2 未満の場合、0 です。この値は、7 番目のクロックエッジの RX 値として、データパスにサンプル提供されます。2/3 ボーテイングがイネーブルになっていない場合、RX 入力は、スタートビットの検出後、単に 5 番目のクロックエッジでサンプリングされるだけで、その後 8 番目の正のクロックエッジごとに継続されます。

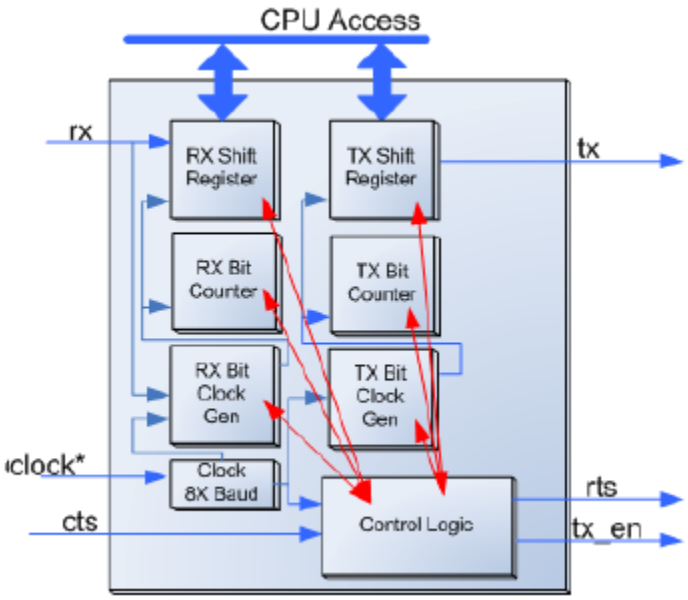
オーバーサンプリングレート 16x がイネーブルになると、ボーティング アルゴリズムがカウンタサイクル 8～10 で作動し、カウンタの出力が、RX 値として 11 番目のサイクルでデータパスによって、サンプリングされます。ボーティングがイ

ネーブルでない場合、RX 入力は 9 番目のクロックエッジでサンプリングされ、その後 16 番目のクロックエッジごとに継続されます。

ブロック ダイアグラムと設定

UART が UDB ブロックに実装され、図 1 に説明されています。

図 1. UDB の実装



レジスタ

これまでに説明した API 関数により、ほとんどのアプリケーションに必要な一般的ランタイム機能のサポートが提供されます。以下のセクションで、上級ユーザーを対象として、UART レジスタを簡潔に説明します。

RX および TX のステータス

ステータスレジスタ(RX および TX には独立したステータスレジスタがあります)は読み取り専用レジスタであり、UART 向けに定義されたさまざまなステータスビットが含まれています。これらのレジスタの値には、UART_ReadRxStatus()および UART_ReadTxStatus()関数呼び出しを使用して、アクセスできます。

割り込み出力信号(tx_interrupt および rx_interrupt)は、各レジスタ内のマスクされたビットフィールドの論理和(OR)を取ったものにより生成されます。マスクは、UART_SetRxInterruptMode()および UART_SetTxInterruptMode()関数呼び出しを使用して、設定できます。割り込みを受信すると、割り込みソースは、UART_GetRxInterruptSource()および UART_GetTxInterruptSource()関数呼び出しを使用して、該当するステータスレジスタを読み取ることによって、取得できます。ステータスレジスタは読み取り後クリアで

あり、割り込みソースは、UART_ReadRxStatus()または UART_ReadTxStatus()関数のいずれかが呼び出されるまで、保持されます。ステータスレジスタのすべての操作には、ビットフィールドに対する以下の定義を使用する必要があります。その理由は、これらのビットフィールドは、ビルド時間にステータスレジスタ内で移動することがあるからです。

ステータスレジスタ用に、いくつかのビットフィールド マスクが定義されています。どのビットフィールドでも、割り込みソースとして含めることができます。#defines は、生成されたヘッダファイル(.h)で使用できます。

ステータス データは UART の入力クロックエッジに登録されています。これらのビットのいくつかはスティッキーで、ステータスレジスタの読み取り時にクリアされます。これらは、UART への割り込み出力として使用するために、読み取り後クリアとして割り当てられます。その他のすべてのビットは透過として設定されており、ステータスレジスタの入力から直接、データを表示します。これらはスティッキーではなく、そのために読み取り後クリアではありません。

スティッキーとして定義されたビットはすべて、以下の定義にアスタリスク(*)付きで示されています。

RX ステータスレジスタ

デファイン	説明
UART_RX_STS_MRKSPC *	マーク/スペースパリティビットのステータス。このビットは、マークまたはスペースが、転送のパリティビット位置に表示されたかを示します。アドレスモードが「なし」に設定されていない場合のみ、実装されます。
UART_RX_STS_BREAK *	ブレーク信号が転送時に検出されたことを示します。
UART_RX_STS_PAR_ERROR *	パリティエラーが転送時に検出されたことを示します。
UART_RX_STS_STOP_ERROR *	このビットは、フレーミングエラーを示します。UARTハードウェアで、ストップ ビットがあるべき場所(ロジック1)にロジック0が表示されると、フレーミングエラーが発生します。
UART_RX_STS_OVERRUN *	受信FIFOバファがオーバーランされていることを示します。
UART_RX_STS_FIFO_NOTEMPTY	受信FIFOがNot Emptyかどうかを示します。
UART_RX_STS_ADDR_MATCH *	受信したバイトが、ハードウェアのアドレス検出に使用できる2つのアドレスの1つに一致することを示します。アドレスモードが「なし」に設定されていない場合のみ、実装されます。**Half Duplex**モードでは、この検出用に**Address #1**のみが実装されます。

TX ステータスレジスタ

デファイン	説明
UART_TX_STS_FIFO_FULL	送信FIFOがフルであることを示します。これは、メモリに実装されている送信バッファと混同しないようにしてください。つまり、このバッファのステータスは、ハードウェアでは表示されず、ファームウェアで確認する必要があります。
UART_TX_STS_FIFO_NOT_FULL**	送信FIFOがフルではないことを示します。
UART_TX_STS_FIFO_EMPTY	送信FIFOが空であることを示します。
UART_TX_STS_COMPLETE *	最後のバイトがFIFOから送信されたことを示します。

** - 半二重モードでは使用できません

制御

コントロールレジスタを使うと、UART の一般動作をコントロールできます。このレジスタでは、UART_WriteControlRegister()関数を使用して書き込み、UART_ReadControlRegister()関数を使用して読み取ります。カスタマイザでシンプル UART オプションが選択されている場合、コントロールレジスタは使用されません。詳細については、リソース を参照してください。コントロールレジスタの読み取りまたは書き込みをするとき、ヘッダ(.h)ファイルに定義されているビットフィールドの定義を、使用する必要があります。コントロールレジスタ用#defines は、以下のとおりです。

UART_CTRL_HD_SEND

半二重モードでの RX と TX の間の動作を動的に再設定するために、使用します。このビットは、UART_LoadTxConfig()関数によってセットされ、UART_LoadRxConfig()関数によってクリアされます。

UART_CTRL_HD_SEND_BREAK

セットすると、バスのブレーク信号が送信されます。このビットは UART_SendBreak()関数によって、書き込まれます。

UART_CTRL_MARK

送信バイトのマーク/スペースパリティの動作をコントロールするために、使用します。セットすると、このビットは、バスで送信された次のバイトが、パリティビットの位置に 1 (マーク)を含むことを、示します。後続するバイトはすべて、このビットがファームウェアによってクリアしてリセットされるまで、パリティビットの位置に 0 (スペース)が含まれます。

UART_CTRL_PARITY_TYPE_MASK

パリティタイプコントロールは、次の転送のパリティ動作を定義する 2 ビット幅フィールドです。このビットフィールドは、コントロールレジスタの 2 つの連続したビットです。このビットフィールドでのすべての操作には、使用可能なパリティタイプに関連した#defines を使用する必要があります。これらは、以下のとおりです。

値	説明
UART__B_UART__NONE_REVB	パリティなし
UART__B_UART__EVEN_REVB	偶数パリティ
UART__B_UART__ODD_REVB	奇数パリティ
UART__B_UART__MARK_SPACE_REVB	マーク/スペース パリティ

このビットフィールドは、**Parity Type** 設定パラメータで定義したパリティタイプを使用して初期化時に設定され、動作中に UART_WriteControlRegister()関数呼び出しを使用して変更できます。

UART_CTRL_RXADDR_MODE_MASK

RX アドレスモードコントロールは、UART レシーバに要求されるハードウェアアドレス指定動作を定義するために使用する、3ビットフィールドです。このビットフィールドは、コントロールレジスタの３つの連続したビットです。このビットフィールドのすべての操作に、使用可能な比較モードに関連する#defines を使用する必要があります。これは、以下のとおりです。

値	説明
UART__B_UART__AM_SW_BYTE_BYTE	Software Byte by Byteアドレス検出
UART__B_UART__AM_SW_DETECT_TO_BUFFER	Software Detect to Bufferアドレス検出
UART__B_UART__AM_HW_BYTE_BY_BYTE	Hardware Byte by Byteアドレス検出
UART__B_UART__AM_HW_DETECT_TO_BUFFER	Hardware Detect to Bufferアドレス検出
UART__B_UART__AM_NONE	アドレス検出なし

このビットフィールドは、**Address Mode** 設定パラメータを使用して初期化時に設定され、UART_WriteControlRegister()関数呼び出しを使用して動作中に変更できます。

TX データ(8 ビット)

TX データレジスタには、送信するデータが含まれています。これは、FIFO として実装されます。送信するデータの大部分を処理するため、送信メモリバッファからのデータをコントロールするためのソフトウェアステートマシンがあります。ﾃﾞｰﾀの送信に対処するすべての関数は、データをバスに出力するためにこのレジスタを通す必要があります。このレジスタにデータがあり、フローコントロールがデータを送信できることを示している場合、データはバス上に送信されます。このレジスタ(FIFO)が空になると、それ以上のデータは、FIFO に追加されるまで送信されません。空のとき、DMA は、ヘッダファイルで定義された TX データレジスタアドレスを使用してこの FIFO を満たすように、セットアップできます。

値	説明
UART_TXDATA_REG	TXデータレジスタ

RX データ

RX データレジスタには、FIFO として実装された受信データが含まれています。この受信 FIFO からメモリバッファへのデータの移動をコントロールする、ソフトウェアステートマシンがあります。通常では RX 割り込みでは、データが受信されたこと、CPU または DMA を使用していつデータを取得できるかが、示されます。DMA は、FIFO が空でないときはいつでも、ヘッダファイルに定義された RX データレジスタアドレスを使用して、このレジスタからデータを取得するように、セットアップできます。

値	説明
UART_RXDATA_REG	RXデータレジスタ

定数

一部の列挙型と同様に、いくつかの定数がステータスレジスタやコントロールレジスタ向けに定義されます。これらのほとんどは、ステータスレジスタやコントロールレジスタについて、前述されています。ただし、ヘッダーファイルにはさらに必要な定数があります。各レジスタ定義には、レジスタデータへのポインタまたはレジスタアドレスが必要です。コンパイラにある複数のエンディアンのため、8ビットより長いレジスタにアクセスするには、CY_GET_REGXおよびCY_SET_REGX マクロを使用する必要があります。これらのマクロには、各レジスタに、_PTR で終わる定義を使用する必要があります。

コントロールおよびステータスレジスタビットは、ビルド時間にフィッタ エンジンによって配置および接続できる必要があります。定数は、ビットの配置を定義するために作成されます。個々のステータスおよびコントロールレジスタビットには、レジスタ内でのビット オフセットを定義する、関連する_SHIFT 値があります。これらはヘッダファイルで、最終ビット マスクを_MASK definition (拡張子_MASK は 1 ビットより大きいビットフィールドのみに追加され、すべての 1 ビット値では_MASK 拡張子はありません)として定義するために、使用されます。

DC 電気的特性と AC 電気的特性

以下の値は予想される性能を示し、初期の特性データに基づいています。

「公称ルーティングでの最大」タイミング特性

データ収集が進行中です。この表は、将来のリリースで更新されます。

パラメータ	説明		Min	Typ	Max	Units
f_{CLOCK}	コンポーネント クロック周波数 [1]					
		フルUART	–	–	24	MHz
		単純なUART	–	–	44	MHz
		半二重UART	–	–	50	MHz
		RX専用	–	–	66	MHz
		TX専用	–	–	58	MHz
t_{CLOCK}	クロック周期		$1/f_{CLOCK}$	–	–	ns
f_b	ビット レート		–	–	f_{CLOCK}/ オーバーサンプリング	Mbps
T_{CLOCK}	クロック周期許容誤差					
		8xオーバーサンプリング	–	2.6	–	%

[1] 最大コンポーネント クロック周波数は、選択したモードおよび追加機能によって、異なります。

パラメータ	説明		Min	Typ	Max	Units
		16xオーバーサンプリング	–	3.2	–	%
$\%_{ERR}$	エラー		–	STA [2]	–	%
t_{RES}	リセット パルス幅		t_{CLOCK} + 5	–	–	ns
t_{CTS_TX}	「CTS_N inactive」から「TX_EN active」へ、そして TX で bit を開始		1	–	2	t_{CLOCK}
t_{TX_TXDATA}	「TX」から「TX_DATA」へのディレー		–	1	–	t_{CLOCK}
t_{TX_TXCLK}	「TX change」から「TX_CLK active」へのディレー					
		8xオーバーサンプリング	–	5	–	t_{CLOCK}
		16xオーバーサンプリング	–	9	–	t_{CLOCK}
t_{S_RES}	設定時間をリセット		5	–	–	ns
t_{RTS_RX}	「RTS_N inactive」から「RX data」へ		–	–	STA [3]	ns
t_{RX_RXCLK} t_{RX_RXINT}	「RX」から「RX_CLK」へのディレー					
		8xオーバーサンプリング	4	–	5	t_{CLOCK}
		16xオーバーサンプリング	8	–	9	t_{CLOCK}
t_{RXCLK_RTS}	最後の「RX_CLK raise」から「RTS_N active」へのディレー		–	1	–	t_{CLOCK}
t_{RX_RXDATA}	「RX」から「RX_DATA」へのディレー		0	–	1	t_{CLOCK}

[2] $\%_{ERR}$は、PSoC Creator が正確な周波数クロック周期を生成できないときに、システムに存在します。その値は、このデータシートに後述するように、計算する必要があります。

[3] t_{RTS_RX} 値は「Static Timing Analysis(静的タイミング解析)」結果に応じ、このデータシートで後述するように計算する必要があります。

「すべてのルーティングでの最大[4]」タイミング特性

データ収集が進行中です。この表は、将来のリリースで更新されます。

パラメータ	説明		Min	Typ	Max	Units
f_{CLOCK}	コンポーネント クロック周波数[5]					
		フルUART	–	–	12	MHz
		単純なUART	–	–	22	MHz
		半二重UART	–	–	25	MHz
		RX専用	–	–	33	MHz
		TX専用	–	–	29	MHz
t_{CLOCK}	クロック周期		$1/f_{CLOCK}$	–	–	ns
f_b	ビット レート		–	–	f_{CLOCK}/ オーバーサンプリング	Mbps
T_{CLOCK}	クロック許容誤差					
		8xオーバーサンプリング	–	2.6	–	%
		16xオーバーサンプリング	–	3.2	–	%
$\%_{ERR}$	エラー		–	STA [6]	–	%
t_{RES}	リセット パルス幅		t_{CLOCK} + 5	–	–	ns
t_{CTS_TX}	「CTS_N inactive」から「TX_EN active」へ、そして TX で bit を開始		1	–	2	t_{CLOCK}
t_{TX_TXDATA}	「TX」から「TX_DATA」へのディレー			1	–	t_{CLOCK}
t_{TX_TXCLK}	「TX change」から「TX_CLK active」へのディレー					
		8xオーバーサンプリング	–	5	–	t_{CLOCK}
		16xオーバーサンプリング	–	9	–	t_{CLOCK}
t_{S_RES}	設定時間をリセット		5	–	–	ns

[4] ルーティング時間数の最大値は、公証ルーティング時間数を 2 の倍数で除算して得られます。お使いのコンポーネントインスタンスがこの速度と同じ、または遅いスピードで動作している場合は、ミーティングタイミングはこのコンポーネントで問題にはなりません。

[5] 最大コンポーネント クロック周波数は選択したモードと追加機能によります。

[6] $\%_{ERR}$ は、PSoC Creator が正確な周波数クロック周期を生成できないときに、システムに存在します。その値は、このデータシートに後述するように計算する必要があります。

パラメータ	説明		Min	Typ	Max	Units
t_{RTS_RX}	「RTS_N inactive」から「RX data」へ		–	–	STA [7]	ns
t_{RX_RXCLK} t_{RX_RXINT}	「RX」から「RX_CLK」へのディレー					
		8xオーバーサンプリング	4	–	5	t_{CLOCK}
		16xオーバーサンプリング	8	–	9	t_{CLOCK}
t_{RXCLK_RTS}	最後の「RX_CLK raise」から「RTS_N active」へのディレー		–	1	–	t_{CLOCK}
t_{RX_RXDATA}	「RX」から「RX_DATA」へのディレー		0	–	1	t_{CLOCK}

「Full UART」オプション:

Mode (モード)*	Full UART
パリティ:	偶数
API コントロールが有効:	有効
フローコントロール:	ハードウェア (ピン)
アドレスモード:	ソフトウェアバイト単位
RX バッファサイズ (バイト)	5
TX バッファサイズ (バイト)	5
ブレイク信号ビット:	13
2/3 ボーティングアルゴリズム:	有効
CRC 出力:	有効 (出力ピン)
ハードウェア TX:	有効 (出力ピン)
オーバーサンプリング率:	16x
リセット:	入力ピン

「Simple UART」オプション:

Mode (モード)*	Full UART
パリティ:	なし
API コントロールが有効:	無効
フローコントロール:	なし
アドレスモード:	なし
RX バッファサイズ (バイト)	4
TX バッファサイズ (バイト)	4
ブレイク信号ビット:	なし
2/3 ボーティングアルゴリズム:	無効
CRC 出力:	無効

[7] t_{RTS_RX} 値は「Static Timing Analysis(静的タイミング解析)」結果に応じ、このデータシートで後述するように計算する必要があります。

ハードウェア TX: 無効
オーバーサンプリングレート: 8x
リセット: なし

「Half Duplex UART(半二重 UART)」オプション:

Mode (モード)* Half Duplex
他の全てのオプションは Simple UART と同じ

「RX Only options(RX オンリー)」オプション:

Mode (モード)* 「RX Only(RX オンリー)」
他の全てのオプションは Simple UART と同じ

「TX Only options(TX オンリー)」オプション:

Mode (モード)* 「TX Only(TX オンリー)」
TxBitClkGenDP False (スイッチするには「Advanced tab(詳細タブ」の「Expression View(エクスプレッションビュー)」へ)。
他の全てのオプションは Simple UART と同じ

図 2. TX モードタイミングダイアグラム

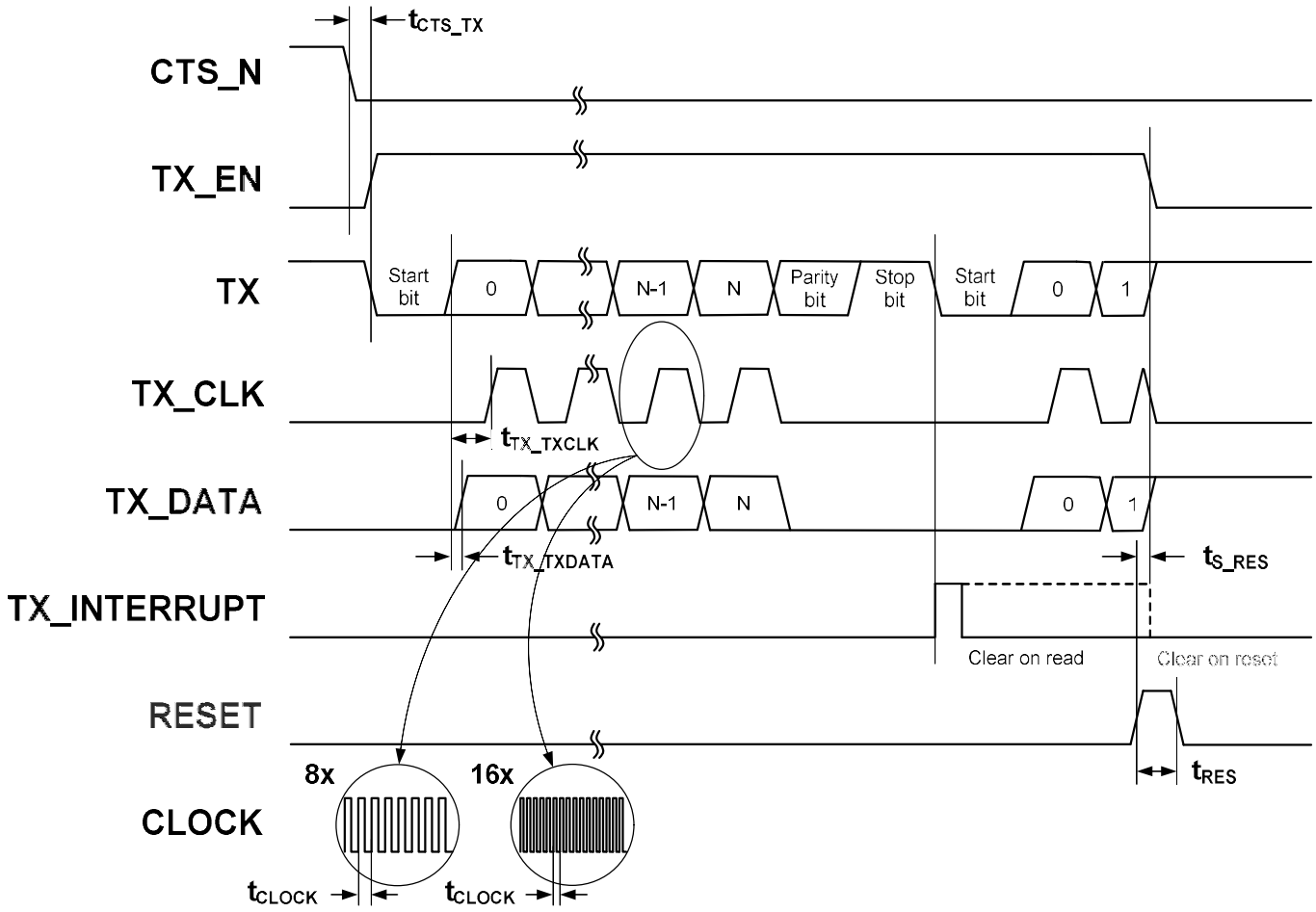
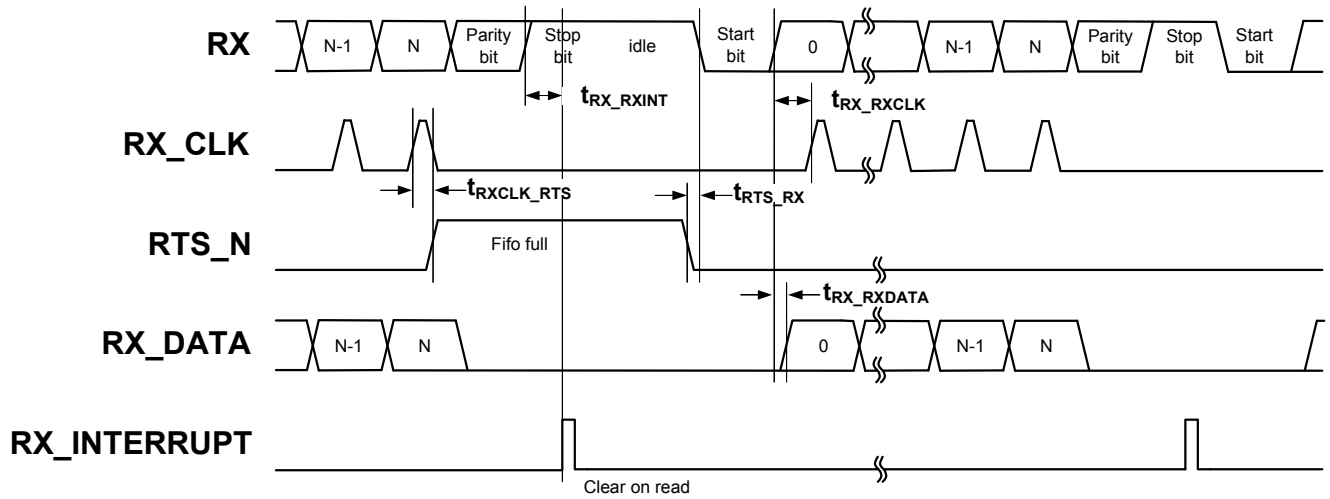


図 3. RX モードタイミングダイアグラム



特性データ用の STA 結果の使用方法

公称ルーティング最大値は、静的タイミング分析 (STA) を使って、複数のテスト パスから収集されます。STA 結果を用いた場合、次の手法で設計の最大値を計算できます。

f_{CLOCK} 最大コンポーネントのクロック周波数はクロックサマリーの時間結果に IntClock として (内部クロックが選択されている場合) または、名前を付けた外部クロックとして表示されます。下図は、*_timing.html* によるクロック制限の例を示しています。

- **Clock Summary Section**

Clock	Type	Nominal Frequency (MHz)	Required Frequency (MHz)	Maximum Frequency (MHz)	Violation
BUS_CLK	Sync	66.000	66.000	N/A	
ClockBlock/clk_bus	Async	66.000	66.000	N/A	
ClockBlock/dclk_0	Async	0.917	0.917	N/A	
ILO	Async	0.001	0.001	N/A	
IMO	Async	3.000	3.000	N/A	
MASTER_CLK	Sync	66.000	66.000	N/A	
PLL_OUT	Async	66.000	66.000	N/A	
UART_1_IntClock	Sync	0.917	0.917	39.588	

t_{CLOCK} 以下の方程式からクロック期間を計算します:

$$t_{CLOCK} = \frac{1}{f_{CLOCK}}$$

f_b ビットレートは、クロック周波数 (f_{CLOCK}) をオーバーサンプリング率で除した答えに等しい。最大ボーレートの計算には、以下の方程式に示すように、オーバーサンプリング率 8x を使用する。

$$f_b = \frac{f_{CLOCK}}{\text{Oversampling}}$$

T_{CLOCK} 以下の方法からクロック許容範囲を計算します:

UART は 8x オーバーサンプリング率で設定され、2/3 ボーティングアルゴリズムは無効で、パリティーなし、そしてワンストップビットであると仮定します。レシーバーは RX ラインを毎ビットごとの 5 番目のクロックでサンプリングします。アクティブ ローのスタートビットの最初の立下りエッジで、新しいフレームを感知します。レシーバー UART はこの立下りエッジでカウンタをリセットし、4 クロックサイクル後にミッドスタートビットが発生し、それに続く各ビットの中間点が 8 クロック毎に現われるのを待ちます。UART クロックが 0 パーセントエラーであると、サンプリングはストップビットの丁度中間点で発生します。しかし、UART クロックが 0 エラーでない場合は、サンプリングは各ビットの中間点よりも早く、または遅く発生します。このエラーは累積し、結果としてストップビットの最大エラーを生じます。仮にビットを 1/2 ビット分 (8 ÷ 2 = ±4 クロック) 早く、または遅くサンプリングすると、ビット遷移でサンプリングするため、不正確なデータを得ます。ビット遷移時間は、通常信号品質では、ビットタイムの 25 パーセントに等しくなります。ストップビットの中間点で許されるエラーは、UART クロックの ±3 期間に等しくなります。

このエラー要因に含める、もう一つのエラーは、スタートビットの立下りエッジが検知された場合の同期エラーです。UART はスタートビット が検知された 8x クロック後の次の立ち上がりエッジで開始し

ます。8x クロックと受信データストリームは非同期なので、スタートビット の立下りエッジは 8x クロック立ち上がりエッジの直ぐ後で発生することがあります。このことは、UART は同期ポイントで、±1 clock のエラーが組み込まれていることを意味しています。その結果、エラー予算は ±2 期間に低減します。

スタートビット の立下りエッジからストップビットの中間までの総クロック期間は 9.5 × 8 = 76 に等しくなります。総クロック耐性は ±2 ÷ 76 × 100% = ±2.6% です。

16x オーバーサンプリングのクロック耐性は: (16 ÷ 2 × (1 – 0.25) – 1) ÷ (9.5 × 16) × 100% = ±3.2%

総耐性はレシーバーとトランスミッター間で任意の比率で分割します。例えば、UART バス (マイクロコントローラまたは PC) の片側のデバイスが、標準的な 100-ppm 水晶振動子で動作している場合、反対側のデバイスは誤差猶予分のほとんど全てを使用できます。

%ERR

PLL クロック周波数と分割機の値のため、PSoC Creator が UART が要求する正確な周波数クロックを生成できない場合に、このエラーがシステムに表示されます。CharComp_clock の望ましい周波数と公証周波数として、「design wide resources (DWR)」でその差を見ることができます。以下の方程式を使用してエラーを計算します:

$$\%_{ERR} = \frac{f_{des} - f_{nom}}{f_{des}} * 100\%$$

Type	Name	Domain	Desired Frequency	Nominal Frequency	Accuracy (%)	Tolerance (%)	Divider	Start on Reset	Source Clock
System	USB_CLK	DIGITAL	48.000 MHz	? MHz	±0	-	1	☐	IMOx2
System	Digital_Signal	DIGITAL	? MHz	? MHz	±0	-	0	☐	
System	XTAL_32KHZ	DIGITAL	32.768 kHz	? MHz	±0	-	0	☐	
System	XTAL	DIGITAL	25.000 MHz	? MHz	±0	-	0	☐	
System	ILO	DIGITAL	? MHz	1.000 kHz	-50, +100	-	0	☑	
System	IMO	DIGITAL	3.000 MHz	3.000 MHz	±1	-	0	☑	
System	BUS_CLK (CPU)	DIGITAL	? MHz	66.000 MHz	±1	-	1	☑	MASTER_CLK
System	MASTER_CLK	DIGITAL	? MHz	66.000 MHz	±1	-	1	☑	PLL_OUT
System	PLL_OUT	DIGITAL	66.000 MHz	66.000 MHz	±1	-	0	☑	IMO
Local	UART_1_IntClock	DIGITAL	921.600 kHz	916.667 kHz	±1	±5	72	☑	Auto: MASTER_CLK

Pins | Clocks | Interrupts | DMA | System | Directives | Flash Security

例えば、115200 ビット/秒 8x オーバーサンプリングで設定された UART では、システムは 921.6-kHz のクロックが必要です。PLL が 66 MHz で設定されている場合は、DWR は 72 で除すので、66000 ÷ 72 = 916,667-kHz クロックを生成します。この例では、エラーは:

(921.6 – 916,667) ÷ 912.6 × 100 = ~0.5%

このエラーとクロック精度エラーの総和はクロック耐性 (T_{CLOCK})を超えません。そうでない場合は、データにエラーを生じます。

クロック精度は選択した IMO クロックによります。3-MHz IMO では ±1% と等しくなります。全エラーは:0.5 + 1 = 1.5% で 8x オーバーサンプリング (2.6%) では、最低クロック耐性より小さくなります。

IMO クロックの他の設定では、精度エラーが大きくなるので、UART との使用は推奨しません。

PSoC 5 シリコンは ±5 パーセントの最小 IMO クロック精度です ; そのため、この場合は外部の水晶ベースのクロックを使用するべきです。

t_{CTS_TX} このパラメーターは UART の実装分析に基づいて特長づけられます。状態機械の f_{CLOCK} クロックへの同調は、立下りエッジ CTS_N 信号を確認し、1 クロックディレーまでのディレーで TX_EN を設定します。TX_EN 信号は出力上にさらに同期し、誤作動の可能性を取り除きます。これにより、1 クロックディレーが追加されます。TX_EN 信号が「high」になると同時に、シフトレジスタが TX データを外に押し出します。

t_{TX_TXCLK} TX 出力から TX_CLK へのディレータイムは、UART 実装分析に基づいており、1/2 ビット長と等しく、TX_DATA 信号の中央にいくためには 1 クロック遅れています。

$$t_{TX_TXCLK} = t_{CLOCK} * \left(\frac{\text{Oversampling}}{2} + 1 \right)$$

t_{TX_TXDATA} このパラメーターは UART の実装分析に基づいて特長づけられます。TX 信号は TX_DATA 出力上で f_{CLOCK} にさらに同期します ; そのため、これらの信号間には 1 クロックのディレーがあります。

t_{RES} このパラメーターは UART の実装分析と STA 分析結果に基づいて特徴づけられます。リセット入力が同期するには、コンポーネント クロックの少なくとも 1 つの立ち上がりエッジが必要です。リセット信号を確実に逃さないために、設定時間を加えるべきです。

$$t_{RES} = t_{CLOCK} + t_{S_RES}$$

t_{S_RES} 「RESET」アクティブ化時間とはピンから内部レジスタへのルーチンパスのディレー時間に、クロックから出力へのディレー時間を加えたものです。以下に示すように、STA 結果中に表示されます。

- **Register to Register Section**
  - **Setup Subsection**
    - **Source Clock : BUS_CLK : Positive edge(Required Frequency 33 MHz)**
      - **Destination Clock : UART_1_IntClock : Positive edge(Required Frequency 33 MHz)**

Path Delay Requirement : 30.303ns(33 MHz)

Source	Destination	FMax (MHz)	Delay (ns)	Slack (ns)	Violation
RESET(0)/fb	\UART 1:BUART:reset reg\/main 0	63.800	15.674	14.629	

- **Clock To Output Section**
  - **UART_1_IntClock**

Source	Destination	Delay (ns)
\UART 1:BUART:reset reg\/q	TX INT(0) PAD	69.350
\UART 1:BUART:reset reg\/q	RX INT(0) PAD	56.266
\UART 1:BUART:reset reg\/q	RTS N(0) PAD	40.453
Net 4/q	TX(0) PAD	29.383

t_{RX_RXCLK} RX 出力から RX_CLK へのディレータイムは、UART 実装分析に基づいており、1/2 ビット長と等しく、RX_DATA 信号の中央にいくためには 1 クロック遅れています。

$$t_{RX_RXCLK} = t_{CLOCK} * \left(\frac{\text{Oversampling}}{2} + 1 \right)$$

t_{RX_RXINT} RX_INTERRUPT 信号は、ストップビットが RX_CLK で受信さらえた場合に生成されます。

t_{RX_RXDATA} RX 信号は RX_DATA 出力上で **f_{CLOCK}** にさらに同期し、これらの信号間には 1 クロックのディレーがあります。

t_{RXCLK_RTS} 最後の「RX_CLK raise」から「RTS_N active」へのディレー 4-byte FIFO がフルの場合に発生します。入力 FIFO がフルになるとすぐ、ハードウェアは RTS_N 信号を自動的に設定します。FIFO は、最後の RX_CLK 立ち上がりエッジから 1 コンポーネントクロックサイクル遅れて読み込まれます。

t_{RTS_RX} 「RTS_N Inactive」と「RX data」とのディレー時間は以下に等しい:

$$t_{RTS_RX} = t_{PD_RTS} + RTS_{PD_PCB} + t_{CTS_TX(transmitter)} + RX_{PD_PCB} + t_{S_RX}]$$

以下の場合:

t_{PD_RTS} が RTS_N からピンへのパスディレー。以下に示すように、STA 結果の「Clock To Output(クロックから出力へ)」セクション中に表示されます。

- **Clock To Output Section**

  - **UART_1_IntClock**

Source	Destination	Delay (ns)
\UART 1:BUART:rx state stop1 reg\/q	RX INT(0) PAD	39.902
\UART 1:BUART:sTX:TxShifter:u0\/f0 blk stat comb	TX INT(0) PAD	37.275
\UART 1:BUART:sRX:RxShifter:u0\/f0 blk stat comb	RTS N(0) PAD	27.550
Net 16/q	TX EN(0) PAD	24.813
Net 21/q	TX CLK(0) PAD	24.799
Net 4/q	TX(0) PAD	24.625
Net 20/q	TX DATA(0) PAD	23.648
Net 22/q	RX DATA(0) PAD	23.186
Net 23/q	RX CLK(0) PAD	22.961

RTS_{PD_PCB} は、レシーバーコンポーネントの RTS_N ピンからトランスミッターデバイスの CTS_N ピンまでの PCB パスディレー。

$t_{CTS_TX(transmitter)}$ 「Transmitter datasheet(トランスミッター データシート)」から参照します。

RX_{PD_PCB} はトランスミッターデバイスの TX ピンからレシーバーコンポーネントの RX ピンまでの PCB パスディレー。

t_{S_RX} は RX signal のパスディレー時間。以下に示すように、STA 結果の「Register to Register(レジスタからカラレジスタへ)」セクション中に表示されます。

- **Register to Register Section**

  - **Setup Subsection**

    - **Source Clock : BUS_CLK : Positive edge(Required Frequency 33 MHz)**

      - **Destination Clock : UART_1_IntClock : Positive edge(Required Frequency 16.5 MHz)**

Path Delay Requirement : 30.303ns(33 MHz)

Source	Destination	FMax (MHz)	Delay (ns)	Slack (ns)	Violation
RX(0)/fb	\UART 1:BUART:rx load fifo\/main 11	39.584	25.263	5.040	
RX(0)/fb	\UART 1:BUART:rx state 2\/main 1	40.780	24.522	5.781	
RX(0)/fb	\UART 1:BUART:rx markspace pre\/main 4	41.750	23.952	6.351	
RX(0)/fb	\UART 1:BUART:rx state 3\/main 7	43.303	23.093	7.210	
RX(0)/fb	\UART 1:BUART:rx state 2\/main 2	44.377	22.534	7.769	
RX(0)/fb	\UART 1:BUART:rx state 2\/main 0	45.652	21.905	8.398	
RX(0)/fb	\UART 1:BUART:rx load fifo\/main 10	46.955	21.297	9.006	
RX(0)/fb	\UART 1:BUART:rx break detect\/main 0	54.702	18.281	12.022	
RX(0)/fb	\UART 1:BUART:rx last\/main 0	54.702	18.281	12.022	
RX(0)/fb	\UART 1:BUART:rx markspace pre\/main 0	54.702	18.281	12.022	

コンポーネントの更新履歴

ここでは、前のバージョンからコンポーネントに加えられた主な変更を示します。

バージョン	変更内容	変更理由 / 影響
2.10	UART_PutString() API のパラメータータイプを uint8* から char* へ変更。	この API の共通使用方法は、組み込みストリングと一緒にパラメーターとして使用します。UART_PutString(“Hello World”). “char“ タイプはコンパイラ警告なしにこの用法に用いられるべきです。
	UART_ClearRxBuffer()/ UART_ClearTxBuffer() APIs は ハードウェア FIFO も消去します。	受信/送信途中のデータがないことを保証するため、ハードウェア FIFO は消去する必要があります。
	UART_SendBreak() API に関するタイプミスを修正。UART_WAIT_FOR_COMLETE_REINT を UART_WAIT_FOR_COMPLETE_REINT に変更。	タイプミス修正。
	全 UART API が .cyre file に含まれている場合は CYREENTRANT キーワードを追加。	全ての API が完全にリエントラントではありません。コンポーネント API ソースファイルに含まれているコメントは、どの関数が候補であるかを示しています。 この変更は、安全な方法で使用されている、リエントラントではない関数へのコンパイラ警告を削除するために必要です: 並列呼び出しから、フラグや「Critical Sections」を使って保護します。
	アドレスモードの機能性を更新。	アドレスされていないパケットを自動的にスキップするように、これらのモードをアップグレードしました。
	内部 RX バッファが使用されている場合は、固定ハードウェアフロー制御モード。内部バッファがオーバーランする場合は、RX ISR 内のコードは FIFO からのデータの読みだしを停止します。その結果、RTS 信号はトランスミッター UART を保持します。	データはハードウェア FIFO から読みだされ、s/w バッファがオーバーランしたかどうかに関わらず、s/w バッファに移動します。
	内部クロックコンポーネントを cy_clock_v1_60 に更新。	Clock v1_60 が直近のコンポーネントのバージョンです。
	RX と TX のバッファサイズを最小値 4 に制限。	UART は常に 4 バイト FIFO をバッファとして使用します。
	データシートのマイナーな編集と更新	
2.0.a	データシートのマイナーな編集と更新	
2.0	tx_en 出力が記録されました	配置と信号間のディレーによっては、出力のいかなる組み合わせも誤作動である場合があります。 グリッジを除去するため、出力はレジスタ出力にすべきです。
	リセット入力をレジスタ入力化。	レジスタ入力はリセット入力が使用される場合の最大ボーレートを改善します。
	データシートに特性データを追加	

バージョン	変更内容	変更理由 / 影響
	データシートのマイナーな編集と更新	
1.50	Sleep/Wakeup (スリープ/ウェイクアップ) と Init/Enable (初期化/有効化) API を追加。	ローパワーモードをサポートし、初期化と有効化の制御を分離するほとんどのコンポーネントとの共通インターフェースを提供するため。
	ブレイク信号は長さの選択(11~14)と、「SendBreak」関数に追加されたパラメーターがあります。	UART のブレイク信号長は特定されていないため、11 ~ 14 ビットのセレクションを追加しました。
	16x オーバーサンプリングモードの追加。	16x オーバーサンプリングモードは高速でのエラーに対するジッタの影響を減らします。
	ソフトウェアオプションを **Parity Type** 選択から削除し、代わりに**API control enabled** チェックボックスを追加した。	これにより、パリティ API 制御が必要な場合初期設定値を選択する方法を可能にしました。 このオプションを選択して UART コンポーネントを 1.20 バージョンから更新する場合は、バージョン 1.50 ではパリティオプション「None」を選択するように推奨します。

Copyright © 2005-2012 Cypress Semiconductor Corporation 本文書に記載される情報は、予告なく変更される場合があります。Cypress Semiconductor Corporationは、サイプレス製品に組み込まれた回路以外のいかなる回路を使用することに対しても一切の責任を負いません。特許又はその他の権限下で、ライセンスを譲渡又は暗示することもありません。サイプレス製品は、サイプレスとの書面による合意に基づくものでない限り、医療、生命維持、救命、重要な管理、又は安全の用途のために仕様することを保証するものではなく、また使用することを意図したものでもありません。さらにサイプレスは、誤動作や故障によって使用者に重大な傷害をもたらすことを合理的に予想される、生命維持システムの重要なコンポーネンツとしてサイプレス製品を使用することを許可していません。生命維持システムの用途にサイプレス製品を供することは、製造者がそのような使用におけるあらゆるリスクを負うことを意味し、その結果サイプレスはあらゆる責任を免除されることを意味します。
PSoC Designer™及びProgrammable System-on-Chip™は、Cypress Semiconductor Corp.の商標、PSoC®は同社の登録商標です。本文書で言及するその他全ての商標又は登録商標は各社の所有物です。
全てのソースコード(ソフトウェア及び/又はファームウェア)はCypress Semiconductor Corporation (以下「サイプレス」)が所有し、全世界(米国及びその他の国)の特許権保護、米国の著作権法並びに国際協定の条項により保護され、かつそれらに従います。サイプレスが本書面によるライセンシーに付与するライセンスは、個人的、非独占的かつ譲渡不能のライセンスであって、適用される契約で指定されたサイプレスの集積回路と併用されるライセンシーの製品のみをサポートするカスタムソフトウェア及び/又はカスタムファームウェアを作成する目的に限って、サイプレスのソースコードの派生著作物を複製、使用、変更、そして作成するためのライセンス、並びにサイプレスのソースコード及び派生著作物をコンパイルするためのライセンスです。上記で指定された場合を除き、サイプレスの書面による明示的な許可なくして本ソースコードを複製、変更、変換、コンパイル、又は表示することは全て禁止されます。
免責条項:サイプレスは、明示的又は黙示的を問わず、本資料に関するいかなる種類の保証も行いません。これには、商品性又は特定目的への適合性の黙示的な保証が含まれますが、これに限定されません。サイプレスは、本文書に記載される資料に対して今後予告なく変更を加える権利を留保します。サイプレスは、本文書に記載されるいかなる製品又は回路を適用又は使用したことによって生ずるいかなる責任も負いません。サイプレスは、誤動作や故障によって使用者に重大な傷害をもたらすことが合理的に予想される生命維持システムの重要なコンポーネンツとしてサイプレス製品を使用することを許可していません。生命維持システムの用途にサイプレス製品を供することは、製造者がそのような使用におけるあらゆるリスクを負うことを意味し、その結果サイプレスはあらゆる責任を免除されることを意味します。
ソフトウェアの使用は、適用されるサイプレスソフトウェアライセンス契約によって制限され、かつ制約される場合があります。