Vol 54, No 4—December 2020

ANALOG DEVICES
想像を超える可能性を
AHEAD OF WHAT'S POSSIBLE™

AnalogDialogue

UART——多様な非同期通信に対応可能なハードウェア通信プロトコル

著者：Eric Peña、シニア・ファームウェア・エンジニア
Mary Grace Legaspi、ファームウェア・エンジニア

概要

UART(Universal Asynchronous Receiver/Transmitter) は、デバイス間の通信に広く使われているプロトコルです。本稿では、標準的な手順に従い、UARTをハードウェア通信プロトコルとして使用する方法について説明します。

UARTについては、通信の仕様を適切に定めて適切な構成を行うことが重要です。そうすれば、シリアル・データをやり取りする様々な通信プロトコルに対応できます。シリアル通信では、1本の信号ラインまたはワイヤを使用して、ビット単位でデータを伝送します。双方向の通信を実現したい場合には、送信、受信用に各1本（計2本）のワイヤを使用します。アプリケーションやシステムの要件にもよりますが、シリアル通信では必要な回路やワイヤの規模／数を抑えられるので、実装コストも低減できます。

はじめに

本稿では、UARTを使用する際の基本的な原則について説明します。なかでも、コードを開発する際に理解しておく必要があるパケット伝送、標準的なフレーム・プロトコル、カスタマイズされたフレーム・プロトコルに着目することにします。特に、カスタマイズされたフレーム・プロトコルに対応する機能を実装すれば、よりセキュアな状態を維持することが可能になります。

また、本稿では、システム開発において、使用するICのデータシートを確認する際の基本的な手順について説明します。UARTの規格に対する理解を深めて、それに適切に準拠することが重要です。本稿では、新たな製品／システムを開発する際、UARTの機能や特徴を最大限に生かせるようにすることを目標とします。

「コミュニケーションにおける唯一にして最大の問題は、それが達成されたという幻想を抱いてしまうことである」-- George Bernard Shaw

通信プロトコルは、デバイス間の通信を体系化する上で大きな役割を担います。システムの要件に応じ、通信プロトコルは様々な形で設計されます。どのようなプロトコルが定められるにしても、適切な通信を実現するためにデバイス間で共有される具体的な規則が取り決められることになります。

UARTは、デバイス間（機能ブロック間）に適用するハードウェア通信プロトコルとして広く使用されています。例えば、コンピュータ機器、組み込みシステム、あるいはマイクロコントローラにおいて活用されています。UARTは、送信と受信に計2本のワイヤしか使用しないという点で、他の通信プロトコルとは大きく異なります。

UARTはハードウェア通信プロトコルとして広く採用されていますが、常に最適な形で使われているというわけではありません。例えば、マイクロコントローラ製品の多くはUARTモジュールを内蔵しています。しかし、それを使用する際、フレーム・プロトコルが適切に実装されているか否かということについて注意が払われないケースは少なくありません。

UARTは、速度を設定可能な非同期シリアル通信を使用するハードウェア通信プロトコルだと定義できます。非同期というのは、送信デバイスから受信デバイスに送られるビット・データの同期をとるためのクロック信号が存在しないということを意味します。

インターフェース

図1をご覧ください。各UARTデバイスにおいて、2つの信号をやり取りする機能はそれぞれ次のように呼ばれます。

- トランスミッタ（Tx）
- レシーバー（Rx）

各デバイスのトランスミッタ・ライン、レシーバー・ラインの役割は、シリアル・データの送信／受信を行うことです。それにより、シリアル通信が実現されます。

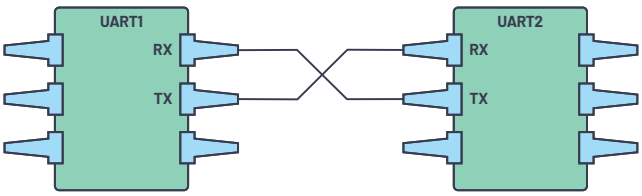

図1. 相互に直接通信を行う2つのUARTデバイス

図2のように、送信側のUARTデバイスには、パラレル形式でデータを送信するデータ・バスが接続されます。送信側のUARTデバイスからのデータは、伝送ライン（ワイヤ）を介し、受信側のUARTデバイスにシリアル形式でビットごとに送信されます。受信側のUARTデバイスは、シリアル・データをパラレル形式に変換して受信側のデータ・バスに引き渡します。

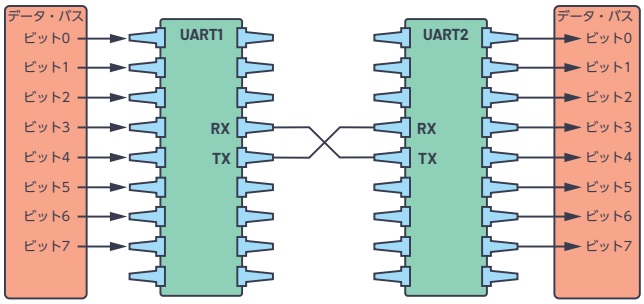

図2. UARTデバイスとデータ・バスの関係

UARTの信号ラインは、2つのUARTデバイスの間でデータを送受信するための媒体として機能します。UARTデバイスには、送信専用または受信専用のピンが設けられる点に注意してください。

UARTをはじめとするほとんどのシリアル通信では、送信側、受信側の両デバイスにおいてボー・レートを同じ値に設定する必要があります。ここで、ボー・レートとは通信チャンネルに情報が転送される速度のことです。シリアル・ポートでは、1秒あたりの最大転送ビット数がボー・レートに相当します。

UARTについて知っておくべき情報を表1にまとめました。

表1. UARTの概要

ワイヤの数	2
速度	9600, 19200, 38400, 57600, 115200, 230400, 460800, 921600, 1000000, 1500000
伝送方法	非同期
マスタの最大数	1
スレーブの最大数	1

UARTのインターフェースでは、クロック信号を使用して送信デバイスと受信デバイスの同期をとるということは行われません。つまり、データは非同期で送受信されます。送信デバイスは、独自のクロック信号に基づいてビットストリームを生成します。一方の受信デバイスは、独自のクロック信号を使用して受信データのサンプリングを行います。両方のデバイスのボー・レートを同一に設定することにより、同期ポイントが管理されることになります。仮にボー・レートが同一でなかったとしたら、データの送受信のタイミングに影響が及び、データの処理において問題が生じるおそれがあります。ボー・レートの許容誤差は最大10%です。それを超えると、ビット・データの送受信のタイミングがずれて正しく処理を実行できなくなります。

データ伝送

UARTでは、パケット方式でデータの伝送が行われます。送信デバイスと受信デバイスの間では、シリアル・パケットの構成や、物理的なハードウェア・ラインの制御といった処理を行う必要があります。パケットは、開始ビット、データ・フレーム、パリティ・ビット、停止ビットで構成されます（図3）。

図3. UARTのパケットの例

開始ビット（Start Bit）

通常、UARTのデータ伝送ラインは、データを伝送していないときには電圧レベルがハイの状態で維持されます。データの転送を開始する際には、送信側のUARTデバイスは伝送ラインの電圧を1クロック・サイクルの間、ハイからローに引き下げます。一方、受信側のUARTデバイスは電圧がハイからローに遷移したことを検出すると、ボー・レートの周波数でデータ・フレームのビットの読み出しを開始します（図4）。

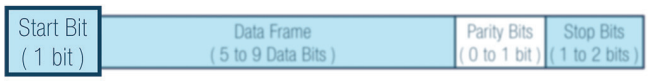

図4. 開始ビット

データ・フレーム（Data Frame）

データ・フレームには、転送の対象となる本来のデータが含まれています（図5）。パリティ・ビットを使用する場合、データ・フレームの長さは5～8ビットです。パリティ・ビットを使用しない場合には9ビットまで拡張できます。ほとんどの場合、データは最下位ビットから順に送信されます（LSBファースト）。

図5. データ・フレーム

パリティ・ビット（Parity Bits）

パリティ・ビットは、受信側のUARTデバイスが、伝送中にデータが変化していない（エラーが発生していない）ことを確認するための手段です（図6）。

ビット・データは、電磁放射、ボー・レートの不整合、長距離にわたるデータ伝送によって壊れてしまう可能性があります。受信側のUARTデバイスは、データ・フレームを読み取った後、値が1のビットの数をカウントし、その総数が偶数であるか奇数であるかを確認します。パリティ・ビットの値が0（偶数パリティ）である場合、データ・フレームにおいて値が1（ハイ）のビットの総数は偶数でなければなりません。また、パリティ・ビットの値が1（奇数パリティ）である場合には、データ・フレームにおいて値が1のビットの総数は奇数でなければなりません。

パリティ・ビットがデータの内容とマッチしていれば、伝送時にエラーは発生しなかったと見なされます。一方、パリティ・ビットの値が0なのに総数が奇数である場合、またはパリティ・ビットの値が1なのに総数が偶数である場合には、データ・フレーム内のいずれかのビットが何らかの原因で反転してしまったと判定されます。

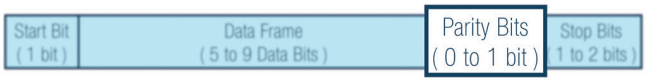

図6. パリティ・ビット

停止ビット（Stop Bits）

パケットの終わりを知らせるためのビットです（図7）。送信側のUARTデバイスは、パケットの終わりを知らせるために、1～2ビット分の間、データ伝送ラインの電圧をローからハイに引き上げます。

図7. 停止ビット

UARTにおけるデータの伝送手順

UARTにおけるデータの伝送手順は、5つのステップから成ります。以下、各ステップについて順に説明します。

【ステップ1】

送信側のUARTデバイスは、データ・バスからパラレル形式のデータを受信します（図8）。これらのデータがデータ・フレームに相当します。

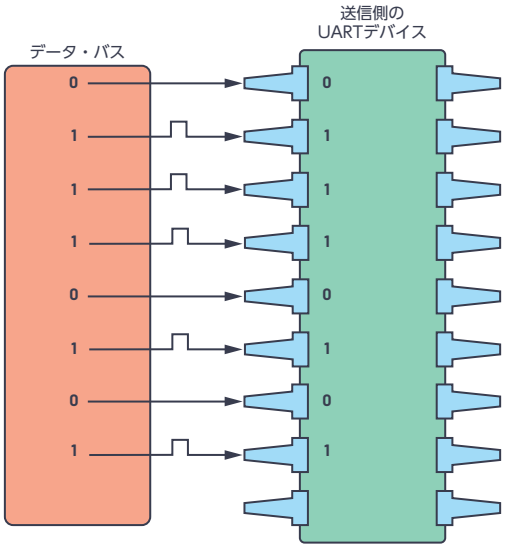

図8. データ・バスから送信側
UARTデバイスへのデータの送信

【ステップ2】

送信側のUARTデバイスは、開始ビット、パリティ・ビット、停止ビットをデータ・フレームに追加します（図9）。

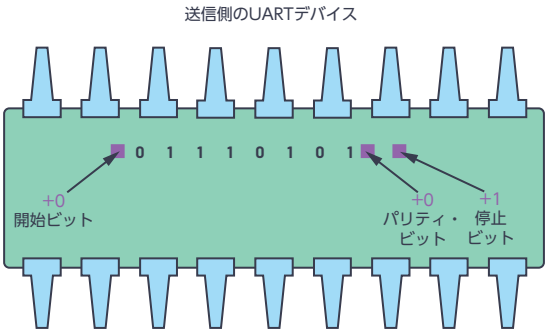

図9. Tx側におけるパケットの構成

【ステップ3】

パケット全体が、送信側のUARTデバイスから受信側のUARTデバイスに送信されます。その際、パケットは開始ビットに始まり停止ビットで終わるシリアル形式で扱われます。受信側のUARTデバイスは、事前に設定されたボー・レートで信号ラインのデータのサンプリングを実行します（図10）。

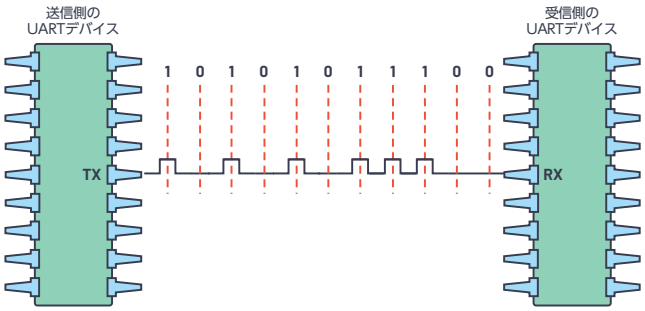

図10. パケットの伝送

【ステップ4】
受信側のUARTデバイスは、パケットから開始ビット、パリティ・ビット、停止ビットを除去し、データ・フレームだけを残します（図11）。

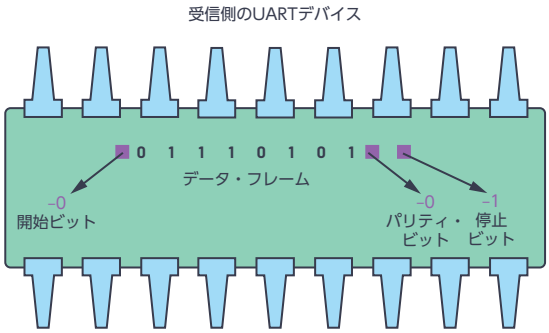

図11. Rx側におけるパケットの処理

【ステップ5】
受信側のUARTデバイスは、シリアル形式のデータをパラレル形式のデータに変換します。変換後のデータを、受信側のデータ・バスに転送します（図12）。

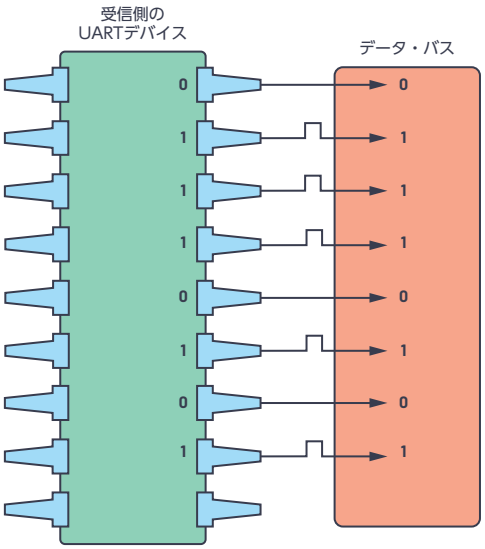

図12. 受信側のUARTデバイスからデータ・バスへのデータの送信

フレーム・プロトコル

UARTで提供されていながら最大限に活用されていない主要な機能の1つに、フレーム・プロトコルがあります。フレーム・プロトコルの主な目的は、各デバイスのセキュリティ／保護を強化することです。したがって、この機能はぜひ活用すべきです。

例えば、2つのデバイスが同じフレーム・プロトコルを使用しているとします。ここで、構成を確認することなく同じUARTデバイスに対して接続を行うと、デバイスが異なるピンに接続されてしまい、システムが誤動作を起こしてしまうことがあります。

それに対し、フレーム・プロトコルの機能を実装すると、設計したフレーム・プロトコルに合致する形で情報が受信されていることを、パースを実行して確認することが義務づけられます。そのため、よりセキュアな状態が確保されます。各フレーム・プロトコルは、固有かつセキュアなものになるよう特別に設計されます。

フレーム・プロトコルを設計する際には、CRC（Cyclic Redundancy Check）値を含む所望のヘッダとトレーラを設定することができます。図13では、ヘッダとして2バイトのデータを設定しています。

図13. フレーム・プロトコルの例

この例を参考にすれば、デバイスに固有のヘッダ、トレーラ、CRC値を設定することができるでしょう。

ヘッダ1（H1は0xAB）とヘッダ2（H2は0xCD）

ヘッダは、適切なデバイスと通信しているかどうかを判別するための一意的な識別子です。

コマンド（CMD）の選択

コマンドは、2つのデバイスの間で通信を行うために設計されたコマンド・リストの中から選択します。

コマンドあたりのデータ長（DL）

データ長は、選択するコマンドに依存します。選択したコマンドによっては、データ長を長くとることができます。長さはまちまちですが、調整も行えるようになっています。

データn（可変データ）

デバイスから転送されるペイロード（データの本体）です。

トレーラ1（T1は0xE1）とトレーラ2（T2は0xE2）

トレーラのデータは末尾に付加されます。ヘッダと同様に、一意的な識別子として機能します。

CRC値

CRC値は、未処理のデータに生じる偶発的な誤りを検出するために付加されます。送信デバイスに付加されたCRC値と受信デバイスで計算したCRC値は必ず一致する必要があります。

各UARTデバイスにフレーム・プロトコルの機能を実装し、セキュリティを強化することをお勧めします。なお、フレーム・プロトコルは、送信デバイスと受信デバイスで同じ構成にする必要があります。

UARTの動作

ハードウェア通信プロトコルを使用する際には、必ず事前にデータシートとハードウェア・リファレンス・マニュアルを確認してください。確認の手順は7つのステップから成ります。以下、それぞれについて順に説明します。

【ステップ1】

データシートで、デバイスのインターフェースを確認します（図14）。

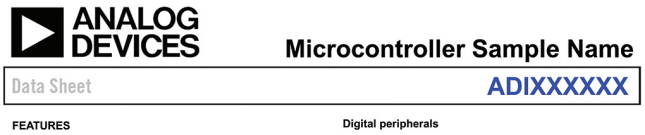

図14. マイクロコントローラのデータシートの例

【ステップ2】

メモリ・マップで、UARTに関するアドレスを確認します（図15）。

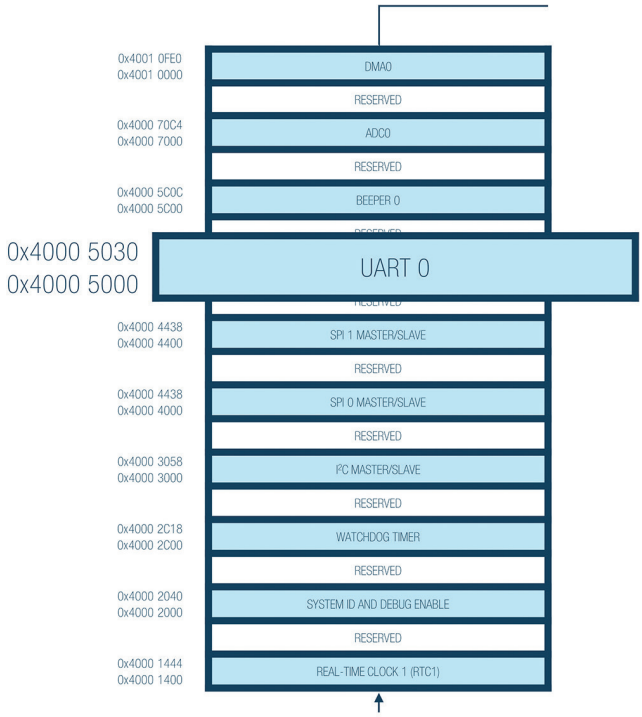

図15. マイクロコントローラのメモリ・マップ

【ステップ3】

動作モード、データのビット長、パリティ・ビット、停止ビットなど、UARTのポートに関する具体的な情報について確認します。データシートには、以下のようなことが書かれているはずです。

UARTのポート

サンプルのマイクロコントローラは、PCの標準的なUARTに対して完全な互換性を持つ全二重UARTポートを搭載しています。このポートは、他のペリフェラルやホストに対する簡素化されたUARTインターフェースとして機能します。サポートするのは、全二重通信、DMA転送、シリアル・データの非同期転送です。また、5～8個のデータ・ビットと、パリティなし／偶数パリティ／奇数パリティをサポートします。フレームは、1.5個または2個の停止ビットで終了します。

【ステップ4】

ボー・レートの計算方法など、UARTの動作に関する詳細情報を確認します。ボー・レートは計算式に基づいて設定します。どのような計算式を使うのかは、マイクロコントローラ製品ごとに異なります。詳細情報の例を以下に示します。

- データ・ビット：5～8ビット
- 停止ビット：1ビット、2ビット、1 1/2ビット
- パリティ・ビット：なし、偶数、奇数
- オーバーサンプリング比：4/8/16/32倍にプログラムすることが可能
- ボー・レート：$PCLK/((M + N/2048) \times 2^{OSR + 2} \times DIV$

最後に示したボー・レートの計算式で使われている変数の意味は以下のとおりです。

PCLK：ペリフェラル・クロック

OSR：オーバーサンプリング比（UART_LCR2.OSR = 0～3）

DIV：ボー・レートの分周比（UART_DIV = 1～65535）

M：フラクショナル・ボー・レートのDIVMの値（UART_FBR.DIVM = 1～3）

N：フラクショナル・ボー・レートのDIVNの値（UART_FBR.DIVN = 0～2047）

【ステップ5】

ボー・レートを決めるPCLKとして、どのクロックを使用するのか確認します。ここで取り上げている例の場合、26MHzまたは16MHzのPCLKを使用できます。OSR、DIV、DIVM、DIVNは製品によって異なることに注意してください。

表2. 26MHzのPCLKを使用する場合のボー・レートの例

ボー・レート	OSR	DIV	DIVM	DIVN
9600	3	24	3	1078
115200	3	4	1	1563

表3. 16MHzのPCLKを使用する場合のボー・レートの例

ボー・レート	OSR	DIV	DIVM	DIVN
9600	3	17	3	1078
115200	3	2	2	348

【ステップ6】
続いて、UARTの構成に使用するレジスタの詳細を確認します。UART_LCR2、UART_DIV、UART_FBRなど、ボー・レートを計算するためのパラメータを確認してください。表4に、確認すべきレジスタについてまとめました。

表4. UARTの構成に使用するレジスタ

名前	説明
UART_DIV	ボー・レートの分周比
UART_FIBR	フラクショナル・ボー・レート
UART_LCR2	セカンド・ライン制御

【ステップ7】
各レジスタについて確認したら、値を代入してボー・レートを計算します。その後、UARTの実装を開始します。

なぜ、UARTについて理解すべきなのか？

UARTの通信プロトコルについて精通していると、高い堅牢性と優れた品質が求められる製品の開発に役立ちます。2本のワイヤだけを使ってデータを送信する方法や、データ全体（ペイロード）を転送する方法を理解しておけば、データを誤りなく送受信することが可能になります。UARTは、非常によく使われるハードウェア通信プロトコルです。そのため、UARTに関する知識は、将来の設計の柔軟性を高めるためにも役立つはずです。

ユースケース

UARTは様々なアプリケーションで利用できます。以下、いくつかの例を挙げておきます。

- デバッグ：システム開発を行っている際には、バグを早期に検出することが重要です。この作業を行っているとき、UARTを追加することでシステムからのメッセージを取得することができます。
- 製造時の機能レベル・トレーシング：製造時には、ログが非常に重要な役割を果たします。ログを基に各種の機能について判定を行うことで、製造ラインで何が起きているのかオペレータに通知することが可能になります。
- 顧客／クライアントに対するアップデート：ソフトウェアのアップデートは非常に重要な作業です。包括的で動的なハードウェアとアップデートが可能なソフトウェアを組み合わせることにより、完全なシステムを構築することできます。
- テスト／検証：出荷前に十分なテスト／検証を行うことで、最高の品質を備える製品を顧客に提供することが可能になります。

参考資料

「Basics of UART Communication（UART通信の基礎）」Electronics Hub、2017年7月

Scott Campbell「Basics of UART Communication（UART通信の基礎）」Circuit Basics

Robert Keim「Back to Basics: The Universal Asynchronous Receiver/Transmitter（UARTの基本をおさらいする）」All About Circuits、2016年12月

「What Is UART Protocol? UART Communication Explained（UARTプロトコルとは何なのか、その基本を学ぶ）」Arrow

著者について

Eric Peña（eric.pena@analog.com）は、アナログ・デバイセズのシニア・ファームウェア・エンジニアです。2019年4月に入社（フィリピンのカビテ支社）しました。現在はコンシューマ・ソフトウェア・エンジニアリング・グループの設計／レイアウト・チームに所属しています。それ以前はTechnology Enabler Designerにファームウェア・エンジニアとして、Fujitsu Ten Solutionsにシステム・エンジニアとして勤務していました。マニラのアダムソン大学でコンピュータ工学の学士号を取得しています。

著者について

Mary Grace Legaspi（mary.legaspi@analog.com）は、アナログ・デバイセズのファームウェア・エンジニアです。2018年9月に入社（フィリピンのカビテ支社）しました。現在はコンシューマ・ソフトウェア・エンジニアリング・グループの設計／レイアウト・チームに所属しつつ、フィリピン大学で経営学の修士号の取得を目指しています。タルラク州立大学で電子工学の学士号を取得しました。

©2020 Analog Devices, Inc. All rights reserved.
本紙記載の商標および登録商標は、各社の所有に属します。
Ahead of What's Possibleはアナログ・デバイセズの商標です。

AD5412-0-12/20