ADUC702X シリアル・ダウンロード・プロトコル

Aude Richard 著

はじめに

MicroConverter®製品ファミリーの重要な機能は、コードを内蔵のフラッシュ/EE プログラム・メモリへイン・サーキットでダウンロードするデバイスの機能です。イン・サーキットでのコードのダウンロードは、デバイスの UART シリアル・ポートを使用して行われるため、一般にシリアル・ダウンロードと呼ばれています。シリアル・ダウンロード機能を使うと、ターゲット・システムに部品を直接ハンダ付けした状態で再書込みできるため、外付けのデバイス書込器が不要になります。また、シリアル・ダウンロード機能を使うと、現場でのシステム・アップグレードも可能になります。必要なものは、MicroConverter に対するシリアル・ポート・アクセスだけです。これは、メーカーがデバイスを交換することなく、現場でシステム・ファームウェアをアップグレードできることを意味します。

すべての MicroConverter デバイスは、パワーオン時または外部リセット信号入力時の特定のピン設定を使って、シリアル・ダウンロード・モードに設定することができます。ADuC702x ファミリーの MicroConverter の場合、抵抗(1 kΩ)を介して P0.0 入力ピンをローレベルに接続します。パワーオン時またはハード・リセット入力時にデバイスがこの状態がを検出すると、デバイスはシリアル・ダウンロード・モードになります。このモードで、内蔵の常駐ローダ・ルーチンが起動されます。

内蔵ローダはデバイスの UART を設定して、特定のシリアル・ダウンロード・プロトコルを使ってホスト・マシンと交信して、データのフラッシュ/EE メモリ・スペースへのダウンロードを管理します。ダウンロードするプログラム・データのフォーマットでは、リトル・エンディアンを使用する必要があります。 シリアル・ダウンロード・モードは、デバイスの標準電源定格内(2.7 V～3.6 V)で動作させる必要があります。特定のプログラミング用高電圧は、チップ内で発生するため不要です。図 1 に、評価ボードでシリアル・ダウンロード・モードを開始させる方法を示します。

また、アナログ・デバイセズは QuickStart™ 開発ツールとして、Windows®プログラム(C:\ADuC702x\Download\ ARMWSD.exe)も提供しています。このプログラムを使うと、ユーザーはコードを PC シリアル・ポート COM1、COM 2、COM 3、または COM 4 から MicroConverter へダウンロードすることができます。ただし、ホスト・マシンがこのアプリケーション・ノートで説明するシリアル・ダウンロード・プロトコルに準拠する限り、任意のマスター・ホスト・マシン(PC、マイクロコントローラ、または DSP)から MicroConverter へダウンロードできることに注意してください。

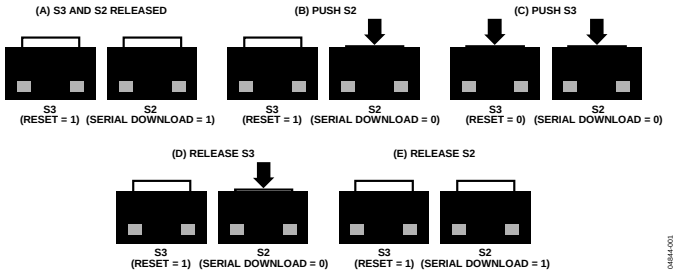

図 1.MicroConverter のシリアル・ダウンロード・モード設定

Rev. B

アナログ・デバイセズ株式会社

本　　社／〒105-6891　東京都港区海岸 1-16-1　ニューピア竹芝サウスタワービル
電話 03（5402）8200
大阪営業所／〒532-0003　大阪府大阪市淀川区宮原 3-5-36　新大阪 MT ビル 2 号
電話 06（6350）6868

このアプリケーション・ノートでは、MicroConverter シリアル・ダウンロード・プロトコルについて説明し、エンド・ユーザーがこのプロトコル(組込み型ホストと組込み型 MicroConverter との間)を理解して、エンド・ターゲット・システムに実装できるようにします。

紛らわしさを避けるため、用語「ホスト」は、データを MicroConverter へダウンロードするホスト・マシン(PC、マイクロコントローラ、または DSP)を意味するものと定義します。用語「ローダ」は、MicroConverter に内蔵のシリアル・ダウンロード・ファームウェアを意味するものと定義します。

MicroConverter ローダの実行

ADuC702x MicroConverter 上のローダは、抵抗 (代表値 1 kΩ のプルダウン)を介してシリアル・ダウンロード P0.0 ピンをロー・レベルに接続してデバイスをリセット (デバイス自体の RESET 入力ピンをトグル、または電源のオン/オフによりデバイスをリセット)することにより起動されます。

物理インターフェース

ローダは起動されると、ホストから同期用のバック・スペース(BS = 0x08)文字が送信されてくるのを待ちます。ローダはこの文字のタイミングを計測して、その結果を使って、MicroConverter の UART シリアル・ポートを、送信または受信、ホストと同じボー・レート、8 データ・ビット、パリティなしに設定します。ボー・レートは、600 bps～115200 bps の範囲である必要があります。ローダはバック・スペースを受信すると、直ちに次に示す 24 バイト ID データ・パケットを送信します。

15 バイト = 製品識別マーク

3 バイト = ハードウェアとファームウェアのバージョン番号

4 バイト = 予約済み

2 バイト = ライン・フィードとキャリッジ・リターン

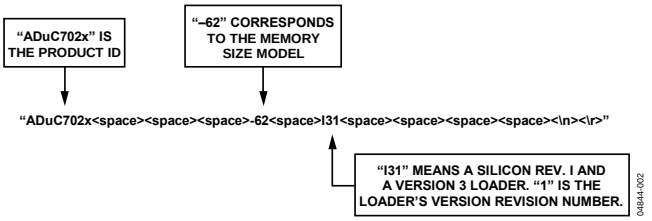

図 2.ID データ・パケット

データ・トランスポート・パケット・フォーマットの定義

UART の設定が終わると、データ転送が開始できます。一般的な通信データ・トランスポート・パケット・フォーマットを表 1 に示します。

パケット開始 ID フィールド

最初のフィールドはパケット開始 ID フィールドで、2 個の開始文字(0x07 と 0x0E)が含まれます。これらのバイトは固定で、ローダが有効なデータ・パケットの開始を検出する際に使います。

データ・バイト数フィールド

次のフィールドは、データ 1 (コマンド機能)を含む合計データ・バイト数です。最小データ・バイト数は 5 で、これはコマンド機能とアドレスに該当します。最大許容データ・バイト数は 255 で、これはコマンド機能、4 バイト・アドレス、250 バイトのデータに該当します。

コマンド機能フィールド(データ 1)

コマンド機能フィールドは、データ・パケットの機能を指定します。5 個の有効コマンド機能の内の 1 つを指定します。5 個のコマンド機能は、5 文字の ASCII 文字(E、W、V、P、または R)の内の 1 文字で指定されます。データ・パケット・コマンド機能の一覧を表 2 に示します。

アドレス・フィールド(データ 2～データ 5)

アドレス・フィールドには、32 ビットのアドレス、h、u、m、l が含まれ、MSB が h ロケーションに、LSB が l ロケーションに配置されます。

データ・バイト・フィールド(データ 6～データ 255)

ユーザー・コードがバイト単位でダウンロードされ検証されます。データ・バイト・フィールドには、最大 250 のデータ・バイトが含まれます。

データは通常 Intel®拡張 16 バイト・レコード・フォーマット(このアプリケーション・ノートの終わりに記載する(Intel 拡張 16 進フォーマット・セクションを参照)から分解されて、ホストによりローダへ送信する前に、表 2.に説明するデータ・パケットの一部として再組立されます。

チェックサム・フィールド

データ・パケットのチェックサムがこのフィールドに書き込まれます。バイト数フィールドの 16 進値と、データ 1～データ 255(存在するだけの数)のフィールドの 16 進値の和から、2 の補数チェックサムが計算されます。チェックサムは、この和の 2 の補数値になります。したがって、データ・バイト数からチェックサムまでのすべてのバイトの和の LSB は 0 である必要があります。これは、数学的に次のように表されます。

$$CS = 0x00 - (\text{データ・バイト数} + \sum_{N=1}^{255} \text{データ・バイト}_N)$$

別の表現では、開始 ID を除くすべてのバイトの 8 ビット和は、00h になります。

コマンドの確認

ローダ・ルーチンは、各データ・パケットに対して BEL (0x07)を否定応答として、または ACK (0x06)を肯定応答として発行します。

ローダが不正なチェックサムまたは無効なアドレスを受信したとき、ローダは BEL を送信します。古いデータ(未消去)の重ね書きで新しいデータをダウンロードしても、ローダは警告を発しません。PC インタフェース側は、コードがダウンロードされるすべてのロケーションが消去されることに注意する必要があります。

表 1. データ・トランスポート・パケットのフォーマット

Start ID		No. of Data Bytes	Data 1 CMD	Data 2 to Data 5 (Address: h, u, m, l)	Data x (x = 6 to 255)	Checksum
0x07	0x0E	5 to 255	E, W, V, P, or R	h, u, m, l	xx	CS

表 2.データ・パケット・コマンドの機能

Command Functions	Command Byte in Data 1 Field	Loader Positive Acknowledge	Loader Negative Acknowledge
Erase Page	E (0x45)	ACK (0x06)	BEL (0x07)
Write	W (0x57)	ACK (0x06)	BEL (0x07)
Verify	V (0x56)	ACK (0x06)	BEL (0x07)
Protect	P (0x50)	ACK (0x06)	BEL (0x07)
Run (Jump to User Code)	R (0x52)	ACK (0x06)	BEL (0x07)

表 3.フラッシュ/EE メモリ消去コマンド

Start ID		No. of Data Bytes	Data 1 CMD	Data 2 to Data 5 (Address: h, u, m, l)	Data 6 (pages)	Checksum
0x07	0x0E	6	E (0x45)	h, u, m, l	x pages (1 to 124)	CS

表 4.フラッシュ/EE メモリ・プログラム・コマンド

Start ID		No. of Data Bytes	Data 1 CMD	Data 2 to Data 5 (Address: h, u, m, l)	Data x (x = 1 to 250)	Checksum
0x07	0x0E	5 + x (6 to 255)	W (0x57)	h, u, m, l	Data bytes	CS

表 5. 検証コマンド、ビット変更

Original Bits	Transmitted Bits	Restored Bits
7	4	7
6	3	6
5	2	5
4	1	4
3	0	3
2	7	2
1	6	1
0	5	0

消去コマンド

消去コマンドを使うと、ユーザはデータ 2～データ 5 で指定される特定のページのフラッシュ/EE を消去することができます。アドレスは、ページの開始に丸め処理されます。このコマンドには消去するページ数も含まれます。アドレスが 0x00000000 で、かつページ数が 0x00 の場合、ローダはこのコマンドをマス消去コマンドと解釈して、ユーザー・コード・スペース全体とフラッシュ/EE 保護を消去します。消去コマンドのデータ・パケットを表 3 に示します。

書込みコマンド

書込みコマンドには、データ・バイト数(5 + x)、コマンド、書き込む先頭データ・バイトのアドレス、書き込むデータ・バイトが含まれます。バイトは、フラッシュ/EE に書き込まれます。チェックサムが不正または受信したアドレスが範囲外の場合、ローダは BEL を送信します。ホストがローダから BEL を受信すると、ダウンロード・プロセスが停止されて、ダウンロード・シーケンス全体が再起動されます。

検証コマンド

検証コマンドは、表 5 に示すようにほとんど書込みコマンドと同じです。コマンド・フィールドは V (0x56)ですが、エラー検出のチャンスを増やすために、データ・バイトが変更されています。ローレベルの 5 ビットがハイ側へ 5 ビット・シフトされ、ハイレベルの 3 ビットがロー側へ 3 ビット・シフトされます。

ローダが正しいビット・シーケンスに戻した後に、その値をフラッシュ内の値と比較します。比較結果が正しく、かつチェックサムが正しい場合は、ACK (0x06)が返されます。その他の場合には、BEL (0x07)が返されます。

フラッシュ/EE メモリ保護コマンド

このコマンドを使うとき、次の 3 ステップ・シーケンスに従います。

1. コマンドを開始します。タイプは 0x00 である必要があり、「huml」は任意の値にします。
2. 保護対象ページ・グループのアドレスを送信します。各ページ・グループに対してこのステップを繰り返します。タイプを 0x0F にします。
3. 「huml」内でキーを送信し、タイプを 0x01 にします。FEEADR に値「hu」を、FEEDAT に値「ml」を、それぞれ設定します。キーが必要ない場合は、「huml」を 0xFFFFFFFF にします。

たとえば、ページ 0～ページ 7 を書込みから保護する場合は、読み出し保護を設定して、キー0x12345678 を使います。次のコマンドを送信します。

1. シーケンス開始:
   0x07 0x0E 0x06 0x50 0xXXXXXXXX 0x00 CS
2. 保護:
   0x07 0x0E 0x06 0x50 0x00000000 0x0F CS
   (ページ 0～ページ 3)
   0x07 0x0E 0x06 0x50 0x00000200 0x0F CS
   (ページ 4～ページ 7)
   0x07 0x0E 0x06 0x50 0x0000F800 0x0F CS
   (読み出し保護)
3. キーおよびシーケンス終了:
   0x07 0x0E 0x06 0x50 0x12345678 0x01 CS

保護コマンドはローダのレビジョン 0 以後のバージョンのみで使用可能なことに注意してください。レビジョン 0 では、FEEADR = ml かつ FEEDAT = ml です。ローダのそれより後バージョンでは、FEEADR = hu です。

このプロトコルでは、フラッシュ/EE メモリを非保護にすることはできません。保護を解除するときは、マス消去コマンドを使います。

リモート実行コマンド

ホストがすべてのデータ・パケットをローダへ送信した後、ホストは最後のパケットを送信して、ローダにコードの実行を開始させることができます。

2 種類のリモート実行があります。

- ソフトウェア・リセット(h、u、m、l = 0x1)
- ユーザー・コードへのジャンプ(h、u、m、l = 0x0)

表 8 に、リモート実行またはリセットの例を示します。ソフトウェア・リセットによりすべてのペリフェラルがリセットされるため、ソフトウェア・リセットの実行が推奨されます。

表 6.フラッシュ/EE メモリ検証コマンド

Start ID		No. of Data Bytes	Data 1 CMD	Data 2 to Data 5 (Address: h, u, m, l)	Data x (x = 1 to 250)	Checksum
0x07	0x0E	5 + x (6 to 255)	V (0x56)	h, u, m, l	Modified data bytes	CS

表 7.フラッシュ/EE メモリ保護コマンド

Start ID		No. of Data Bytes	Data 1 CMD	Data 2 to Data 5 (Address: h, u, m, l)	Data 6	Checksum
0x07	0x0E	0x06	P (0x50)	h, u, m, l	Type	CS

表 8.リモート実行コマンド

Packet ID		No. of Data Bytes	Data 1 CMD	Data 2 to Data 5 (Address: h, u, m, l)	Checksum
0x07	0x0E	0x05	R (0x52)	h, u, m, l = 0x1	0xA8

Intel 拡張 16 進フォーマット

Intel 拡張 16 進フォーマットは、表示可能 ASCII または印字可能フォーマットのマシン言語を保存する規格です。Hex 8 フォーマットに似ていますが、Intel 拡張リニア・アドレス・レコードは、データ・アドレスの上位 16 ビットも設定する出力である点が異なります。各データ・レコードはコロンで始まり、その後ろに 8 文字のプレフィックスが続き、2 文字のチェックサムで終わります。各レコードのフォーマットは次の通りです。

:BBAAAATTHHHH....HHHCC

ここで、

BB は、2 桁の 16 進バイト・カウントで、行に表示されるデータ・バイト数を表します。

AAAA は、4 桁の 16 進アドレスで、データ・レコードの開始アドレスを表します。

TT は、次に示す 2 桁のレコード・タイプです。

00–データ・レコード

01–End of file レコード

02–拡張セグメント・アドレス・レコード

03–開始セグメント・アドレス・レコード

04–拡張リニア・アドレス・レコード

05–開始リニア・アドレス・レコード

HH は、2 桁の 16 進データ・バイトです。

CC は、2 桁の 16 進チェックサムで、レコード内のすべての先行バイト(プレフィックスを含む)の和の 2 の補数です(すべてのバイト和+チェックサム= 00)。

レコード・タイプ

データ・レコード

レコード・タイプ 00 (データ・レコード)は、ファイルのデータを含むレコードです。データ・レコードはコロンの開始文字(:)で始まり、その後ろにバイト・カウント、先頭バイトのアドレス、レコード・タイプ(00)が続きます。データ・バイトは、レコード・タイプの後ろに続きます。チェックサムがデータ・バイトの後ろに続き、レコード内の先行バイト(開始文字を除く)の 2 の補数です。データ・レコードの例を次に示します(スペースは分かり安くするために挿入してあるので、実際のオブジェクト・ファイルからは除く必要があります):

:10 0000 00 FFFEFDFCFBFAF9F8F7F6F5F4F3F2F1F0 FF

:05 0010 00 0102030405 AA

エンド・レコード

レコード・タイプ 01 (エンド・レコード)は、データ・ファイルの終わりを示します。エンド・レコードはコロン開始文字(:)で始まり、その後ろにバイト・カウント(00)、アドレス(0000)、レコード・タイプ(01)、チェックサム(FF)が続きます。たとえば、

:00 0000 01 FF

拡張セグメント・アドレス・レコード

レコード・タイプ 02 (拡張セグメント・アドレス・レコード)は、セグメント・ベース・アドレスのビット 4～ビット 19 を指定します。オブジェクト・ファイル内の任意の場所に登場することができ、それが変更されるまで、ファイル内のすべての後続データ・レコードの絶対メモリ・アドレスに影響を与えます。拡張セグメント・アドレス・レコードはコロン開始文字 (:)で始まり、その後ろにバイト・カウント (02)、アドレス (0000)、レコード・タイプ (02)、セグメント・ベース・アドレスのビット 4～ビット 19 で表した 4 文字の 16 進値、2 文字のチェックサムが続きます。たとえば、

:02 0000 02 1000 55

開始セグメント・アドレス・レコード

レコード・タイプ 03 (開始セグメント・アドレス・レコード)は、オブジェクト・ファイルの実行開始セグメント・ベース・アドレスのビット 4～ビット 19 を指定します。たとえば、

:02 0000 03 0000 55

拡張リニア・アドレス・レコード

レコード・タイプ 04 (拡張リニア・アドレス・レコード)は、ディステネーション・アドレスのビット 16～ビット 31 です。オブジェクト・ファイル内の任意の場所に登場することができ、変更されるまで、ファイル内のすべての後続データ・レコードの絶対メモリ・アドレスに影響を与えます。拡張リニア・アドレス・レコードはコロン開始文字 (:)で始まり、その後ろにバイト・カウント (02)、アドレス (0000)、レコード・タイプ (04)、ディステネーション・アドレスのビット 16～ビット 31 で表した 4 文字の 16 進値、2 文字のチェックサムが続きます。たとえば、

:02 0000 04 FFFF 55

開始リニア・アドレス・レコード

レコード・タイプ 05 (開始リニア・アドレス・レコード)は、オブジェクト・ファイルの実行開始アドレスのビット 16～ビット 31 を指定します。たとえば、

:02 0000 05 0000 55

簡単な Intel Hex オブジェクト・ファイル

拡張リニア・アドレス、拡張セグメント・アドレス、データ、エンドの各レコードを含む Intel Hex オブジェクト・ファイルの例を次に示します。

:020000040108EA

:0200000212FFBD

:0401000090FFAA5502

:00000001FF

1. データ・レコードの拡張リニア・アドレス・オフセット(この例では 0108)。

   :02 0000 04 0108 EA

2. データ・レコードの拡張セグメント・アドレスを求めます(この例では 12FF)。

   :02 0000 02 12FF BD

3. データ・レコード内のデータのアドレス・オフセットを求めます(この例では 0100)。

   :04 0100 00 90FFAA55 02

4. データ・レコードの先頭バイトの絶対アドレスを計算します。

   + 0108 0000 リニア・アドレス・オフセット、16 ビット左シフト
   + 0001 2FF0 セグメント・アドレス・オフセット、4 ビット左シフト
   + 0000 0100 データ・レコードからのアドレス・オフセット
   = 0109 30F0 先頭データ・バイトの 32 ビット・アドレス

5. 計算:

   010930F0 90
   010930F1 FF
   010930F2 AA
   010930F3 55

制限

レコード・タイプ 02、タイプ 03、タイプ 04、タイプ 05 は実装されていません。サポートされていないレコードは無視されます。内部フラッシュをアクセスするときは、アドレスの下位 16 ビットのみを使います。そのため、上位 16 ビットが変化するレコード・タイプを無視するのが安全です。

AN04844-0-4/06(B)-J