AN-1160
アプリケーション・ノート

Cortex-M3 ベースの ADuCxxx の シリアル・ダウンロード・プロトコル

はじめに

Cortex-M3 ベースの ADuCxxx の重要な機能は、内蔵のフラッシュ/EE プログラム・メモリへイン・サーキットでコードをダウンロードする機能です。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・プログラマが不要になります。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。必要なことは、Cortex-M3 ベースの ADuCxxx へシリアル・ポート・アクセスすることだけです。つまり、デバイスを交換しなくても、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ということです。

Cortex-M3 ベースの ADuCxxx デバイスはすべて、パワーオン時またはリセット時または特定のリセット時に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。

シリアル・ダウンロード・モードの開始条件については、デバイスごとのユーザー・ガイドを参照してください。例えば、ADuCM360 は、カーネル実行時に P2.2 入力ピンがチェックされます。パワーアップ後または何らかのリセット後にこのピンがロー・レベルである場合、デバイスはシリアル・ダウンロード・モードになります。

このモードでは、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 を設定した後、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・マシンと通信して、フラッシュ/EE メモリ・スペースへデータをダウンロードします。ダウンロードするプログラム・データのフォーマットは、リトル・エンディアンである必要があります。

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で動作します。プログラミング用の高電圧は、チップ内で発生するため不要です。

開発ツールの一部として、アナログ・デバイセズは Windows® プログラム（CM3WSD.exe）を提供しています。このプログラムを使うと、COM1 ～ COM31 の PC シリアル・ポートから Cortex-M3 ベースの ADuCxxx デバイスへコードをダウンロードすることができます。ただし、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・プロトコルにホスト・マシンが準拠する限り、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 Cortex-M3 ベースの ADuCxxx デバイスへダウンロードできることに注意してください。

このアプリケーション・ノートでは、Cortex-M3 ベースの ADuCxxx 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 Cortex-M3 ベースの組込み型 ADuCxxx デバイスとの間のプロトコル）を理解し、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 Cortex-M3 ベースの ADuCxxx デバイス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、Cortex-M3 ベースの ADuCxxx デバイス内蔵のシリアル・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。

アナログ・デバイセズ社は、提供する情報が正確で信頼できるものであることを期していますが、その情報の利用に関して、あるいは利用によって生じる第三者の特許やその他の権利の侵害に関して一切の責任を負いません。また、アナログ・デバイセズ社の特許または特許の権利の使用を明示的または暗示的に許諾するものでもありません。仕様は、予告なく変更される場合があります。本紙記載の商標および登録商標は、それぞれの所有者の財産です。※日本語版資料は REVISION が古い場合があります。最新の内容については、英語版をご参照ください。

Rev. A

©2015 Analog Devices, Inc. All rights reserved.

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　新大阪トラストタワー
電話 06（6350）6868

目次

はじめに.. 1
改訂履歴.. 2
MicroConverter ローダの実行.. 3
物理インターフェース 3
データ・トランスポート・パケット・フォーマットの定義......... 3
コマンド 4
コマンド例 5
LFSR コード例.. 6

改訂履歴

1/13—Rev. 0 to Rev. A

Changes to Introduction.. 1

9/12—Revision 0: Initial Version

MicroConverter ローダの実行

抵抗（代表値 1 kΩ）を介して特定の GPIO ピンをロー・レベルにプルダウンし、さらにデバイス自体の RESET 入力ピンをトグルしてリセットすると、ADuCxxx デバイス上のローダが起動されます。（）特定の GPIO がロー・レベルへプルダウンされた状態では、ウォッチドッグ・リセット、パワーオン・リセット、ソフトウェア・リセットなどのリセットを行っても、シリアル・ダウンロード・モードは開始されません。シリアル・ダウンロード・モードの開始条件については、デバイスのユーザー・ガイドを参照してください。

例えば、ADuCM360 では、カーネル実行時に P2.2 入力ピンがチェックされます。このピンがロー・レベルで、かつ RSTSTA.EXTRST = 0x1 の場合、P2.2 入力ピンがチェックされて、デバイスはシリアル・ダウンロード・モードになります。

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、ADuCxxx 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 ～ 115,200 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

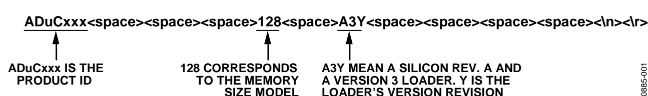

図 1. ID データ・パケットの例

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、データ転送を開始できます。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、有効なデータ・パケットの開始を検出する際にローダが使います。

バイト数フィールド

次のフィールドは、合計バイト数フィールドです。最小バイト数は 5 で、これはコマンド・フィールドと値フィールドに該当します。最大許容バイト数は 255 で、これはコマンド機能、4 バイト値、250 バイトのデータです。

コマンド・フィールド（データ 1）

コマンド・フィールドは、データ・パケットの機能を指定します。4 種類の有効コマンド機能を指定することができます。4 種類のコマンド機能は、4 文字の ASCII 文字（E、W、V、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。

値フィールド（データ 2 ～ データ 5）

値フィールドは、ビッグ・エンディアン・フォーマットの 32 ビット値です。

データ・バイト・フィールド（データ 6 ～ データ 255）

データ・バイト・フィールドは、最大 250 バイトのデータです。

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（存在する値だけ）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 の補数値です。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 0x00 になるはずです。これは、数学的に次のように表されます。

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

別の表現をすれば、開始 ID を除くすべてのバイトの 8 ビット和が 0x00 になるということです。

コマンドのアクノリッジ

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

不正なチェックサムまたは無効なアドレスをローダが受信したとき、ローダは BEL を送信します。古いデータ（まだ消去されていないデータ）にダウンロード・データが上書きされる場合でも、ローダは警告を発しません。（）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 PC インターフェース側は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 Bytes	Command	Value				Data Bytes	Checksum
ID0	ID1		Data 1	Data 2	Data 3	Data 4	Data 5	Data [x]	CS
0x07	0x0E	0x05 to 0xFF	E, W, V, or R	MSB			LSB	0x00 to 0xFF	0x00 to 0xFF

コマンド

内蔵ローダの全コマンドのリストを表 2 に示します。

消去コマンド

消去コマンドを使うと、ユーザーは値フィールドで指定される特定の開始ページのフラッシュ/EE を消去することができます。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、書込む先頭データ・バイトのアドレス、書込むデータ・バイトが含まれます。バイトは、到着したときにフラッシュ/EE に書込まれます。チェックサムが不正な場合、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

ローダはページの内容を確認するために、ページの最後の 4 バイトの値と、最後の 4 バイトを除くページの 24 ビット LFSR の 2 つの情報を必要とします（LFSR コード例のセクションを参照）。

ページを確認するときは、次の 2 ステップのシーケンスに従う必要があります。確認対象の各ページに対して次の 2 ステップ・シーケンスを繰り返します。

1. 値 0x80000000 を値フィールドへ、ページの最後の 4 バイトをデータ・バイト・フィールドへ、それぞれ送信します。
2. 開始ページ・アドレスを値フィールドへ、ページの SIGN コマンドの結果をデータ・バイト・フィールドへ、それぞれ送信します。

ローダはこれらの 2 つのパケットを受信した後、特定のページの LFSR を計算して、入力された値と比較します。比較結果が正しく、かつそのページのアドレス 0x1FC の値がステップ 1 で指定された値に一致する場合、ACK （0x06）が返されます。その他の場合には、BEL （0x07）が返されます。

リモート・リセット・コマンド

ホストは、すべてのデータ・パケットをローダへ送信した後、ローダにリセットを実行させる最後のパケットを送信することができます。ソフトウェア・セルフ・リセットが組み込まれています。値フィールドは常に 0x1 です。

ホストは、シリアル・プログラミングの開始に使用する特定の GPIO ピンをこのコマンドの発行前にアサートしないようにする必要があります。デバイスがリセットされると、カーネルは通常の動作を再開します。ローダ開始チェックがもう一度行われるため、このとき特定の GPIO ピンのアサートを解除しておく必要があります（カーネルは RSTSTA を変更しませんので、外部リセットをチェックすれば、外部リセットが行われたかどうかが検出されます）。表 7 に、リモート・リセットの例を示します。

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）
Remote Reset	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 Bytes	Command	Value				No. of Pages	Checksum
ID0	ID1		Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	CS
0x07	0x0E	0x06	E （0x45）	0x00	ADR[23:16]	ADR[15:8]	ADR[7:0]	0x01 to 0xFF	0x00 to 0xFF

表 4.フラッシュ/EE メモリ書込みコマンド

Start ID		No. of Bytes	Command	Value				Data Bytes	Checksum
ID0	ID1		Data 1	Data 2	Data 3	Data 4	Data 5	Data [x]	CS
0x07	0x0E	5 + x （0x06 to 0xFF）	W （0x57）	0x00	ADR[23:16]	ADR[15:8]	ADR[7:0]	0x00 to 0xFF	0x00 to 0xFF

表 5.フラッシュ/EE メモリ検証コマンド、ステップ 1

Start ID		No. of Bytes	Command	Value				Data Bytes				Checksum
ID0	**ID1**		**Data 1**	**Data 2**	**Data 3**	**Data 4**	**Data 5**	**Data 6**	**Data 7**	**Data 8**	**Data 9**	**CS**
0x07	0x0E	0x09	V （0x56）	0x80	0x00	0x00	0x00	Data at 0x1FC	Data at 0x1FD	Data at 0x1FE	Data at 0x1FF	0x00 to 0xFF

表 6.フラッシュ/EE メモリ検証コマンド、ステップ 2

Start ID		No. of Bytes	Command	Value				Data Bytes				Checksum
ID0	**ID1**		**Data 1**	**Data 2**	**Data 3**	**Data 4**	**Data 5**	**Data 6**	**Data 7**	**Data 8**	**Data 9**	**CS**
0x07	0x0E	0x09	V （0x56）	0x00	ADR[23:16]	ADR[15:8]	ADR[7:0]	LFSR[0:7]	LFSR[15:8]	LFSR [23:16]	0x00	0x00 to 0xFF

表 7.リモート・リセット・コマンド

Start ID		No. of Bytes	Command	Value				Checksum
ID0	**ID1**		**Data 1**	**Data 2**	**Data 3**	**Data 4**	**Data 5**	**CS**
0x07	0x0E	0x05	R （0x52）	0x00	0x00	0x00	0x01	0xA8

コマンド例

ポート・アナライザを使って取得したデータの例を次に示します。

消去コマンド

0x00000200 の 1 ページ消去、
IRP_MJ_WRITE Length 10: 07 0E 06 45 00 00 02 00 01 B2
IRP_MJ_READ Length 1 : 06

ユーザー・スペース全体のマス消去、
IRP_MJ_WRITE Length 10: 07 0E 06 45 00 00 00 00 00 B5
IRP_MJ_READ Length 1: 06

書込みコマンド

0x00000200 から開始の 16 データ・バイトの書込み、
IRP_MJ_WRITE Length 25: 07 0E 15 57 00 00 02 00 77 FF 2C B1 00 20 00 F0 5A FC 08 B1 01 20 00 E0 1F
IRP_MJ_READ Length 1 : 06

検証コマンド

次の検証コマンドに対する 0x1FC の値を 0x11223344 に指定、
IRP_MJ_WRITE Length 13: 07 0E 09 56 80 00 00 00 *44 33 22 11 77*
IRP_MJ_READ Length 1 : 06

0x00000200 のページ検証、LFSR を 0x00841B81 に指定、最終値が 0x11223344 であることをチェック、
IRP_MJ_WRITE Length 13: 07 0E 09 56 00 00 02 00 81 1B 84 00 7F
IRP_MJ_READ Length 1 : 06

リモート・リセット・コマンド

IRP_MJ_WRITE Length 9: 07 0E 05 52 00 00 00 01 A8
IRP_MJ_READ Length 1: 06

LFSR コード例

シグネチャは、多項式 $x^{24}+x^{23}+x^{6}+x^{5}+x+1$ を使用する 24 ビット CRC。初期値は 0xFFFFFF。

```
long int GenerateChecksumCRC24_D32 (unsigned long ulNumValues,unsigned long *pulData)
{
   unsigned long i,ulData,lfsr = 0xFFFFFF;

   for (i= 0x0; i < ulNumValues;i++)
      {
      ulData = pulData[i];
      lfsr = CRC24_D32 (lfsr,ulData) ;
      }

   return lfsr;
}

static unsigned long CRC24_D32 (const unsigned long old_CRC, const unsigned long Data)
{
   unsigned long D      [32];
   unsigned long C      [24];
   unsigned long NewCRC [24];
   unsigned long ulCRC24_D32;
   unsigned long int f, tmp;
   unsigned long int bit_mask = 0x000001;

   tmp = 0x000000;
   // Convert previous CRC value to binary.
   bit_mask = 0x000001;
   for (f = 0; f <= 23; f++)
      {
      C[f]   =  (old_CRC & bit_mask)  >> f;
      bit_mask          = bit_mask << 1;
      }

   // Convert data to binary.
   bit_mask = 0x000001;
   for (f = 0; f <= 31; f++)
      {
      D[f]   =  (Data & bit_mask)  >> f;
      bit_mask       = bit_mask << 1;
      }

   // Calculate new LFSR value.
   NewCRC[0] = D[31] ^ D[30] ^ D[29] ^ D[28] ^ D[27] ^ D[26] ^ D[25] ^
               D[24] ^ D[23] ^ D[17] ^ D[16] ^ D[15] ^ D[14] ^ D[13] ^
               D[12] ^ D[11] ^ D[10] ^ D[9] ^ D[8] ^ D[7] ^ D[6] ^
               D[5] ^ D[4] ^ D[3] ^ D[2] ^ D[1] ^ D[0] ^ C[0] ^ C[1] ^
               C[2] ^ C[3] ^ C[4] ^ C[5] ^ C[6] ^ C[7] ^ C[8] ^ C[9] ^
               C[15] ^ C[16] ^ C[17] ^ C[18] ^ C[19] ^ C[20] ^ C[21] ^
               C[22] ^ C[23];
   NewCRC[1] = D[23] ^ D[18] ^ D[0] ^ C[10] ^ C[15];
   NewCRC[2] = D[24] ^ D[19] ^ D[1] ^ C[11] ^ C[16];
   NewCRC[3] = D[25] ^ D[20] ^ D[2] ^ C[12] ^ C[17];
   NewCRC[4] = D[26] ^ D[21] ^ D[3] ^ C[13] ^ C[18];
   NewCRC[5] = D[31] ^ D[30] ^ D[29] ^ D[28] ^ D[26] ^ D[25] ^ D[24] ^
               D[23] ^ D[22] ^ D[17] ^ D[16] ^ D[15] ^ D[14] ^ D[13] ^
               D[12] ^ D[11] ^ D[10] ^ D[9] ^ D[8] ^ D[7] ^ D[6] ^
               D[5] ^ D[3] ^ D[2] ^ D[1] ^ D[0] ^ C[0] ^ C[1] ^ C[2] ^
               C[3] ^ C[4] ^ C[5] ^ C[6] ^ C[7] ^ C[8] ^ C[9] ^ C[14] ^
               C[15] ^ C[16] ^ C[17] ^ C[18] ^ C[20] ^ C[21] ^ C[22] ^
               C[23];
   NewCRC[6] = D[28] ^ D[18] ^ D[5] ^ D[0] ^ C[10] ^ C[20];
   NewCRC[7] = D[29] ^ D[19] ^ D[6] ^ D[1] ^ C[11] ^ C[21];
   NewCRC[8] = D[30] ^ D[20] ^ D[7] ^ D[2] ^ C[12] ^ C[22];
   NewCRC[9] = D[31] ^ D[21] ^ D[8] ^ D[3] ^ C[0] ^ C[13] ^ C[23];
   NewCRC[10] = D[22] ^ D[9] ^ D[4] ^ C[1] ^ C[14];
   NewCRC[11] = D[23] ^ D[10] ^ D[5] ^ C[2] ^ C[15];
   NewCRC[12] = D[24] ^ D[11] ^ D[6] ^ C[3] ^ C[16];
   NewCRC[13] = D[25] ^ D[12] ^ D[7] ^ C[4] ^ C[17];
   NewCRC[14] = D[26] ^ D[13] ^ D[8] ^ C[0] ^ C[5] ^ C[18];
   NewCRC[15] = D[27] ^ D[14] ^ D[9] ^ C[1] ^ C[6] ^ C[19];
   NewCRC[16] = D[28] ^ D[15] ^ D[10] ^ C[2] ^ C[7] ^ C[20];
   NewCRC[17] = D[29] ^ D[16] ^ D[11] ^ C[3] ^ C[8] ^ C[21];
   NewCRC[18] = D[30] ^ D[17] ^ D[12] ^ C[4] ^ C[9] ^ C[22];
```

```
   NewCRC[19] = D[31] ^ D[18] ^ D[13] ^ C[5] ^ C[10] ^ C[23];
   NewCRC[20] = D[19] ^ D[14] ^ C[6] ^ C[11];
   NewCRC[21] = D[20] ^ D[15] ^ C[7] ^ C[12];
   NewCRC[22] = D[21] ^ D[16] ^ C[8] ^ C[13];
   NewCRC[23] = D[31] ^ D[30] ^ D[29] ^ D[28] ^ D[27] ^ D[26] ^ D[25] ^
                D[24] ^ D[23] ^ D[22] ^ D[16] ^ D[15] ^ D[14] ^ D[13] ^
                D[12] ^ D[11] ^ D[10] ^ D[9] ^ D[8] ^ D[7] ^ D[6] ^
                D[5] ^ D[4] ^ D[3] ^ D[2] ^ D[1] ^ D[0] ^ C[0] ^ C[1] ^
                C[2] ^ C[3] ^ C[4] ^ C[5] ^ C[6] ^ C[7] ^ C[8] ^ C[14] ^
                C[15] ^ C[16] ^ C[17] ^ C[18] ^ C[19] ^ C[20] ^ C[21] ^
                C[22] ^ C[23];

   ulCRC24_D32 = 0;
   // LFSR value from binary to hex.
   bit_mask = 0x000001;
   for  (f = 0; f <= 23; f++)
      {
      ulCRC24_D32 = ulCRC24_D32 + NewCRC[f] * bit_mask;
      bit_mask = bit_mask << 1;
      }
   return (ulCRC24_D32 & 0x00FFFFFF) ;
}
```

©2015 Analog Devices, Inc. All rights reserved.　商標および登録商標は、それぞれの所有者の財産です。