AN-1160
APPLICATION NOTE

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106, U.S.A. • Tel: 781.329.4700 • Fax: 781.461.3113 • www.analog.com

Cortex-M3-Based Serial Download Protocol

INTRODUCTION

A key feature of the Analog Devices, Inc., Cortex-M3-based device family is the ability of the devices to download code to their on-chip Flash/electronically erasable (EE) program memory while in circuit. An in circuit code download is conducted over the device universal asynchronous receiver transmitter (UART) serial port, and is thus commonly referred to as a serial download.

The serial download capability allows developers to reprogram the device while it is soldered directly onto the target system, thus avoiding the need for an external device programmer. The serial download feature also enables system upgrades to be performed in the field; all that is required is serial port access to the Cortex-M3-based device. This serial download feature means manufacturers can upgrade system firmware in the field without having to swap out the device.

Any Cortex-M3-based device can be configured for serial download mode via a specific pin configuration at power-on or after any reset or a specific reset.

Refer to the device specific hardware reference manual for the entry criteria to serial download mode. For example, on the ADuCM360, the P2.2 input pin is checked during kernel execution. If this pin is low after power-up or any type of reset, the device enters serial download mode.

In serial download mode, an on-chip resident loader routine is initiated. The on-chip loader configures the device UART and, via a specific serial download protocol, communicates with any host machine to manage the download of data into its Flash/EE memory spaces. The format of the program data to download must be little endian.

Serial download mode operates within the standard supply rating of the device. Therefore, there is no requirement for a specific high programming voltage because it is generated on-chip.

As part of the development tools, a Windows® program (CM3WSD.exe) is provided by Analog Devices. This program allows the user to download code from the PC serial ports, COM1 to COM31, inclusive, to the Cortex-M3-based device. Note, however, that any master host machine—PC, microcontroller, or digital signal processor (DSP)—can download to the Cortex-M3-based device if the host machine adheres to the serial download protocols detailed in this application note.

This application note details the Cortex-M3-based device serial download protocol, allowing end users to understand and implement this protocol (embedded host to embedded Cortex-M3-based device) in an end target system.

For the purposes of clarity, the term host refers to the host machine (PC, microcontroller, or DSP) attempting to download data to the Cortex-M3-based device. The term loader refers to the on-chip serial download firmware on the Cortex-M3-based device.

The Analog Devices Cortex-M3-based device family includes the ADuCM310, the ADuCM320/ADuCM322, the ADuCM360/ADuCM361/ADuCM362/ADuCM363, and the ADuCRF101.

TABLE OF CONTENTS

Introduction 1
Revision History 2
Running the Microconverter Loader 3
Physical Interface 3
Defining the Data Transport Packet Format 3
Commands 4
Command Example 6
LFSR Code Example 7

REVISION HISTORY

11/2017—Rev. A to Rev. B

Changes to Application Note Title and Introduction Section 1
Changes to Running the Microconverter Loader Section, Physical Interface Section, Figure 1, Checksum Field Section, and Table 1 3
Changes to Verify Command for ADuCM360/ADuCM361 and ADuCRF101 Section, Table 3, Table 4, Table 5, and Table 6 4
Added Verify Command for ADuCM362/ADuCM363, ADuCM320/ADuCM322, and ADuCM310 Section, Cyclic Redundancy Check (CRC) Calculation Section, Table 7, Table 8, and Table 9; Renumbered Sequentially 5
Changes to Remote Reset Command and Table 10 5
Changes to Erase Command Example Section, Write Command Example Section, Verify Command for ADuCM360/ADuCM361 and ADuCRF101 Example Section, Verify Command for ADuCM362/ADuCM363, ADuCM320/ADuCM322, and ADuCM310 Example Section, and Remote Reset Command Example Section 6

1/2013—Rev. 0 to Rev. A

Changes to Introduction 1

9/2012—Revision 0: Initial Version

RUNNING THE MICROCONVERTER LOADER

The loader on the Cortex-M3-based device is initiated by pulling a specific GPIO pin low through a resistor (typically 1 kΩ pull-down) and resetting the device by toggling the RESET input pin on the device itself. Other resets, such as watchdog reset, power-on reset, and software reset with specific GPIO pulled low, result in serial download mode entry for all Cortex-M3-based devices except the ADuCRF101. Refer to the device hardware reference manual for the entry criteria to serial download mode.

For example, on the ADuCM360, the P2.2 input pin is checked during kernel execution. If this pin is low and the RSTSTA register ≠ 0x00 at the time that the P2.2 input pin is checked, the device enters serial download mode.

PHYSICAL INTERFACE

After it is triggered, the loader waits for the host to send a backspace (BS = 0x08) character to synchronize. The loader measures the timing of this character and, accordingly, configures the device UART serial port to transmit or receive at the baud rate of the host with 8 data bits and no parity. The baud rate must be between 600 bps and 115,200 bps, inclusive.

When it receives the backspace, the loader immediately sends the following 24-byte ID data packet:

- 15 bytes = product identifier
- 3 bytes = hardware and firmware version number
- 4 bytes = reserved for future use
- 2 bytes = line feed and carriage return

See Figure 1 for an example ID data packet.

DEFINING THE DATA TRANSPORT PACKET FORMAT

When the UART has been configured, a data transfer can begin. The general communications data transport packet format is shown in Table 1.

Packet Start ID Field

The first field is the packet start ID field, which contains two start characters (0x07 and 0x0E). These bytes are constant and are used by the loader to detect a valid data packet start.

Number of Bytes Field

The next field is the total number of bytes field. The minimum number of bytes is five, which corresponds to the command field and the value field. The maximum number of bytes allowed is 255: a command function, a 4-byte value, and 250 bytes of data.

Command Field (Data 1)

The command field describes the function of the data packet. One of four valid command functions is allowed. The four command functions are described by one of four ASCII characters: E, W, V, or R. The list of data packet command functions is shown in Table 2.

Value Field (Data 2 to Data 5)

The value field contains a 32-bit value in big endian format.

Data Bytes Field (Data 6 to Data 255)

The data bytes field contains a maximum of 250 data bytes.

Checksum Field

The data packet checksum is written into the checksum field. The twos complement checksum is calculated from the summation of the hexadecimal values in the number of bytes field and the hexadecimal values in the Data 1 to Data 255 fields (however many exist). The checksum is the twos complement value of this summation. Thus, the LSB of the sum of all the bytes from the number of data bytes to the checksum inclusive is 0x00. This can also be expressed mathematically as

$$CS = 0x00 - (Number\ of\ Bytes + \sum_{N-1}^{255} Data\ Byte_N\ )$$

where *CS* is the checksum.

Expressed differently, the 8-bit sum of all bytes excluding the start ID must be 0x00.

Acknowledge of Command

The loader routine issues a bell (BEL) (0x07) as a negative response or an acknowledge (ACK) (0x06) as a positive response to each data packet.

A BEL is transmitted by the loader if it receives an incorrect checksum or an invalid address. The loader does not provide a warning if data is downloaded over old (unerased) data. The PC interface must ensure that any location where code is downloaded is erased.

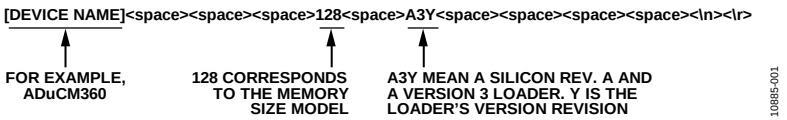

Figure 1. Example ID Data Packet

Table 1. Data Transport Packet Format

Start ID			Command	Value				Data Bytes	Checksum
ID0	ID1	No. of Bytes	Data 1	Data 2	Data 3	Data 4	Data 5	Data [x]	CS
0x07	0x0E	0x05 to 0xFF	E, W, V, or R	MSB	[N/A][1]	[N/A][1]	LSB	0x00 to 0xFF	0x00 to 0xFF

[1] N/A means not applicable.

COMMANDS

The complete list of commands implemented in the on-chip loader is shown in Table 2.

Erase Command

The erase command allows the user to erase Flash/EE memory from a specific start page address determined by the value field. This command also includes the number of pages to erase.

If the address is 0x00000000 and the number of pages is 0x00, the loader interprets this value as a mass erase command, erasing the entire user code space.

The data packet for the erase command is shown in Table 3.

Write Command

The write command includes the number of data bytes (5 + x), the command, the address of the first data byte to program, and the data bytes to program. The bytes are programmed into Flash/EE memory as they arrive. The loader sends a BEL if the checksum is incorrect or if the address received is out of range. If the host receives a BEL from the loader, the download process must be aborted and the entire download sequence started again.

Verify Command for ADuCM360/ADuCM361 and ADuCRF101

The loader requires two pieces of information to verify the contents of a page: the contents of the last 4 bytes of the page and the 24-bit linear feedback shift register (LFSR) of the page excluding the last 4 bytes (see the LFSR Code Example section).

To verify a page, a two step sequence must be followed. Repeat this two step sequence for each page to be verified.

1. Send the value 0x80000000 in the value field and the last 4 bytes of the page in the data bytes field.
2. Send the start page address in the value field and the result of the SIGN command of the page in the data bytes field.

After receiving these two packets, the loader computes the LFSR of the specified page and compares it to the supplied value. If it is correct and the value at Address 0x1FC of that page matches the value specified in Step 1, ACK (0x06) is returned; otherwise, BEL (0x07) is returned.

Table 2. Data Packet Command Functions

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)

Table 3. Erase Flash/EE Memory Command

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

Table 4. Write Flash/EE Memory Command

Start ID			Command	Value				Data Bytes	Checksum
ID0	ID1	No. of Bytes	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

Table 5. Verify Flash/EE Memory Command for ADuCM360/ADuCM361 and ADuCRF101, Step 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 (LSB)	Data 7	Data 8	Data 9 (MSB)	CS
0x07	0x0E	0x09	V (0x56)	0x80	0x00	0x00	0x00	Last four bytes of the page				0x00 to 0xFF

Table 6. Verify Flash/EE Memory Command for ADuCM360/ADuCM361 and ADuCRF101, Step 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

Verify Command for ADuCM362/ADuCM363, ADuCM320/ADuCM322, and ADuCM310

The loader requires two pieces of information to verify the contents of a page: the contents of the last 8 bytes of the page and the 32-bit forward signature of the page excluding the last 8 bytes.

To verify a page, a three step sequence must be followed. Repeat this three step sequence for each page to be verified.

1. Send the value 0x80000000 in the value field and the last word of the page in the data bytes field.
2. Send the value 0x90000000 in the value field and the second to last word of the page in the data bytes field.
3. Send the start page address in the value field and the forward signature of the page in the data bytes field.

After receiving these three packets, the loader computes the forward signature of the specified page using the flash controller FSIGN command and compares it to the supplied value. If it is correct and the value at Address 0x1FC of that page matches the value specified in Step 1, ACK (0x06) is returned; otherwise, BEL (0x07) is returned.

Remote Reset Command

After the host transmits all data packets to the loader, the host can send a final packet instructing the loader to perform a reset. A software self reset is implemented. The value field must always be 0x1.

The host must ensure that the specific GPIO pin used to initiate the serial programming is no longer asserted before issuing this command. When the device resets, reenter the kernel as normal. The loader entry check is performed once more; therefore, the specific GPIO pin must be deasserted at this time. (The kernel does not modify the RSTSTA register, so the check for an external reset still detects that an external reset occurred). Table 10 shows an example of a remote reset.

Cyclic Redundancy Check (CRC) Calculation

For the ADuCM360/ADuCM361 and the ADuCRF101, the signature is a 24-bit CRC with the polynomial

$$x^{24} + x^{23} + x^6 + x^5 + x + 1$$

The initial value is 0xFFFFFFFF.

For the ADuCM362/ADuCM363, the ADuCM320/ADuCM322, and the ADuCM310, the signature is a 32-bit CRC with the polynomial

$$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$$

The initial value is 0xFFFFFFFF.

Table 7. Verify Flash/EE Memory Command for ADuCM362/ADuCM363, ADuCM320/ADuCM322, and ADuCM310, Step 1

Start ID		No. of	Command	Value				Data Bytes				Checksum
ID0	ID1	Bytes	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6 (LSB)	Data 7	Data 8	Data 9 (MSB)	CS
0x07	0x0E	0x09	V (0x56)	0x80	0x00	0x00	0x00	Last four bytes of the page				0x00 to 0xFF

Table 8. Verify Flash/EE Memory Command for ADuCM362/ADuCM363, ADuCM320/ADuCM322, and ADuCM310, Step 2

Start ID		No. of	Command	Value				Data Bytes				Checksum
ID0	ID1	Bytes	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6 (LSB)	Data 7	Data 8	Data 9 (MSB)	CS
0x07	0x0E	0x09	V (0x56)	0x90	0x00	0x00	0x00	Second to last four bytes of the page				0x00 to 0xFF

Table 9. Verify Flash/EE Memory Command for ADuCM362/ADuCM363, ADuCM320/ADuCM322, and ADuCM310, Step 3

Start ID		No. of	Command	Value				Data Bytes				Checksum
ID0	ID1	Bytes	Data 1	Data 2	Data 3	Data 4	Data 5	Data 6	Data 7	Data 8	Data 9	CS
0x07	0x0E	0x09	V (0x56)	ADR [32:24]	ADR [23:16]	ADR [15:8]	ADR [7:0]	LFSR [0:7]	LFSR [15:8]	LFSR [23:16]	LFSR [32:24]	0x00 to 0xFF

Table 10. Remote Reset Command

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

COMMAND EXAMPLES

The following is an example of data captured using a port analyzer.

Erase Command Example

To erase one page at 0x00000200, issue the following command:

```
IRP_MJ_WRITE   Length 10: 07 0E 06 45 00 00 02 00 01 B2
IRP_MJ_READ    Length 1 : 06
```

To mass erase the entire user space, issue the following command:

```
IRP_MJ_WRITE   Length 10: 07 0E 06 45 00 00 00 00 00 B5
IRP_MJ_READ    Length 1: 06
```

Write Command Example

To write 16 data bytes starting at 0x00000200, issue the following command:

```
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
```

Verify Command for ADuCM360/ADuCM361 and ADuCRF101 Example

If the value at 0x1FC for the next verify command is specified to be 0x11223344, issue the following command:

```
IRP_MJ_WRITE   Length 13: 07 0E 09 56 80 00 00 00 44 33 22 11 77
IRP_MJ_READ    Length 1 : 06
```

If the verify of the page is at 0x00000200 and the LFSR is specified to be 0x00841B81, the last value is checked against 0x11223344. Issue the following command:

```
IRP_MJ_WRITE   Length 13: 07 0E 09 56 00 00 02 00 81 1B 84 00 7F
IRP_MJ_READ    Length 1 : 06
```

Verify Command for ADuCM362/ADuCM363, ADuCM320/ADuCM322, and ADuCM310 Example

If the value at 0x7FC for the next verify command is specified to be 0x11223344, issue the following command:

```
IRP_MJ_WRITE   Length 13: 07 0E 09 56 80 00 00 00 44 33 22 11 77
IRP_MJ_READ    Length 1 : 06
```

If the value at 0x7F8 for the next verify command is specified to be 0x55667788, issue the following command:

```
IRP_MJ_WRITE   Length 13: 07 0E 09 56 90 00 00 00 88 77 66 55 57
IRP_MJ_READ    Length 1 : 06
```

If the verify of the page is at 0x00000800 and LFSR is specified to be 0x00841B81, the last value is checked against 0x11223344 and 0x55667788. Issue the following command:

```
IRP_MJ_WRITE   Length 13: 07 0E 09 56 00 00 08 00 81 1B 84 00 79
IRP_MJ_READ    Length 1 : 06
```

Remote Reset Command Example

To perform a reset to the loader, issue the following command:

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

LFSR CODE EXAMPLE

The signature is a 24-bit CRC with the polynomial $x^{24} + x^{23} + x^6 + x^5 + x + 1$. The initial value is 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);
}
```


AN10885-0-11/17(B)