ANALOG DEVICES

AN-724
APPLICATION NOTE

One Technology Way • P.O. Box 9106 • Norwood, MA 02062-9106 • Tel: 781/329-4700 • Fax: 781/326-8703 • www.analog.com

ADuC702x Serial Download Protocol

by Aude Richard

INTRODUCTION

One of the many features of the MicroConverter® product family is the ability of the device to download code to its on-chip Flash/EE program memory while in-circuit. This in-circuit code download feature is conducted over the device UART serial port, and is thus commonly referred to as serial download. Serial download capability allows developers to reprogram the part while it is soldered directly onto the target system, avoiding the need for an external device programmer. Serial download also opens up the possibility of system upgrades in the field; all that is required is serial port access to the MicroConverter. This means manufacturers can upgrade system firmware in the field without having to swap out the device.

Any MicroConverter device can be configured for serial download mode via a specific pin configuration at power-on or during application of the external reset signal. For the ADuC702x family of MicroConverters, the P0.0 input pin is pulled low through a resistor (1 kΩ). If this condition is detected by the part on power-on or during application of a hard reset input, the part will enter serial download mode. In this 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.

It should be noted that serial download mode operates within the standard supply rating of the part (2.7 V to 3.6 V). Therefore, there is no requirement for a specific high programming voltage since it's generated on-chip. Figure 1 shows how to enter serial download mode on an evaluation board.

As part of Analog Devices' QuickStart™ Development Tools, a Windows® executable program is provided (C:\ADuC702x\Download\ARMWSD.exe) that allows the user to download code from the PC (PC serial ports COM1, 2, 3, or 4) to the MicroConverter. However, it should be emphasized that any master host machine (PC, microcontroller, or DSP) can download to the MicroConverter once the host machine adheres to the serial download protocols detailed in this application note.

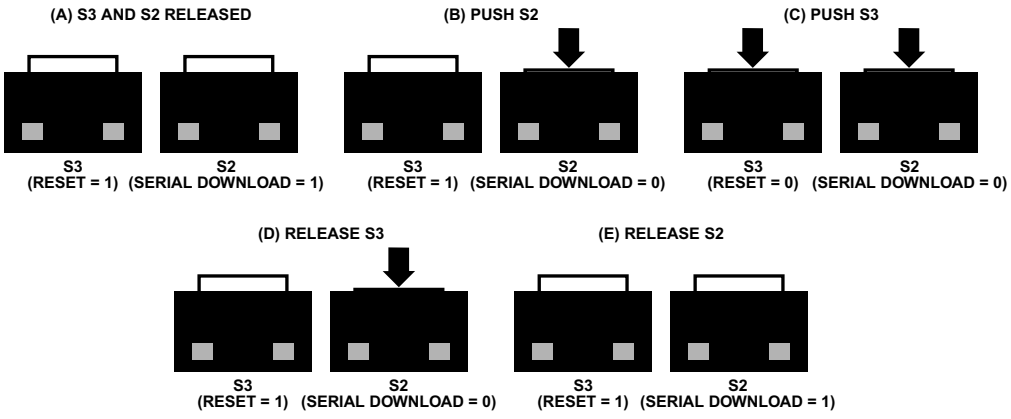

Figure 1. MicroConverter in Serial Download Mode

This application note outlines in detail the MicroConverter serial download protocol, allowing end users to both fully understand the protocol and, if required, to implement this protocol (embedded host to embedded MicroConverter) successfully 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 MicroConverter. The term "loader" refers specifically to the on-chip serial download firmware on the MicroConverter.

RUNNING THE MICROCONVERTER LOADER

The loader on the ADuC702x MicroConverter is run by pulling the P0.0 pin (serial download) low through a resistor (typically 1 kΩ pull-down) and resetting the part (toggling the RST input pin on the part itself, or a power cycle will reset the part).

THE PHYSICAL INTERFACE

Once triggered, the loader waits for the host to send a backspace (BS = 0x08) character to synchronize. It then configures the MicroConverter UART serial port to transmit/receive at the host's baud rate, 8 data bits and no parity. The baud rate must be between 600 bps and 19200 bps included. On receiving the backspace, the loader will immediately send the following 24-byte ID data packet at 9600 bps:

15 bytes = product identifier

3 bytes = hardware and firmware version number

4 bytes = reserved for future use

2 bytes = line feed and carriage return

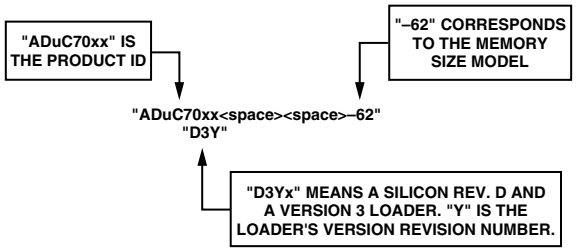

Figure 2. ID Data Packet

DEFINING THE DATA TRANSPORT PACKET FORMAT

Once the UART has been configured, the transfer of data can begin. The general communications data transport packet format is shown in Table I.

Packet Start ID Field

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

Number of Data Bytes Field

The next field is the total number of data bytes, including Data 1 (Command Function). The minimum number of data bytes is five, which corresponds to the command function and the address. The maximum number of data bytes allowed is 255: command function, 4-byte address, and 250 bytes of data.

Command Function Field (Data 1)

The Command Function field describes the function of the data packet. One of four valid command functions are allowed. The four command functions are described by one of four ASCII characters: E, W, V, or R.

Address Field (Data 2 → 5)

The address field contains a 32-bit address, h, u, m, l, with MSB in h and LSB in l.

Data Byte Fields 6 → 255

User code is downloaded/verified by bytes. The data byte field contains a maximum of 250 data bytes. The data must be stripped out of the Intel® Extended Hex 16-byte record format and reassembled by the host as part of the above data form before transmission to the loader.

Checksum Field

The data packet checksum is written into this field. The twos complement checksum is calculated from the summation of the hex values in the number of bytes field and the hex values in the data 1 to 255 fields (if they exist). The checksum is the twos complement value of this summation, i.e., the 8-bit sum of number of bytes plus the number of data bytes 1 to 255, so the checksum should equal 0x00. This can also be expressed mathematically as

$$\text{Checksum} = 0x00 - (\text{No. Data Bytes} + \Sigma\ \text{Data Byte}_N)$$

Table I. Data Transport Packet Format

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

Table II. Data Packet Command Functions

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

Table III. Erase Flash/EE Memory Command

Start ID		No. of Data Bytes	Data 1 CMD	Data 2 → 5 (Address: h, u, m, l)	Data 6 (pages)	Checksum
0x07	0x0E	6	E (0x45)	h, u, m, l	x pages	No. of Data Bytes + Σ Data (Twos Complement)

Table IV. Program Flash/EE Memory Command

Start ID		No. of Data Bytes	Data 1 CMD	Data 2 → 5 (Address: h, u, m, l)	Data x (x = 1 → 250)	Checksum
0x07	0x0E	5 + no. of Data x 5 → 255	W (0x57)	h, u, m, l	Complemented Data Bytes	No. of Data Bytes + Σ Data x –515

The loader routine will respond with a BEL (0x07) as a negative response or an ACK (0x06) as a positive response to the previous data packet communication.

A BEL is transmitted by the loader if it receives an incorrect data packet format on verification of the checksum byte or if data is downloaded to an invalid address. The loader does not give a warning if data is downloaded over old (unerased) data. The PC interface must ensure that any location where code will be downloaded is erased.

The full set of data packet command functions is shown in Table II.

Erase Command

The erase command allows the user to erase from one page up to all pages of Flash/EE from a specific address determined by data 2 → 5. This command also requires the number of pages to erase. If the address is 0x00000000 and the number of pages is 0x00, the loader will interpret it as a mass erase command, erasing the entire user code space.

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

Write Command

The write command requires the number of data bytes (data 1 + data 2 + data 2 → 5 + data x), the command, the address of the first data byte to program, and the data bytes to program. The bytes will be programmed into Flash/EE as they arrive. The loader will send 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 should be aborted and the entire download sequence started again.

Verify Command

The verify command is almost identical to the write command, as shown in Table V. The command field is V (0x56), but to improve the chance of detecting errors the data bytes are modified; the low 5 bits are shifted to the high 5 bits, and the high 3 bits are shifted to the low 3 bits.

Table V. Verify Command, Bit Modifications

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

Table VI. Program Flash/EE Memory Command

Start ID		No. of Data Bytes	Data 1 CMD	Data 2 → 5 (Address: h, u, m, l)	Data 6 (pages)	Checksum
0x07	0x0E	5 + no. of Data x 5 → 255	V (0x56)	h, u, m, l	Complemented Data Bytes	No. of Data Bytes + Σ Data x –515

Table VII. Jump to User Code (Remote RUN) Command

Packet ID		No. of Data Bytes	Data 1 CMD	Data 2 → 5 (Address: h, u, m, l)	Data x (x = 1 → 250)	Checksum
0x07	0x0E	0x05	R (0x52)	h, u, m, l	Complemented	0xA9

The loader restores the correct bit sequence and compares it to the flash contents. If it is correct and the checksum is correct, ACK (0x06) is returned; otherwise BEL (0x07) is returned.

Jump to User Code (Remote Run) Command

Once the host has transmitted all data packets to the loader, the host can send a final packet instructing the loader to force the MicroConverter program counter to a given address, and thus begin executing the code that has just been downloaded. Table VII shows an example of a Remote RUN or "jump to user code" from address 0x00.

Only run from start of the Flash/EE (h, u, m, l = 0x80000) is supported at present.

INTEL EXTENDED HEX FORMAT

Intel Extended Hexadecimal format or Intel Extended Hex format is a standard for storing machine language in displayable ASCII or printable format. It is similar to the Hex 8 format except that the Intel extended linear address record is output to also establish the upper 16 bits of the data address. Each data record begins with a colon, followed by an 8-character prefix and ends with a 2-character checksum. Each record has the following format:

:BBAAAATTHHHH....HHHCC

where:

BB is a 2-digit hexadecimal byte count representing the number of data bytes that will appear on the line;

AAAA is a 4-digit hexadecimal address representing the starting address of the data record;

TT is a 2-digit record type:

00–Data record

01–End of file record

02–Extended segment address record

03–Start segment address record

04–Extended linear address record

05–Start linear address record

HH is a 2-digit hexadecimal data byte

CC is a 2-digit hexadecimal checksum that is the twos complement of the sum of all preceding bytes in the record, including the prefix (sum of all bytes + checksum = 00).

RECORD TYPES

Data Record

Record type 00, the data record, is the record that contains the data of the file. The data record begins with the colon start character (":") followed by the byte count, the address of the first byte, and the record type ("00"). Following the record type are the data bytes. The checksum follows the data bytes and is the twos complement of the preceding bytes in the record, excluding the start character. The following are examples of data records (spaces are included for clarity only and should not be found in a real object file).

:10 0000 00 FFFEFDFCFBFAF9F8F7F6F5F4F3F2F1F0 FF
:05 0010 00 0102030405 AA

End Record

Record type 01, the end record, signals the end of the data file. The end record starts with the colon start character (":") followed by the byte count ("00"), the address ("0000"), the record type ("01"), and the checksum ("FF").

:00 0000 01 FF

Extended Segment Address Record
Record type 02, the extended segment address record, defines Bits 4 through 19 of the segment base address. It can appear anywhere within the object file and it affects the absolute memory address of all subsequent data records in the file until it is changed. The extended segment address record starts with the colon start character (":") followed by the byte count ("02"), the address ("0000"), the record type ("02"), the 4-character hexadecimal number represented by Bits 4 through 19 of the segment base address, and the 2-character checksum.

:02 0000 02 1000 55

Start Segment Address Record
Record type 03, the start segment address record, defines Bits 4 through 19 of the execution start segment base address for the object file.

:02 0000 03 0000 55

Extended Linear Address Record
Record type 04, the extended linear address record, defines Bits 16 through 31 of the destination address. It can appear anywhere within the object file and it affects the absolute memory address of all subsequent data records in the file until it is changed. The extended linear address record starts with the colon start character (":") followed by the byte count ("02"), the address ("0000"), the record type ("04"), the 4-character hexadecimal number represented by Bits 16 through 31 of the destination address, and the 2-character checksum.

:02 0000 04 FFFF 55

Start Linear Address Record
Record type 05, the start linear address record, defines Bits 16 through 31 of the execution start address for the object file.

:02 0000 05 0000 55

Below is an example of an Intel Hexadecimal Object File that contains the following records: extended linear address, extended segment address, data, and end.

:020000040108EA

:0200000212FFBD

:0401000090FFAA5502

:00000001FF

1. Determine the extended linear address offset for the data record (0108 in this example).

   :02 0000 04 0108 EA

2. Determine the extended segment address for the data record (12FF in this example).

   :02 0000 02 12FF BD

3. Determine the address offset for the data in the data record (0100 in this example).

   :04 0100 00 90FFAA55 02

4. Calculate the absolute address for the first byte of the data record.

   + 0108 0000 linear address offset shifted left 16 bits

   + 0001 2FF0 segment address offset shifted left 4 bits

   + 0000 0100 address offset from data record

   = 0109 30F0 32 bit address for first data byte

5. Calculations

   010930F0 90

   010930F1 FF

   010930F2 AA

   010930F3 55

AN04844–0–6/04(0)