AN-724
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

ADuC70xx Serial Download Protocol

INTRODUCTION

A key feature of the MicroConverter® product family is the ability of the devices to download code to their on-chip Flash/EE program memory while in-circuit. An in-circuit code download is conducted over the device UART serial port, and is thus commonly referred to as a serial download. The 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. The serial download feature also enables system upgrades to be performed 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 the application of the external reset signal. For the ADuC70xx family of MicroConverters, the $\overline{\text{BM}}$ input pin is pulled low through a resistor (1 kΩ). If this condition is detected by the part at power-on or during application of a hard reset input, the part enters 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.

Note 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 because it is generated on-chip. Figure 1 shows how to enter serial download mode on an evaluation board.

As a QuickStart™ development tool, a Windows® program (ARMWSD.exe) is provided by Analog Devices, Inc. This program allows the user to download code from PC serial ports COM1 to COM31, inclusive, to the MicroConverter. Note, however, that any master host machine (PC, micro-controller, or DSP) can download to the MicroConverter once the host machine adheres to the serial download protocols detailed in this application note.

This application note details the MicroConverter serial download protocol, allowing end users to understand and to successfully implement this protocol (embedded host to embedded MicroConverter) 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 to the on-chip serial download firmware on the MicroConverter.

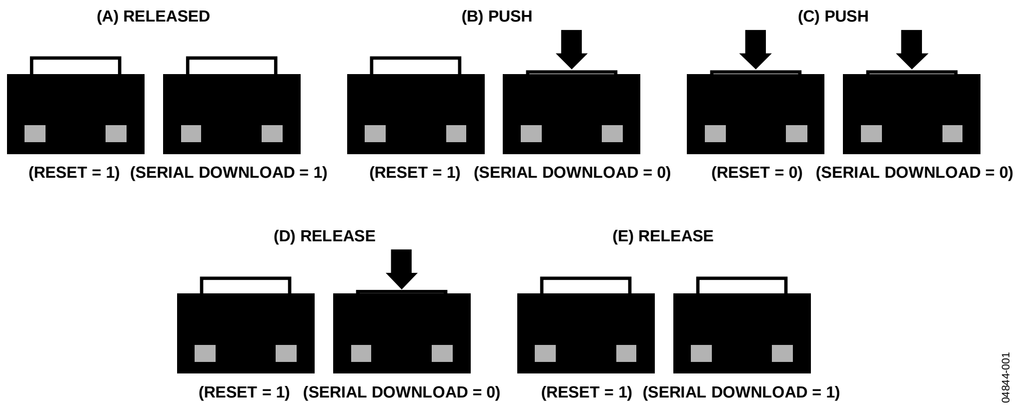
(A) RELEASED
(RESET = 1) (SERIAL DOWNLOAD = 1)
(B) PUSH
(RESET = 1) (SERIAL DOWNLOAD = 0)
(C) PUSH
(RESET = 0) (SERIAL DOWNLOAD = 0)
(D) RELEASE
(RESET = 1) (SERIAL DOWNLOAD = 0)
(E) RELEASE
(RESET = 1) (SERIAL DOWNLOAD = 1)
04844-001

Figure 1. MicroConverter in Serial Download Mode

TABLE OF CONTENTS

RUNNING THE MICROCONVERTER LOADER

The loader on the ADuC70xx MicroConverter is run by pulling the serial download $\overline{BM}$ pin low through a resistor (typically 1 kΩ pull-down) and resetting the part (toggling the RESET input pin on the part itself or power cycling resets the part).

THE PHYSICAL INTERFACE

Once 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 MicroConverter UART serial port to transmit/receive at the host's baud rate with 8 data bits and no parity. The baud rate must be between 600 bps and 115,200 bps included. Note that the ADuC7060/ADuC7061 is limited to a baud rate of 38400 bps. On receiving 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

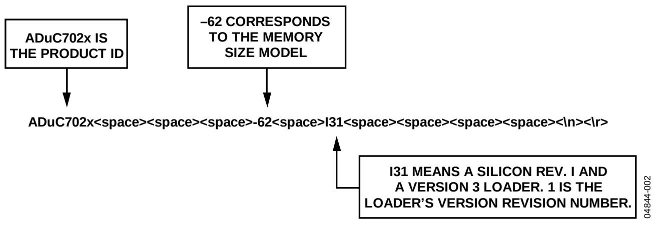
ADuC702x IS THE PRODUCT ID
–62 CORRESPONDS TO THE MEMORY SIZE MODEL
ADuC702x<space><space><space>-62<space>I31<space><space><space><space><\n><\r>
I31 MEANS A SILICON REV. I AND A VERSION 3 LOADER. 1 IS THE LOADER'S VERSION REVISION NUMBER.
04844-002

Figure 2.Example ID Data Packet

DEFINING THE DATA TRANSPORT PACKET FORMAT

Once 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 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: a command function, a 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 five valid command functions is allowed. The five command functions are described by one of five ASCII characters: E, W, V, P, or R. The list of data packet command functions is shown in Table 2.

Address Field (Data 2 to Data 5)

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

Data Byte Field (Data 6 to Data 255)

User code is downloaded/verified by bytes. The data byte field contains a maximum of 250 data bytes.

The data is normally stripped out of the Intel® Hex extended 16-byte record format (see the Intel Extended Hex section) and reassembled by the host as part of the data packet described in Table 1 before transmission to the loader.

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 (as many as 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 should be 0. This can also be expressed mathematically as

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

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

Acknowledge of Command

The loader routine issues a BEL (0x07) as a negative response or an 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 give a warning if data is downloaded over old (unerased) data. The PC interface must ensure that any location where code is downloaded is erased.

Table 1. Data Transport Packet Format

Start ID						
ID0	ID1	No. of Data Bytes	Data 1 CMD	Data 2 to Data 5	Data x (x = 6 to 255)	Checksum
0x07	0x0E	5 to 255	E, W, V, P, or R	h, u, m, l	xx	CS

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)
Protect	P (0x50)	ACK (0x06)	BEL (0x07)
Run (Jump to User Code)	R (0x52)	ACK (0x06)	BEL (0x07)

Table 3. Erase Flash/EE Memory Command

Start ID						
ID0	ID1	No. of Data Bytes	Data 1 CMD	Data 2 to Data 5	Data 6 (Pages)	Checksum
0x07	0x0E	6	E (0x45)	h, u, m, l	x pages (1 to 124)	CS

Table 4. Program Flash/EE Memory Command

Start ID						
ID0	ID1	No. of Data Bytes	Data 1 CMD	Data 2 to Data 5	Data x (x = 1 to 250)	Checksum
0x07	0x0E	5 + x (6 to 255)	W (0x57)	h, u, m, l	Data bytes	CS

Table 5. 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

Erase Command

The erase command allows the user to erase Flash/EE from a specific page determined by Data 2 to Data 5. The address is rounded down to the page start. 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 as a mass erase command, erasing the entire user code space and the Flash/EE protection. 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 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 should be aborted and the entire download sequence started again.

Verify Command

The verify command is almost identical to the write command (see Table 5). 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.

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.

Flash/EE Memory Protection Command

To use this command, a 3-step sequence must be followed:

1. Initiate the command. Type must be 0x00 and h, u, m, l can be any value.
2. Send the address of the group of pages to protect. Repeat this step for each group of pages. Type must be 0x0F.
3. Send the key in h, u, m, l; type must be 0x01. FEEADR takes the value of hu and FEEDAT takes the value of ml. If no keys are required, h, u, m, l must be 0xFFFFFFFF.

For example, to protect Page 0 to Page 7 against writing, set the read protection and use Key 0x12345678. The following commands must be sent:

- Start sequence:
  0x07 0x0E 0x06 0x50 0xXXXXXXXX 0x00 CS
- Protection:
  0x07 0x0E 0x06 0x50 0x00000000 0x0F CS
  (Page 0 to Page 3)
  0x07 0x0E 0x06 0x50 0x00000200 0x0F CS
  (Page 4 to Page 7)
  0x07 0x0E 0x06 0x50 0x0000F800 0x0F CS
  (read protection)
- Key and end of sequence:
  0x07 0x0E 0x06 0x50 0x12345678 0x01 CS

Note that the protection command is only available in Revision 0 and later versions of the loader. In Revision 0, FEEADR = ml and FEEDAT = ml. In later versions of the loader, FEEADR = hu.

This protocol does not allow the Flash/EE memory to be unprotected. To remove the protection, use a mass erase command.

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 start executing code.

Two types of remote run are implemented

- A software reset, with h, u, m, l = 0x1.
- A jump to user code, with h, u, m, l = 0x0.

Table 8 shows an example of a remote run or reset. Executing a software reset is recommended as it resets all peripherals.

Table 6. Verify Flash/EE Memory Command

Start ID						
ID0	**ID1**	**No. of Data Bytes**	**Data 1 CMD**	**Data 2 to Data 5**	**Data x (x = 1 to 250)**	**Checksum**
0x07	0x0E	5 + x (6 to 255)	V (0x56)	h, u, m, l	Modified data bytes	CS

Table 7. Flash/EE Memory Protection Command

Start ID						
ID0	**ID1**	**No. of Data Bytes**	**Data 1 CMD**	**Data 2 to Data 5**	**Data 6**	**Checksum**
0x07	0x0E	0x06	P (0x50)	h, u, m, l	Type	CS

Table 8. Remote Run Command

Packet ID					
ID0	**ID1**	**No. of Data Bytes**	**Data 1 CMD**	**Data 2 to Data 5**	**Checksum**
0x07	0x0E	0x05	R (0x52)	h, u, m, l = 0x1	0xA8

INTEL EXTENDED HEX FORMAT

d 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 appears 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 (10), the address of the first byte (0000), and the record type (00). The data bytes follow the record type. 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 are not found in a real object file):

:10 0000 00 FFFEFDFCFBFAF9F8F7F6F5F4F3F2F1F0 78

:05 0010 00 0102030405 DC

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). For example:

:00 0000 01 FF

Extended Segment Address Record

Record Type 02, the extended segment address record, defines Bit 4 through Bit 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 Bit 4 through Bit 19 of the segment base address (1000), and the 2-character checksum (FB). For example:

:02 0000 02 1000 FB

Start Segment Address Record

Record Type 03, the start segment address record, defines Bit 4 through Bit 19 of the execution start segment base address for the object file. For example:

:02 0000 03 0000 FB

Extended Linear Address Record

Record Type 04, the extended linear address record, defines Bit 16 through Bit 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 Bit 16 through Bit 31 of the destination address (FFFF), and the 2-character checksum (FC). For example:

:02 0000 04 FFFF FC

Start Linear Address Record

Record Type 05, the start linear address record, defines Bit 16 through Bit 31 of the execution start address for the object file. For example:

:02 0000 05 0000 F9

Sample Intel Hex Object File

Here is an example of an Intel Hex object file that contains the following records: extended linear address, extended segment address, data, and end.

```
:020000040108F1
:0200000212FFEB
:0401000090FFAA556D
:00000001FF
```

1. Determine the extended linear address offset for the data record (0108 in this example).
   :02 0000 04 0108 F1
2. Determine the extended segment address for the data record (12FF in this example).
   :02 0000 02 12FF EB
3. Determine the address offset for the data in the data record (0100 in this example).
   :04 0100 00 90FFAA55 6D
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

LIMITATIONS

Record Type 02, Record Type 03, Record Type 04, and Record Type 05 are not implemented. Unsupported records are ignored. Only the low 16 address bits are significant to access internal flash, therefore, it is safe to ignore the record types that change the high 16 bits.

NOTES