MicroConverter™ Technical Note - uC004
Understanding the Serial Download Protocol

1.0 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 refered to as **'serial download'**. Serial download capability allows developers to re-program 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 that manufacturers can upgrade system firmware in the field without having to swop out the device.

Any MicroConverter device can be configured for serial download mode via a specific pin configuration at power-on or application of the external reset signal. For the ADuC812, the *PSEN* input pin is pulled low through a resistor. If this condition is detected by the part on power-on or during application of a hard Reset input then 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 will communicate with any host machine to manage the download of data into its FLASH/EE memory spaces. It should be noted that serial download mode operates within the standard supply rating of the part (2.7V to 5.5V). Therefore, there is no requirement for a specific high programming voltage as this is also generated on-chip. A graphical example of the serial download system in operation is shown in figure 1 below.

As part of Analog Devices' QuickStart development tool-suite, a PC DOS executable program is provided (download.exe) which allows the user to download code from the PC (PC serial ports COM1,2...) to the ADuC812 MicroConverter. It should however be emphasized that any master host machine (PC, microcontroller, DSP or other) can download to the ADuC812 once the host machine adheres to the serial download protocols detailed in this technical note.

The objective of this technical note is therefore to outline 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.

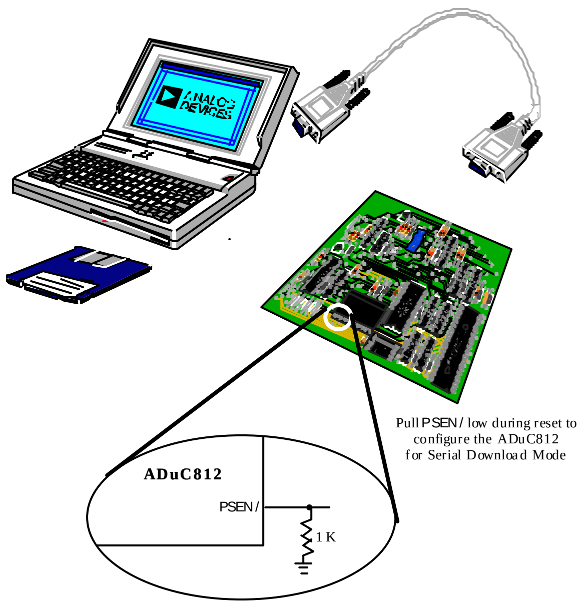

Pull PSEN/ low during reset to configure the ADuC812 for Serial Download Mode
ADuC812
PSEN/
1K

Figure 1. ADuC812 in Serial Download Mode

2.0 The ADuC812 Loader :

The serial download protocol on any MicroConverter part is defined by the on-chip loader routine. The ADuC812 on-chip loader has undergone 2 revisions, namely :

Loader Version 1 : is on all silicon with branded date code < 9933 (silicon shipped prior to August 1999)
Loader Version 2 : is on all silicon with branded date code >= 9933 (silicon shipped after August 1999)

The version 1 loader protocol supports download of the raw Intel Hex format file directly to the ADuC812 Flash/EE program memory space. The protocol as defined is straight forward, functional and is detailed in section 2.2 below.

The version 2 loader supports a more comprehensive protocol allowing serial download to the Flash/EE program space or the Flash/EE data space. These and more, additional features facilitate both a more flexible and more secure use of the in-circuit download capability on the ADuC812.

For users who wish to support both protocols in their end target system, section 3.0 below details a C-Source program which illustrates how this can be done. The 'download.c' source code accompanying this technical note is included in its compiled form (download.exe) as part of the Analog Devices MicroConverter QuickStart Development Tools. When run from the PC Dos command line, the program will communicate with any version of the loader and download code to the ADuC812 Flash/EE program memory space. This C-Source code could with very little modification be compiled for any host machine (microcontroller, DSP, etc...) required to download to the ADuC812 .

For the purposes of clarity in the description below, the term **'Host'** refers to the host machine (PC, microcontroller, DSP or other machine) attempting to download data to the MicroConverter. The term **'loader'** refers specifically to the on-chip serial downloader firmware resident on the MicroConverter.

2.1 The ADuC812 Version 2 Loader :

2.1.2 Running the Loader :

As mentioned previously, the loader on the ADuC812 is run by pulling the *PSEN* pin low through a resistor (typically 1KΩ pulldown) and power cycling the part or toggling the 'RESET' input pin on the part itself thus initiating a reset cycle. On reset or after a power cycle the loader will immediately send the following 25 byte ID data packet….

10 bytes	- product identifier	**"ADI<space> 812<space><space><space>"**
4 bytes	- firmware version number	**"V201"**
2 bytes	- line feed and carriage return	
2 bytes	- hardware configuration	
6 bytes	- Reserved for future use	
1 byte	- 2's complement checksum of the first 24 bytes	

2.1.2 The Physical Interface :

Once triggered, the loader configures the ADuC812 UART serial port to transmit/receive at 9600 baudrate, 8 data bits and no parity. The baudrate is derived indirectly from the ADuc812 master clock frequency i.e. 9600 baudrate will be set based on a master clock frequency of 11.0592MHz. If the master clock frequency is increased or decreased the baudrate will increase or decrease correspondingly i.e. if the master clock frequency is set to 1MHz then the loader will configure the UART baudrate at (1Mhz/11.0592MHz) or 868.055baud. The PC program download.exe, discussed in section 3 later allows the user to input the target master clock rate and thus reconfigure the PC baudrate to match that of the loader.

2.1.3 Interrogating the Loader :

The loader should be first interrogated to verify both that the loader is correctly present and that the loader firmware version number is as expected. This interrogation sequence is used for example in the download.c program outlined in section 3.0. In this code it allows the host decide what loader is present therefore which communications protocol to adopt in the subsequent code download sequence.

The loader is interrogated (at anytime) by sending the following data packet to the MicroConverter :

Interrogation Data Packet : <21 hex> <5Ahex> <00hex> <A6hex>

The MicroConverter loader responds immediately by sending its 25 byte ID data packet mentoned earlier and detailed again below…

25 byte loader ID Data Packet :

10 bytes	- product identifier	**"ADI<space> 812<space><space><space>"**
4 bytes	- firmware version number	**"V201"**
2 bytes	- line feed and carriage return	
2 bytes	- hardware configuration	
6 bytes	- Reserved for future use	
1 byte	- 2's complement checksum of the first 24 bytes	

2.1.4 Defining the Data Transport Packet Format :

Once the host has confirmed the presence of the loader the transfer of data can begin. The general communications data transport packet format is shown in figure 2 below.

Packet Start ID	No. of Data Bytes	Data 1 (Command Function)	Data 2	Data 3	Data *x* (x=25 max.)	Checksum
07 hex and 0E hex	1 – 25 Dec	C or A or W or E or U	-	-	-	No of DataBytes + Data 1 + Data 2 + . ======== ~(Checksum)

Figure 2. Data Transport Packet Format

Packet Start ID Field :

The first field is the 'Packet Start ID' field and contains two start characters (07hex and 0Ehex). 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 used to hold the total number of data bytes, the maximum number of data bytes allowed is 25 but can vary from 1 to 25 depending on the data packet being transmitted.

Command Function Field :

The 'Command Function' field describes the function of the data packet. One of 5 valid command functions are allowed. It should be noted that some functions (eg. Erase commands) require that no additional data be transmitted. The five command functions are described by one of five ASCII characters, 'C', 'A', 'W', 'E', or 'U'.

The loader routine will respond with an 'NAK' (07hex) as a negative response or with an 'ACK' (06hex) as positive response to the previous data packet communication. The full list of data packet command functions is shown in figure 3 below.

Command Functions	Command Byte In Data 1 Field	Loader Negative Response	Loader Positive Response
Erase Flash/EE Code Memory.	‘C’ (43 hex)	NAK (07hex)	ACK (06 hex)
Erase Flash/EE Code & Data Memory	‘A’ (41 hex)	NAK (07hex)	ACK (06 hex)
Write Flash/EE Code Block	‘W’ (57 hex)	NAK (07hex)	ACK (06 hex)
Write Flash/EE Data Block	‘E’ (45 hex)	NAK (07hex)	ACK (06 hex)
Jump to User code	‘U’ (55 hex)	NAK (07hex)	ACK (06 hex)

Figure 3. Data Packet Command Functions

Data Byte Fields 2 – 25 :
The data bytes to be downloaded are contained in these. This data must be stripped out of the standard Intel HEX 16 byte record format and re-assembled by the host as part of the above data format, before transmission to the loader.

Checksum Field :
The data packet checksum is written into this field. The checksum is calculated as the sum of all bytes in Data 1 to Data X fields, which is written in 2’s complement format into the checksum field.

2.1.5 Using the Data Packet Format to Download Data :

A program file can now be downloaded from the host to the MicroConverter loader using the data packet formats described above. Figure 4 shows a flowchart for a typical download sequence.

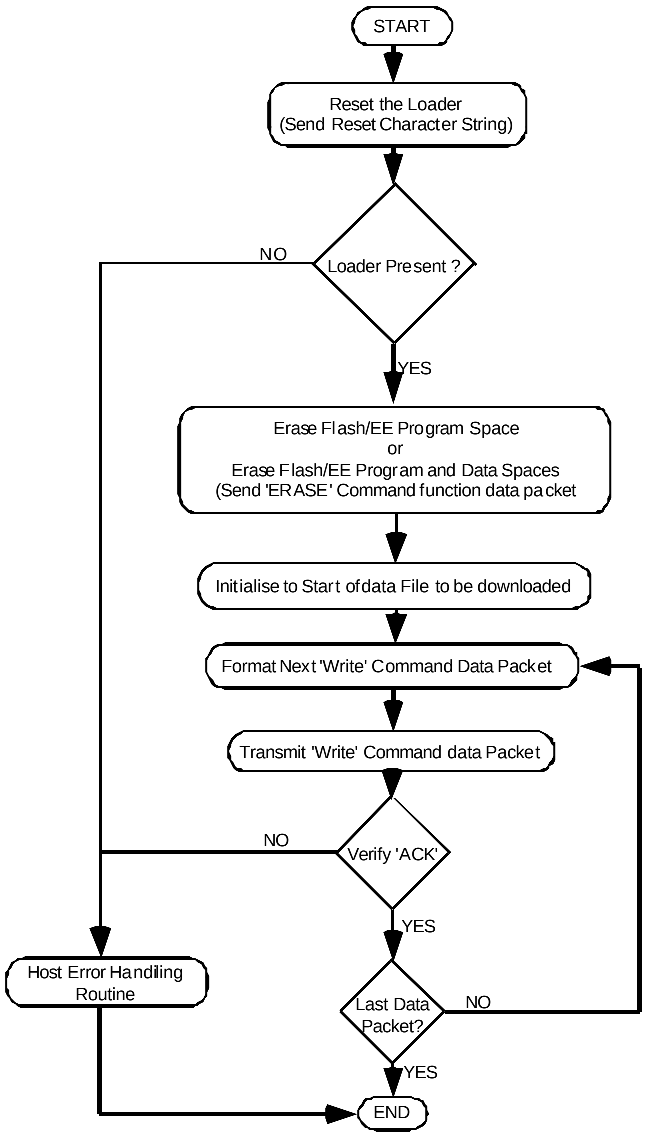

START
Reset the Loader
(Send Reset Character String)
Loader Present ?
NO
YES
Erase Flash/EE Program Space
or
Erase Flash/EE Program and Data Spaces
(Send 'ERASE' Command function data packet
Initialise to Start of data File to be downloaded
Format Next 'Write' Command Data Packet
Transmit 'Write' Command data Packet
Verify 'ACK'
NO
YES
Host Error Handling Routine
Last Data Packet?
NO
YES
END

Figure 4. Flowchart for a Typical Data Download Sequence

The following points should be noted :

a. If the host is downloading to the Flash/EE program memory or to the Flash/EE data memory then these memory spaces must be first erased using ‘C’ (Erase Program Memory) or ‘A’ (Erase Program and Data Memory) command bytes. The loader will respond with a ‘NAK’ if the host attempts to download to a memory space that has not been first erased.

b. The erase commands shown in Figure 3 above do not require any additional data in the data packet format. An example of a data packet initiating erasure of Flash/EE program and Data space is shown in figure 5 below.

Packet Start ID	**No. of Data Bytes**	**Data 1 (Command Function)**	**Checksum**
07 hex and 0E hex	01(hex)	'A' (41 hex)	BEhex Σ of No. of Data Bytes and Data Bytes 1 (2'sComp)

Figure 5. Erase Flash/EE Program and Data Space Data Packet Command Function.

c. All of the download data packet commands require a start address. The address is contained in three bytes immediately proceeding the command function data byte. The download data packets also contain the raw data to be downloaded. The first data byte is written by the loader to the address specified in the data packet format, the loader then increments the address and writes the next byte and so on until all data bytes in the packet have been written. An example of a data packet which will program 8 locations in the Flash/EE program space starting at address 0000H is shown in Figure 6 below.

Start ID	**No. of Data Bytes**	**Data 1 Write Cmd**	**Data 2 ADR U**	**Data 3 ADR M**	**Data 4 ADR L**	**Data 5 Prog 1**	**Data 6 Prog 2**	**Data 7 Prog 3**	**Data 8 Prog 4**	**Data 9 Prog 5**	**Data 10 Prog 6**	**Data 11 Prog 7**	**Data 12 Prog 8**	**Check Sum**
07 hex and 0E hex	0C (hex) 12 (Dec)	'W' (45 hex)	00 (hex)	00 (hex)	00 (hex)	00 (hex)	0C (hex)	0E (hex)	0C (hex)	0F (hex)	0E (hex)	4F (hex)	63 (hex)	BA (hex) Σ of No. of Data Bytes and Data Bytes 1 to12 (2'sComp)

Figure 6. Write Flash/EE Program Space Data Packet Command Function.

d. As can be seen from the ADuC812 datasheet, the 640Byte Flash/EE data space must be programmed in 4 byte pages (this 640Byte space is configured as 160 pages). An example of a data packet which will program page 5 in the Flash/EE data space is shown in figure 7 below.

Start ID	**No. of Data Bytes**	**Data 1 Write Cmd**	**Data 2 ADR U**	**Data 3 ADR M**	**Data 4 ADR L**	**Data 5 Prog 1**	**Data 6 Prog 2**	**Data 7 Prog 3**	**Data 8 Prog 4**	**Check sum**
07 hex and 0E hex	08 (hex) 8 (Dec)	‘E‘ (45 hex)	00 (hex)	00 (hex)	05 (hex)	0A (hex)	0B (hex)	0C (hex)	0D (hex)	80 (hex) Σ of No. of Data Bytes and Data1 to Data 8 (2’sComp)

Figure 7. Write Flash/EE Data Space Data Packet Command Function.

e. As mentioned earlier, the MicroConverter loader routine will respond with an ‘NAK’ (07hex) as a negative response or with an ‘ACK’ (06hex) as positive response to the previous data packet communication. A ‘NAK’ will be transmitted by the loader under the following conditions :
- ➢ Incorrect data packet format on verification of the checksum byte.
- ➢ Loader fails to verify that data has been programmed correctly.
- ➢ Loader is requested to program data to a location that has not been previously erased.

f. The two’s complement checksum is calculated from the summation of the hex values in the No. of Bytes field and the hex values in the Data 1 to 25 fields. The checksum is the two’s complement value of this summation. This can also be expressed mathematically as :

$$\textbf{Checksum} = \textbf{100hex} - \left(\textbf{Data Byte}_{\textbf{(NO. OF BYTES FIELD)}} + \sum_{N=1}^{25} \textbf{Data Byte}_N\right)$$

2.1.6 Telling the Loader to 'RUN' code :

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. Figure 8 below shows an example of a 'jump to user code' at address 00Hex data packet.

Packet Start ID	No. of Data Bytes	Data 1 (Command Function)	Data 2 ADR U	Data 3 ADR M	Data 4 ADR L	Checksum
07 hex and 0E hex	04(hex)	'U' (55 hex)	00 (hex)	00 (hex)	00 (hex)	A7hex Σ of No. of Data Bytes and Data Bytes 1 (2'sComp)

Figure 8. 'Run Program Code' Data Packet Command Function.

2.3 The ADuC812 Version 1 Loader :

2.3.1 Running the Loader :

As with the version 2 loader, the version 1 loader on the ADuC812 is initiated by pulling the *PSEN* pin low through a resistor (typically 1KΩ pulldown) and power cycling the part or toggling the 'RESET' input pin on the part itself thus triggering a reset cycle. On reset or after a power cycle the loader will first erase the Flash/EE program and data memory and immediately send the following 11 byte ID data character string....

11 byte loader ID Data :

8 bytes	- product identifier	**"ADuC812<space>"**
3 bytes	- firmware version number	**"krl"**

2.3.2 The Physical Interface :

Once triggered, the loader configures the ADuC812 UART serial port to transmit/receive in the following mode, 9600 baudrate, 8 data bits, no parity. The baudrate is derived indirectly from the ADuc812 master clock frequency i.e. 9600 baudrate will be set based on a master clock frequency of 11.0592MHz. If the master clock frequency is increased or decreased the corresponding baudrate will increase or decrease correspondingly i.e. if the master clock frequency is set to 1MHz then the loader will configure the UART baudrate at (1MHZ/11.0592MHz) or 868.055baud. The PC program download.exe, discussed in section 3 later allows the user to input the target master clock rate and thus reconfigure the PC baudrate to match that of the loader.

2.3.3 Interrogating the Loader :

The loader should be first interrogated to verify both that the loader is correctly present and that the loader firmware version number is as expected. This interrogation sequence is used for example in the download.c program outlined in section 3.0. In this code it allows the host decide what loader is present therefore which communications protocol to adopt in the subsequent code download sequence.

The loader is interrogated (at anytime) by sending the following character to the MicroConverter :

Interrogation Character : <21 hex> (ASCII '!')

The MicroConverter loader responds immediately by sending its 11 byte ID data character string mentoned earlier and detailed again below...

11 byte loader ID Data :

8 bytes	- product identifier	**"ADuC812<space>"**
3 bytes	- firmware version number	**"krl"**

2.3.4 Downloading Code to the version 1 loader :

Unlike the version 2 loader described previously, the version1 loader supports direct download to Flash/EE program space only. The host must transmit an un-edited version of the standard Intel Hex format program file, the loader expects to see this transmitted directly as part of the download sequence.

The version 1 loader automatically erases Flash/EE program and data spaces as soon as the loader is enabled and before transmitting the 11 byte ID data character string (see section 2.3.1 above). It should also be noted that the loader does not support data download to the Flash/EE data memory space.

Once the loader has been interrogated as described in section 2.3.3 above , the loader will then wait for the transmission of an Intel Standard Hex file which it will program into the Flash/EE program memory space. It receives this file on a per record basis, expecting the host to adhere to the following communications protocol. Note the format of the standard Intel Hex file is described in section 2.3.6 below.

- The loader recognizes a record by the start character ':' and it is important to note that it will use the various bytes that precede the record such as address, number of bytes in record etc. as well as the checksum at the end of the standard record file format. Therefore it is important to that the host transmits the Intel hex file 'as is'.

- At the end of each record, the micro will transmit an ACK (everything OK) or a NACK (everything not OK). The ACK is 06H and NACK is 15H i.e industry standard UART ACK and NACK. So the user host should interpret these appropriately i.e on ACK move onto transmit the next record on NACK report an error message to the host. On an error the micro waits for the next ':' so the host can decide to stop on an error or try to re-transmit the record.

2.3.5 Running the Code :

At the end of the file, the user host can force the code to execute by sending a start address command character string. A start address can be sent by first sending ';' character and follow it by a 4- byte start address. It is important to point out here that in a real application if the host did want to download and follow this by a run from start address sequence, then this start address should be FF00hex . (The start address FF00hex ensures that a resident power-on-configuration routine runs to correctly calibrate the ADC, Internal Voltage Reference before normal user code runs from address 0000Hex.)

If you do not follow the download of the Intel Hex file with a ';' then the only way to force a start of code execution would be to remove the $\overline{PSEN}$ pulldown and force a reset or a new power cycle.

2.3.6 Standard Intel Hex Format :

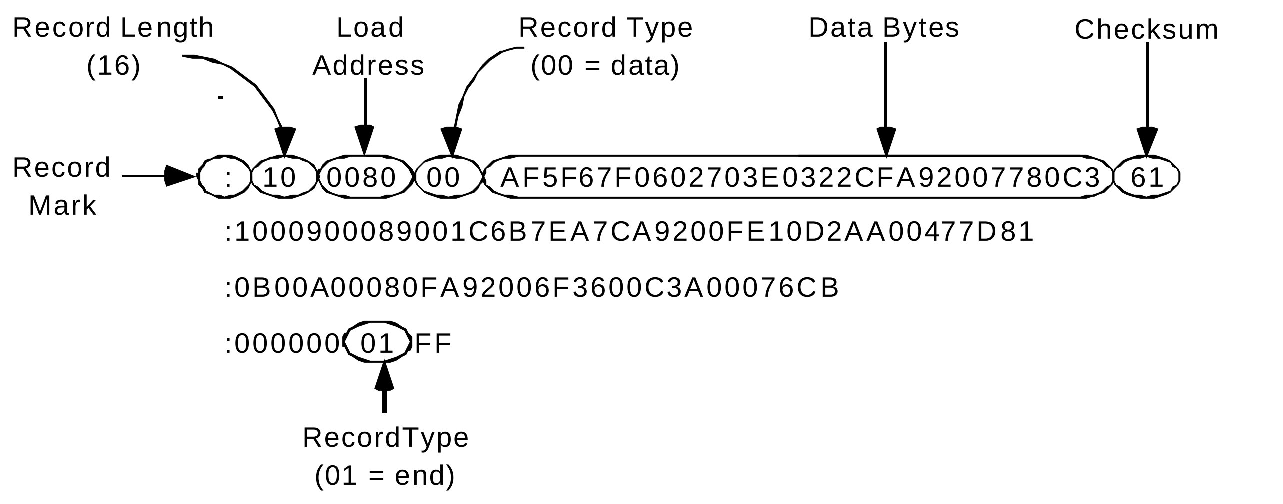
This section describes the Intel Hex standard file format expected by the revision 2 loader protocol. Intel Hexadecimal format or Intel hex format is a standard for storing machine language in displayable or printable format.

A standard Intel hex file is generated by assembling your MicroConverter (8051 compatible source code) program. An 8051 compatible Assembler (Metalink 2-pass assembler) is available as part of the QuickStart development System or as a free download executable from the MicroConverter web-site at www.analog.com/microconverter.

An Intel hex file is a series of lines or 'hex records' containing the following fields…

Field	Bytes	Description
Record Mark	1	":" indicates the start of record
Record Length	2	number of data bytes in record
Load Address	4	starting address for data bytes
Record Type	2	00 = data record, 01 = end record
Data Bytes	0 – 16	data
Checksum	2	Sum of all bytes in record + checksum = 0

These fields are shown more explicitly in figure 9 below which illustrates a typical example print-out from an Intel hex file.


Record Length
(16)
Load
Address
Record Type
(00 = data)
Data Bytes
Checksum
Record
Mark
: 10 0080 00 AF5F67F0602703E0322CFA92007780C3 61
:1000900089001C6B7EA7CA9200FE10D2AA00477D81
:0B00A00080FA92006F3600C3A00076CB
:000000 01 FF
RecordType
(01 = end)

Note : Spaces are included in the first and last lines above for illustration purposes only.

Figure 9. Intel Hex Format

3.0 C-Source Code Supporting both Protocols :

This section describes in general terms a C-Source, PC download program that is available on the MicroConverter Web-Site (download.c). While the download.c program can be compiled for the PC and run from DOS, it is also intended that this C-Source program could be compiled with relatively few changes for any target host system thus facilitating an embedded downloader.

The program is compatible with both versions of the loader and is intended as an example of how to communicate from any host machine to the on-chip loader firmware. Embedded in the program are the routines that strip the data from the input Intel Hex file format (readBlockFromFile , readRecordFromFile) and the routines that build the correctly formatted data packets for transmission to the loader (sendLoaderPacket).

The program also includes some additional user friendly features/options that can be enabled from the dos command line at run time, these include ...

3.1 DOS command line syntax :

DOWNLOAD filename.hex /c:n /f:nn /d /r

Where :

'filname.hex' -	is the name of the Intel standard hex file to be downloaded. A filename must be specified on the command line
/C:n -	select the com port (default is 1)
/F:n.n -	select the target crystal freq in MHz (default 11.0592)
/D -	do not erase Flash/EE data space (supported on firmware rev 2.0 or greater)
/R -	run the program from the beginning (FF00 hex)
/R:xxxx -	run the program from the specified start address

Some Examples

DOWNLOAD test1.hex /c:1 /f:16 /d /r
Downloads a program called test1.hex via the PC COM1 port to a target system where the MicroConverter is running at a 16MHz clock frequency. The Flash/EE data memory space is not erased prior to the download (the Flash/EE program space is erased prior to the download by default). The program is automatically run from address FF00hex as soon as the download sequence is complete.

DOWNLOAD test2.hex /c:2
Downloads a program called test2.hex via the PC COM2 port to a target system where the MicroConverter is running at an 11.0592MHz clock frequency. The Flash/EE data memory space is erased prior to the download (the Flash/EE program space is erased prior to the down by default). The program is not run subsequent to the download sequence and therefore a power cycle or a hard reset will be require (without the $\overline{PSEN}$ pulldown) to initiate program execution.

DOWNLOAD test3.hex /c:1 /r:0F00
Downloads a program called test2.hex via the PC COM1 port to a target system where the MicroConverter is running at an 11.0592MHz clock frequency. The Flash/EE data memory space is erased prior to the download (the Flash/EE program space is erased prior to the down by default). The program is automatically run from address 0F00hex as soon as the download sequence is complete (A user may want to use the 'run from address' option in development to avoid repeating some time consuming configuration routines that would delay code debug).

3.2 Program Flow Summary :

The download.c program flow can be outlined as follows :

Reset sequence:
(Note:
The version 1 loader will automatically erase Flash/EE Program and Data spaces prior to transmitting its ID character string. The version 2 loader will wait until it recieves a specific erase Command byte (see section XXX before choosing to erase Flash/EE program space alone or the Flash/EE program and Data spaces.)

The version loader uses "!" as the reset sequence and the target responds with:
"ADuC812 krl"

The new loader uses a four byte reset command:
"!Z"<0x00><checksum byte>
The target responds with a 25 byte part ID string, begining with "ADI" and ending with a <checksum> byte. (see section XXX)

In order to handle both loader revisions, this downloader program first sends the "!", waits for the old response. If it does not get one, it sends the "Z"<checksum><0x00> and waits for the new response, thereby allowing it to identify which loader revision is present and which protocol to adopt in the subsequent download sequence.

Download Sequence:
The old loader firmware can decode individual intel hex records itself. The packets are read from the file, and sent 'as is' to the target. The target responds to each packet with a ACK or NACK to indicate success or failure respectivly.

The new loader needs the data to be encoded as "write code memory" commands (see section XXX), which are sent as loader command packets:

Run sequence:
The old loader run command was ";ADDR". ADDR was in HEX text format, e.g. ";0000" to run from zero. The target would respond with an ACK if successful.

The new loader uses a "run user code" command, sent as a loader command packet.

A flowchart illustrating the detailed program flow is shown in figure 10 overleaf.

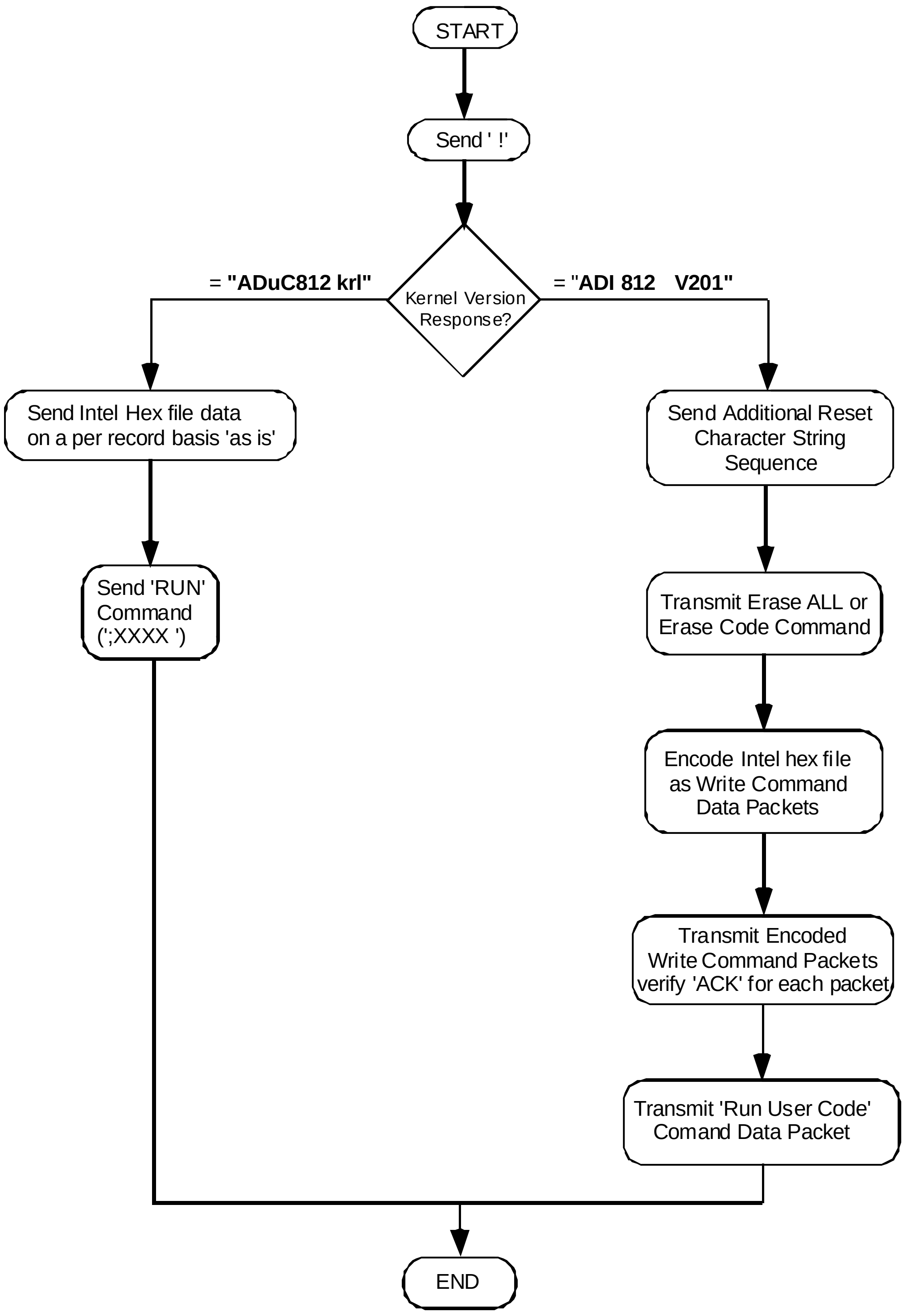

START
Send ' !'
Kernel Version Response?
= "ADuC812 krl"
= "ADI 812 V201"
Send Intel Hex file data on a per record basis 'as is'
Send 'RUN' Command (';XXXX ')
Send Additional Reset Character String Sequence
Transmit Erase ALL or Erase Code Command
Encode Intel hex file as Write Command Data Packets
Transmit Encoded Write Command Packets verify 'ACK' for each packet
Transmit 'Run User Code' Comand Data Packet
END

Figure 10. Download.C Code Flowchart