# MicroConverter® Technical Note - uC007
# User Download (ULOAD) Mode

## 1.0 INTRODUCTION :

The ADuC83X family ("big memory family") all integrate a large program memory space, with 62kBytes of flash/EE program memory available to the user. As with the standard MicroConverter products (ADuC81X and ADuC82X) the full program memory space can be programmed in circuit direct from a 2kByte download program, residing on the chip at address F800h→ FFFFh via a defined protocol (see uC004) using the UART serial port.

Now on the ADuC83X family the user can choose to program the entire user code space (62kBytes) via the 2kByte download program as normal, or, use their own custom defined protocol located in the upper 6kBytes of user program memory (E000H → F7FFH) to reprogram the lower 56kBytes of user program memory using any of the on chip peripherals that they wish. This is known as **User Download Mode** or **ULOAD** mode for short. In this case the 6kByte program is known as the '**bootloader**' as it programs the lower 56kBytes with the **user code**, allowing user code to be easily upgraded as necessary.

Typically, there are two different approaches to bootloaders.

1. Embedded Programmers (the whole of the bootloader is present)
2. Microprogrammers (only a tiny bootloader is present to download the bootloader itself into external RAM)

each approach can be implemented using either

1. ROM based Bootloader
2. Flash Based Bootloader

Each of these can be typically compared for **robustness**, **cost** and **flexibility** with each approach having its own advantages and disadvantages. Embedded programmers require larger non-volatile memory and lack flexibility; the microprogrammer is more difficult to implement and requires that a external RAM is available to run code. In terms of their implementations; ROM based programmers are inflexible and add cost to the system and finally flash based programmers can cripple the product if power failures occur during reprogramming.

Our bootloader tries to provide a simple bootloader that is easy to implement, flexible, very robust and cost effective. To do this we have implemented a bootloader that is a somewhere between a ROM based bootloader and a flash based bootloader. The bootloader itself is physically flash/EE memory integrated in the same block as the rest of the flash/EE program memory. However what differentiates it from ordinary flash based bootloaders is that while running user code the user cannot erase, or reprogram the bootloader code. Hence it appears as ROM to user code.

To program (or reprogram) the bootloader requires that the user code jumps to the kernel program (hardware reset with $\overline{\text{PSEN}}$ pulled low) and follows the serial download protocol as defined in technical note uC004. If an error occurs while programming (or reprogramming) the bootloader, then code execution will begin again in the kernel program allowing the bootloader download to be tried again.

Section 1 of this Technical Note discusses:

- The Flash/EE Program Memory map
- How to implement a user defined bootloader using ULOAD mode
- How fast can the 56kBytes of flash/EE program memory be programmed in ULOAD mode?
- How to implement a safe user protocol when defining your own protocol

Section 2 of this technical note contains a code example with a code description and a flowchart.

## 1.1 FLASH/EE PROGRAM MEMORY MAP:

As with the standard products there are two distinct memory blocks on the ADuC83X family, a flash/EE data memory and a flash/EE program memory. The difference between the standard products and the big memory products is that the on big memory products both the flash/EE data memory and the flash/EE program memory are about 8 times bigger, giving a 64Bytes of flash/EE program memory and 4kBytes of flash/EE data memory.

As shown in figure 1 the **4kBytes of flash/EE data memory** is configured as 1024 pages of 4 bytes. EADRH and EADRL contain the **page address** (0000h<EADRH/L<0400h) of the flash/EE data memory and is programmed as described in section 1.2 for NORMAL mode. The erasing, programming and reading of this space is identical to that of the ADuC81X and ADuC82X products except the page address now requires the inclusion of the EADRH SFR.

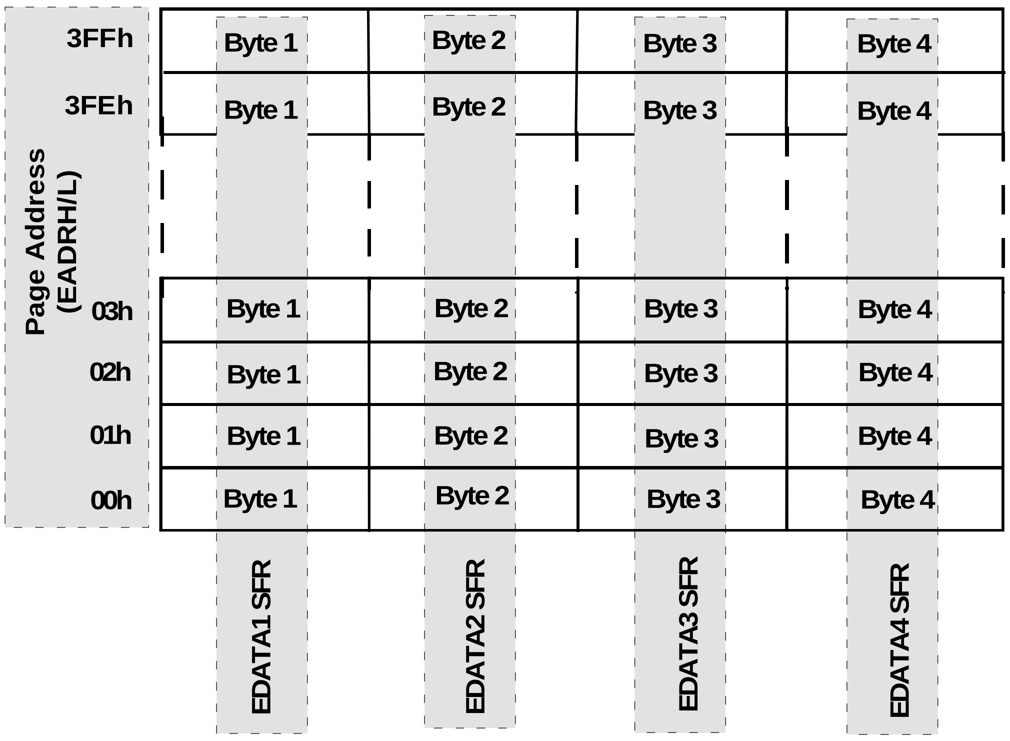

*Figure 1: 4kBytes of flash/EE data memory programming configuration*

As can be seen from figure 2 below the **64kBytes of flash/EE program memory** is divided into two sections; a 2kByte kernel download/debug/emulation program and 62kBytes of user code. Many users of large programs (often C source) will use the flash/EE program memory in this way.

In figure 3 we notice that the 62kBytes of user flash/EE program memory can be sub-divided into two distinct sections if required. The upper 6kBytes of flash/EE program memory can be configured as a user bootloader, and the lower 56kBytes of flash/EE program memory can be used to run the user program. Alternatively some or all of this 56kBytes can also be used as data memory if required by the user. This is often useful in data logger applications.

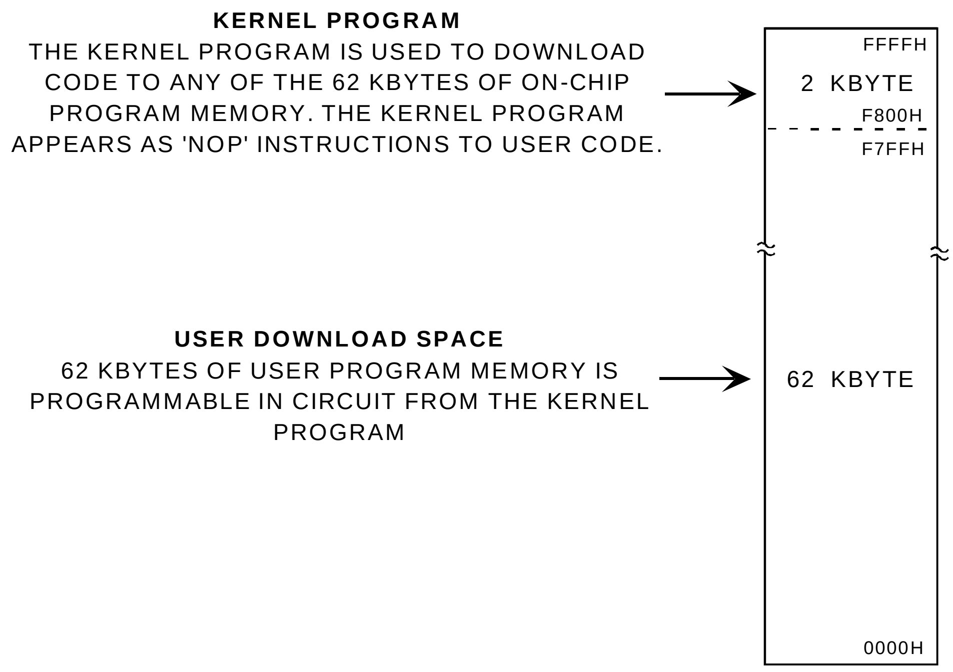
KERNEL PROGRAM
THE KERNEL PROGRAM IS USED TO DOWNLOAD CODE TO ANY OF THE 62 KBYTES OF ON-CHIP PROGRAM MEMORY. THE KERNEL PROGRAM APPEARS AS 'NOP' INSTRUCTIONS TO USER CODE.
USER DOWNLOAD SPACE
62 KBYTES OF USER PROGRAM MEMORY IS PROGRAMMABLE IN CIRCUIT FROM THE KERNEL PROGRAM
FFFFH
2 KBYTE
F800H
F7FFH
62 KBYTE
0000H

*Figure 2: Flash/EE Program Memory Configuration*

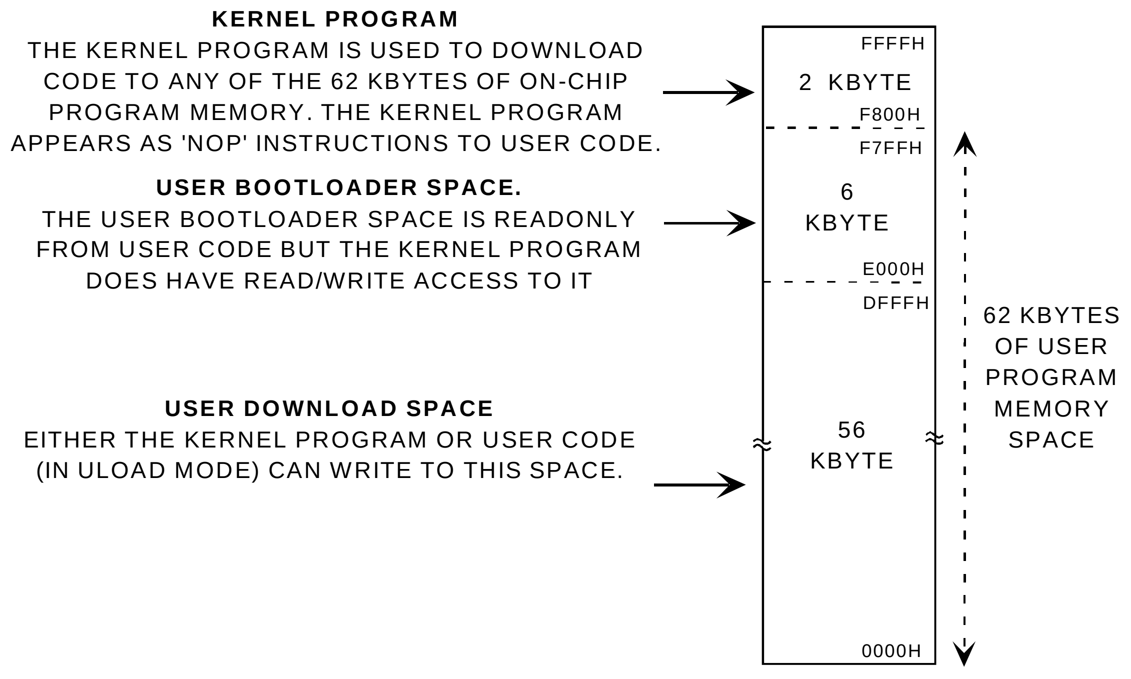
KERNEL PROGRAM
THE KERNEL PROGRAM IS USED TO DOWNLOAD CODE TO ANY OF THE 62 KBYTES OF ON-CHIP PROGRAM MEMORY. THE KERNEL PROGRAM APPEARS AS 'NOP' INSTRUCTIONS TO USER CODE.
USER BOOTLOADER SPACE.
THE USER BOOTLOADER SPACE IS READONLY FROM USER CODE BUT THE KERNEL PROGRAM DOES HAVE READ/WRITE ACCESS TO IT
USER DOWNLOAD SPACE
EITHER THE KERNEL PROGRAM OR USER CODE (IN ULOAD MODE) CAN WRITE TO THIS SPACE.
FFFFH
2 KBYTE
F800H
F7FFH
6 KBYTE
E000H
DFFFH
56 KBYTE
0000H
62 KBYTES OF USER PROGRAM MEMORY SPACE

*Figure 3: Flash/EE Program Memory Configuration in Bootloader Mode*

**2kBYTE KERNEL MEMORY (F800h → FFFFh)**
Every part that leaves the factory contains a kernel program that has three distinct functions.

(1) After power up or reset a small piece of code downloads **factory-calibrated coefficients** to the various calibrated peripherals (ADC, temperature sensor, current sources, bandgap reference, etc).
(2) **downloads** code to any of the 62kBytes of user code space
(3) a **debugger/emulator** program allows in circuit non-intrusive emulation of the MicroConverter.

The 2 KBytes kernel program will appear as NOP instructions to user code. The only way to run the kernel program is reset the part with the $\overline{\text{PSEN}}$ pin pulled low.

**6kBYTE USER BOOTLOADER SPACE (E000h → F7FFh)**
Since the bootloader space is only programmable from the 2kByte kernel program or via parallel programming it can therefore not be erased or reprogrammed while running user code, even if the user code becomes erroneous. Since the bootloader is flash based it will not get erased or reprogrammed during power failures. Hence this space can act like a ROM bootloader.

**NOTE**: If using the flash/EE program memory in bootloader mode then it is strongly recommended powering up at the start of the bootloader program (E000h) after reset instead of 0000h (see section 1.4 'Implementing a safe Bootloader Protocol'). For larger programs then the user can use this space as normal flash/EE program memory.

**56kBYTES PROGRAM MEMORY (0000h → DFFFh)**
This space is erasable and re-programmable from user code if the user enters ULOAD Mode. The user can use this space for data only, for code only or indeed for code and data. When the flash memory control SFR (ECON) is put into ULOAD mode then it is possible to:

- erase the full 56kBytes of flash/EE program memory
- erase the flash/EE program memory space page by page (each page is 64 bytes)
- byte program the code space (the page must be pre-erased)
- page program (256 bytes at a time) the code space (all 4 pages must be pre-erased)

## 1.2 HOW TO IMPLEMENT A USER DEFINED BOOTLOADER USING ULOAD MODE:

Bootloader code can be downloaded into the upper 6kBytes of program memory by

- serially downloading to the 6kByte (and the other 56kBytes if required) of bootloader space via the on-chip serial download protocol.
- Parallel programming the 6kByte (and the other 56kBytes if required) of bootloader space via a parallel programmer as documented in the datasheet.

Once the bootloader has been downloaded then the code can be upgraded in circuit via the users custom bootloader which might use the SPI, UART serial ports, a software I2C routine or indeed using the parallel ports or any other protocol that the user desires.

It should be noted that there is nothing in hardware to prevent the user from erasing the lower 56kBytes of code even if executing code from the 56kBytes. The upper 6kBytes of Flash/EE Program memory however is not changeable in user code and can hence not be erased from user code. It can only be erased and reprogrammed from the kernel program.

By default the flash memory control SFR (ECON) points to the flash/EE data memory. This means that read, write and erase commands programmed to the ECON SFR will act on the flash/EE data memory and not the 56kBytes of flash/EE program memory. To reprogram the 56kBytes of flash/EE program memory the user must enter ULOAD mode. The user can enter ULOAD mode by executing the following instruction.

```
            MOV   ECON, #0F0h ; Enter ULOAD mode
```

To point back to the flash/EE data memory the user must first exit ULOAD mode. ULOAD mode is exited by executing the following instruction.

```
            MOV   ECON, #0Fh  ; Exit ULOAD mode
```

The various commands available to the user during NORMAL mode and ULOAD mode are described in Table 1 below.

| ECON VALUE | COMMAND DESRIPTION (NORMAL MODE) | COMMAND DESCRIPTION (ULOAD MODE) |
|---|---|---|
| **01H READ** | Results in the 4 bytes in the flash/EE data memory, addressed by the **page address EADRH/L,** being read into EDATA 1-4. | Not implemented<br>Use the MOVC command |
| **02H WRITE PAGE** | Results in the 4 bytes in EDATA 1-4 being written to the flash/EE data memory, addressed by the **page address EADRH/L.**<br>**Note:** The 4 bytes in the page being addressed must be pre-erased | Results in the bytes 0-255 of internal XRAM being written to the **256 bytes** of flash/EE program memory addressed by the **page address EADRH**. (0 ≤ EADRH < E0h)<br>Note: The 256 bytes in the page being erased must be pre-erased. |
| **03H RESERVED** | Reserved | Reserved |
| **04H VERIFY PAGE** | Verifies if the data in EDATA1-4 is contained in page address given by EADRH/L. A subsequent read of the ECON SFR will result in a 0 being read if the verification is valid, or a nonzero value being read to indicate an invalid verification. | Not implemented.<br>Use the MOVC and MOVX commands (and dual data pointers) to verify the flash/EE program memory WRITE command. (see VERIFYULOADPROG function in code example) |

| | | |
|---|---|---|
| **05H ERASE PAGE** | Results in the Erase of the **4 byte page** of flash/EE data memory addressed by the **page address EADRH/L** | Results in the **64 byte page** of flash/EE program memory, addressed by the **byte address EADRH/L** being erased.<br>EADRL can point to any of the 64 locations within the page. A new page starts whenever EADRL is equal to 00h, 40h, 80h, or C0h. |
| **06H ERASE BLOCK** | Results in the Erase of the **entire 4kBytes of flash/EE data memory** | Results in the erase of the **entire 56 kBytes of ULOAD** flash/EE program memory. |
| **81H READBYTE** | Results in the byte, in flash/EE data memory addressed by the **byte address EADRH/L**, being read into EDATA1.<br>(0000h<=EADRH/L<=0FFFh) | Results in the byte, in flash/EE data memory addressed by the **byte address EADRH/L**, being read into EDATA1.<br>(0000h ≤ EADRH/L ≤ F7FFh) |
| **82H WRITEBYTE** | Results in the byte in EDATA1 being written into flash/EE data memory, at the **byte address EADRH/L.**<br>(0000h<=EADRH/L<=0FFFh) | Results in the byte in EDATA1 being written into flash/EE program memory, addressed by the **byte address EADRH/L**.<br>(0000h ≤ EADRH/L ≤ F7FFh) |
| **0FH<br>EXIT ULOAD MODE** | Leaves the ECON instructions operate on the **flash/EE data memory**. | **Enters NORMAL MODE** by changing the ECON instruction so that it operates on the **flash/EE program memory**. |
| **F0H<br>ENTER ULOAD MODE** | **Enters ULOAD MODE** by changing the ECON instruction so that it operates on the **flash/EE program memory**. | Leaves the ECON instructions operate on the **program/EE data memory**. |

*Table 1: Flash Memory Commands in NORMAL and ULOAD mode*

## 1.3 HOW FAST CAN THE 56KBYTES OF FLASH/EE PROGRAM MEMORY BE PROGRAMMED IN ULOAD MODE?

There are four time consuming tasks which must occur during a download.

1) Reception of the 56kBytes of data
2) Programming of the 56kBytes of flash/EE program memory
3) Verification of programming of the 56kBytes of flash/EE program memory

**Note:** The flash/EE program memory is **pre-erased** in the one extended instruction that takes 2ms in duration. This can be ignored in the context of about a 5s download.

(1) The first thing to occur is that the data is to be sent to the microcontroller. Lets assume that the data is downloaded in packets as follows.

<'D'><PAGEADD><DATA0->255><CS>

Therefore for every 256 bytes to be downloaded 259 bytes must be sent to the microcontroller core. Lets assume that these bytes are sent via SPI at the quickest bitrate that the core can keep up (assume that the microcontroller core is in slave mode). The loop used to receive the data (slowest part of the reception) might be something like this.

```
LOOP:
     JNB  ISPI, $              ;(24) wait for next reception
     CLR  ISPI                 ;(12)
     MOV  A, SPIDAT            ;(12)
     MOVX @DPTR, A             ;(24) DPTR in auto inc mode
     ADD  A, R1                ;(12) add to Check Sum
     MOV  R1, A                ;(12)
     DJNZ R0, LOOP             ;(24)
```

Therefore this loop requires that data cannot be received any quicker than every 120 core cycles. At 12.58MHz 120 core cycles takes 9.54µs. To calculate the minimum download time then lets assume that data is transmitted via SPI at this frequency. Therefore the 259 bytes will be received in **2.47ms**.

(2) The appropriate page of flash/EE program memory can be easily programmed in one instruction. However this instruction takes **15.5ms**.

(3) Next the code in flash/EE program memory should be verified against the value in internal XRAM. The following loop verifies every address.

```
VERIFYDOWNLOADLOOP:
     MOV  EADRL, A             ;(12)
     MOV  ECON, #81H           ;(24) read byte
     MOVX A, @DPTR             ;(24) DPTR in auto inc mode
     CJNE A, EDATA1, ERROR     ;(24)
     MOV  A, DPL               ;(12)
     JNZ  VERIFYDOWNLOADLOOP   ;(24)
```

This verification loop takes 120 core cycles per byte. Hence to verify the 256 bytes in the page takes **2.45ms**.

Therefore in total, the reception, programming and verification of every page in flash/EE program memory takes 20.5ms. Hence to receive, program and verify the 224 pages takes 4.6s.

**Hence it is possible to program the 56kBytes of flash/EE program memory in under 5s.**

## 1.4) IMPLEMENTING A SAFE BOOTLOADER PROTOCOL

An ideal bootloader allows the user to upgrade user code at any time, over any user-defined interface, without having to change any hardware on the circuit board. This allows a quick and easy way of upgrading the code in the field. The bootloader should remain robust even over the most unlikely conditions, e.g. erroneous code execution or power failures during reprogramming.

To provide a robust bootloader then there are two main points that should be observed.

1) ***What happens if a power failure occurs during programming, causing garbage data to be written into the code space?***
On any flash/EE based product it is possible that if a power failure occurs during programming that an erroneous infinite loop may exist at the start of the code space leaving the part unable to jump to the bootloader. Even worse for microcontrollers controlling dangerous equipment the equipment may end up executing erroneous code causing the equipment to behave dangerously after power has been restored.

   To prevent both of these unwanted phenomenon the ADUC83X family of products allows the user to start code execution from 0000h or E000h after a reset. It is recommended to all users using bootloaders to start execution from E000h. Execution from E000h ensures correct program execution after all conditions.
   NOTE: If there was a problem downloading the original bootloader code then the hardware ($\overline{\text{PSEN}}$ is still pulled low) is still set up correctly to reprogram this code again once the power is restored. The user code will not have been executing User Code until the download was completed correctly.

   To allow the user code to execute and not have to wait for a command "Run Use Code" from another peripheral it is desirable to easily start user code execution. To do this safely it is recommended to include something similar to the following at the start of the bootloader program.

```
READ FLASH/EE DATA MEMORY PAGE 0
IF EDATA1.0 = 1
      EXECUTE BOOTLOADER CODE
ELSE IF EDATA1.0=0
      EXECUTE USER CODE
```

   Now the first time that the bootloader executes EDATA1=FFh (if user erased both flash/EE data and program memory during download). Since, the LSB of BYTE1 on PAGE0 of the flash/EE data memory (which we will call BOOTEN (bootloader enable) from now on) is set then we will execute the bootloader code. If the user code is downloaded with the bootloader for the very first time then the user should also download to the flash/EE data memory ensuring that the BOOTEN bit is cleared (this will ensure that user code at 0000H is run after reset).

   In the case of the bootloader code only being downloaded code execution will begin in the bootloader after reset. User code can then be downloaded via the user's protocol defined in the bootloader. Once the code has been satisfactorily downloaded the BOOTEN bit should be cleared. This ensures that code execution will always begin in user code after a reset/power cycle in future. The BOOTEN bit should be set before any new download starts as discussed below.

2) ***Can erroneous code execution cause the corruption (erase/reprogram) of the flash memory***
While running erroneous code it is possible to execute a jump instruction to any part of user code. This means that any segment of code can be executed at any time during an erroneous condition.

   To prevent erroneous code execution executing an erase or reprogram or the flash/EE program memory on the ADuC83X family of products we have included a double instruction sequence to reprogram the flash/EE program memory. Firstly the user must enter ULOAD mode (point to the flash/EE program memory), and then the user must execute the erase/reprogram flash/EE program memory instruction. It is up to the user to include a check between these commands.

An example of a good check might be for the programming command to issue a “Ready for New Download” command when it wants to upgrade its code. Once the user program receives this command it sets the BOOTEN bit and vectors to the bootloader which checks if the BOOTEN bit is set. If it is set then the bootloader program will enter ULOAD mode. The BOOTEN bit is set to indicate that a download is in progress and if a power failure is to occur that we should attempt to download new code again

Note: Before we vector to the bootloader program we should disable all interrupts by clearing the EA bit in the IE register. Alternatively, as in the case of the code example explained in section 2, we can reset the part via the watchdog timer. Code will resume at E000h after reset (if programmed correctly) with the EA bit (and all other SFRs) cleared.

At this stage the bootloader waits to receive a command to “Erase Flash/EE program memory” or to “Program Flash/EE program memory”. It is only after receiving the commands correctly that the program should erase or reprogram the flash/EE program memory.

Again once the download is satisfactorily completed then the BOOTEN bit should be cleared again to allow future resets to execute user code and not the bootloader program.

Example: To reprogram the code space then the user might follow the following program flow:

Step 1: Download Bootloader
- ⇨ Download Bootloader Via WSD
- ⇨ Enable Code execution from E000H (see uC004)

Step 2: Receive Upgrade Code Command in user code
- ⇨ Set BOOTEN to indicate wish to start a new download
- ⇨ Reset Part via Watchdog Timer (this will resume execution in the Bootloader) (Note: Resetting the part, resets all SFRs and clears any interrupt flags)

Step 3: Bootloader checks if BOOTEN is set. If set execute the Bootloader Program
- ⇨ Enter ULOAD mode
- ⇨ Wait for Erase Command
- ⇨ Erase Code Space

Step 4: Receive Download Command
- ⇨ Wait for Download Command
- ⇨ Download Code

Step 5: Receive Download OK Command
- ⇨ Exit ULOAD mode
- ⇨ Clear BOOTEN
- ⇨ Jump to E000H

## 2.0 EXAMPLE BOOTLOADER CODE DESCRIPTION:

This section describes the example piece of bootloader code (bootload.asm). Refer to the code in section 2.2 and the flowchart in section 2.1 while referring to the code description below.

The user code located at 0000h is a simple blink routine. This blink routine flashes the LED at about 5Hz. To allow the user code to be upgraded simply press the INT0 button to activate the enabled external interrupt. In the INT0 ISR the code waits to receive a 'U' (for upgrade code) over the UART at 115200. If this is received the BOOTEN bit is set, indicating that a new download of code is required and the part is reset via the watchdog timer.

Program execution will restart in the bootloader if the "RUN from E000H" option is enabled. To enable this feature the user must send the following command after the initial download of the bootloader (see uC004)

**<07h><0Eh><02h><'F'><FEh>< CS = BAh>**

This can be done automatically by clicking on the Start Code at E000h button on the WSD available as part of the QuickStart development system.

Once code execution starts at E000h the bootloader reads the BOOTEN bit (LSB of first byte in flash/EE data memory). The first time program execution commences then code will remain in the bootloader because the BOOTEN bit is in the erased state (set).

In this example we firstly reset the watchdog timer to its default value, then configure the UART for 115200 baud which requires running at the maximum PLL frequency and then configure in ULOAD mode.

We then wait for the Erase, Download Page or Download OK command. If we receive an ERASE command, we erase the 56kBytes of flash/EE program memory, send an ACK byte to indicate to the host that we are ready to receive data again and return to waiting for the next command.

If we receive the DOWNLOAD PAGE command we receive the appropriate data, program the data to the appropriate page and then verify that the page has been programmed correctly. If the page has been verified correctly we then send an ACK and wait to receive the next command. If we receive another DOWNLOAD PAGE command the process is repeated until all the appropriate bytes have been programmed.

At the end of the download the DOWNLOAD OK command should be sent. After receiving this command the program will clear the BOOTEN bit to indicate that a valid download occurred, send an ACK frame, reset all SFRs used in the bootloader and then jump to user code at 0000H. With the BOOTEN bit cleared user code at 0000H will always be executed after a reset or power cycle.

Both the original user code, blink1.asm (blinks LED at 5Hz) and a second example of user code, blink2.asm (blinks LED at 1Hz) can be used to demonstrate how simple it is to upgrade user code. Here are the steps involved.

1) Download **bootload.hex** from the WSD with the RUN from E000H feature enabled (code should execute in the bootloader). Run the code.
2) In bootload.exe click on the "**Execute**" button. User code should run, blinking the LED at 5Hz.
3) To upgrade the code click on the **INT0 button** on the eval board
4) Click on the "**Upgrade Code**" button
5) Click on the "**Erase**" button to erase the code
6) Click on the "**Download**" button to download new code. Select **blink2.hex**
7) Click on the "**Execute**" button to run the new code. The new user code should run, blinking the LED at 1Hz
8) The original code can be downloaded again by repeating steps 3 to 7 and this time selecting blink1.hex.

## 2.1 EXAMPLE CODE FLOWCHART:

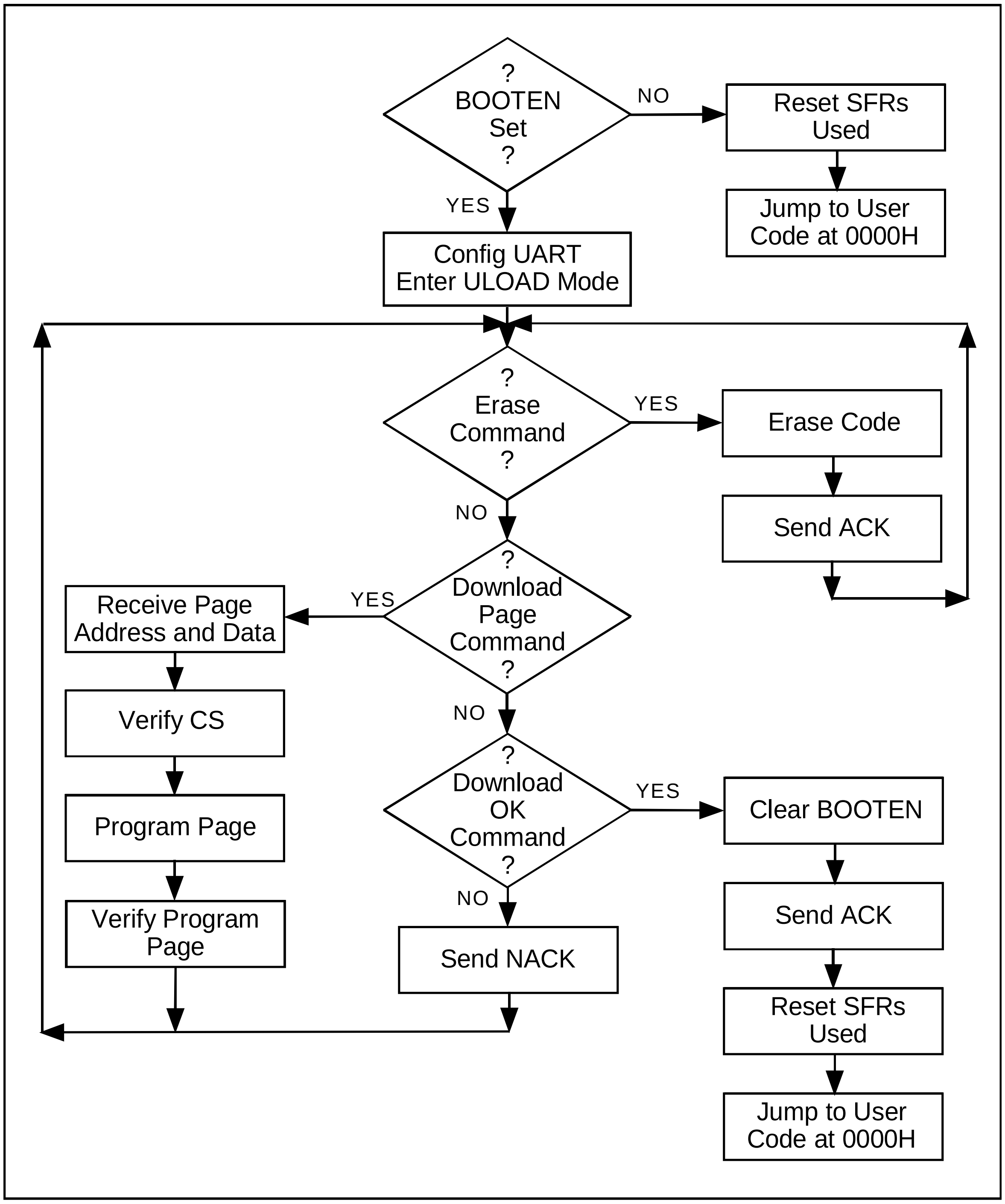

Figure 3: Flow Chart for the Bootloader Example Code

## 2.2 EXAMPLE BOOTLOADER CODE:

```
;********************************************************************
;
; Author        : ADI - Apps            www.analog.com/MicroConverter
;
; Date          : November 2001
;
; File          : Bootload.asm
;
; Hardware      : ADuC834
;
; Description   : Example bootloader program that lies in upper
;                 6kbytes of the 62kByte code space. This bootloader
;                 can be used to download to the bottom 56kBytes.
;
;                 NOTE: This program will only work if the option to
;                       always run code from E000H after download is
;                       selected.
;
;********************************************************************

$MOD834

ACK         EQU   06H
NACK        EQU   15H


;**********************************************************************
;                        EXAMPLE USER CODE
;**********************************************************************
CSEG
ORG     0000H
      AJMP     MAIN

;____________________________________________________________________
                                                           ; INT0 ISR
ORG     0003H
      ; wait to receive a character from the UART
      JNB   RI, $                   ; wait for reception
      CLR   RI
      MOV   A, SBUF
      CJNE  A, #'U', ERROR

      ; plan to upgrade new code => set BOOTEN
      MOV   EADRH, #0
      MOV   EADRL, #0
      MOV   ECON, #1                ; read page
      ORL   EDATA1, #1              ; SET LSB
      MOV   ECON, #5                ; ERASE page
      MOV   ECON, #2                ; program page
      MOV   ECON, #4               ; verify page
      MOV   A, ECON
      JNZ   ERROR
      ; use the watchdog timer to reset part...run from E000H after reset
      CLR   EA                     ; disable interrupts for double
                                   ; write sequence
      SETB  WDWR
      MOV   WDCON, #82h

ERROR:
      RETI

;____________________________________________________________________
                                                              ; MAIN
MAIN:
      ; enable INT0
      SETB  IT0              ; INT0 edge triggered
      SETB  EA               ; enable inturrupts
      SETB  EX0              ; enable INT0

      ; configure at fastest freq
      MOV   PLLCON, #0

      ; configure UART for 115200
      MOV   T3CON, #81h
      MOV   T3FD, #2Dh
      MOV   SCON, #52H

      ; THIS SIMPLE BLINK ROUTINE REPRESENTS THE MAIN PROGRAM
BLINK:
      CPL   P3.4
      CALL  DELAY
      AJMP  BLINK

;____________________________________________________________________
                                                             ; DELAY
DELAY:
      ; 100ms DELAY
      MOV   R0,#205
DLY:
      MOV   R1,#255                ; 205 x 255 x 1.91us
      DJNZ  R1,$
      DJNZ  R0,DLY

      RET
;____________________________________________________________________
```

```
;**********************************************************************
;                       EXAMPLE BOOTLOAGER CODE
;**********************************************************************
CSEG
ORG 0E000h
      ; read BOOTEN
      MOV   EADRH, #0
      MOV   EADRl, #0
      MOV   ECON, #1              ; read page
      MOV   A, EDATA1
      JB    ACC.0, BOOTLOADER

      ; reset EDATA1-4 before running user code
      CLR   A
      MOV   EDATA1, A
      MOV   EDATA2, A
      MOV   EDATA3, A
      MOV   EDATA4, A
      LJMP  0000H

BOOTLOADER:
      ; clear the deliberate WDT reset
      SETB  WDWR
      MOV   WDCON, #10H

      ; configure UART for 115200 baud
      MOV   PLLCON, #0            ; run core at max speed
      MOV   T3CON, #81H
      MOV   T3FD, #2Dh
      MOV   SCON,#52h

      ; configure in ULOAD mode
      MOV   ECON, #0F0h

GETCOMMAND:
      CALL  RECBYTE
      CJNE  A, #'E', $+5
      AJMP  ERASECOMMAND
      CJNE  A, #'D', $+5
      AJMP  DOWNLOADCOMMAND
      CJNE  A, #'O', $+5
      AJMP  DOWNLOADOKCOMMAND
      AJMP  SENDNACK

;====================================================================
;                          ERASE CODE
;====================================================================
ERASECOMMAND:
      ; wait for erase command
      ; <'E'><CS>
      CALL  RECBYTE
      ADD   A,#'E'
      JZ    ERASEOK
      AJMP  SENDNACK

ERASEOK:
      ; erase 56kbytes of code space
      MOV   ECON, #6

      ; send ACK
      AJMP  SENDACK

;=====================================================================
;                            DOWNLOAD CODE
;=====================================================================
DOWNLOADCOMMAND:
      ; wait for download command
      ; <'D'><PAGEADD><DATA0->255><CS>

      MOV   R0, #'D'
      ; get page address
      CALL  RECBYTE
      MOV   EADRH, A
      mov   eadrl, #13
      ADD   A, R0
      MOV   R0, A

      ; if page address =FFh then exit download
      MOV   A, EADRH
      CJNE  A, #0E0H, $+3
      JC    ADDRESSOK               ; C=0 for EADRH < E0h
      AJMP  SENDNACK

ADDRESSOK:
      MOV   DPTR, #0
      MOV   R1, #0                  ; count
      MOV   CFG834, #1              ; int XRAM
READDATA:
      ACALL RECBYTE
      MOVX  @DPTR, A
      INC   DPTR
      ADD   A, R0
      MOV   R0, A
      DJNZ  R1, READDATA            ; REPEAT 256 TIMES

      ; verify checksum
      ACALL RECBYTE
      ADD   A, R0
      MOV   R0, A
      JZ    DOWNLOADCHECKSUMOK
      AJMP  SENDNACK
```

```
DOWNLOADCHECKSUMOK:
      ; program page
      MOV   ECON, #2

; verify download
      MOV   DPCON, #54H            ; main DPTR in auto INC mode
                                   ; shadow DPTR in auto INC mode
                                   ; DPTR in aut toggle mode
      MOV   DPTR, #0               ; main DPTR=0 (XRAM)
      INC   DPCON                  ; select shadow DPTR
      MOV   DPH, EADRH             ; shadow DPTR (CODE)
      MOV   DPL, #0
      MOV   R0, #0

VERIFYDOWNLOADLOOP:
      ; read code memory
      CLR   A
      MOVC  A, @A+DPTR             ; swap to main DPTR
      MOV   B, A
      MOVX  A, @DPTR
      CJNE  A, B, JMPSENDNACK
      DJNZ  R0, VERIFYDOWNLOADLOOP
      MOV   DPCON, #0

      AJMP  SENDACK

JMPSENDNACK:
      AJMP  SENDNACK

;=====================================================================
;                          DOWNLOAD OK COMMAND
;=====================================================================
DOWNLOADOKCOMMAND:
      ; wait for Download OK command
      ; <'O'><CS>
      CALL  RECBYTE
      ADD   A,#'O'
      JZ    EXITULOADMODE
      AJMP  SENDNACK

EXITULOADMODE:
      ; exit ULOAD mode
      MOV   ECON, #0Fh

DOWNLOADOK:
      ; clear BOOTEN
      MOV   EADRH, #0
      MOV   EADRL, #0
      MOV   ECON, #1               ; read page
      ANL   EDATA1, #0FEh          ; clear LSB
      MOV   ECON, #5               ; ERASE page
      MOV   ECON, #2               ; program page
      MOV   ECON, #4                 ; verify page
      MOV   A, ECON
      JZ    BOOTENCLEAR
      AJMP  SENDNACK

BOOTENCLEAR:
      ; send an ACK
      MOV   A, #ACK
      CALL  SENDBYTE
      JNB   TI, $                    ; disabling UART shortly
                                     ; => wait for char to send

RESETSFR:
      ; reset SFRs
      CLR   A
      MOV   B, A
      MOV   PSW, A
      MOV   EADRH, A
      MOV   EADRL, A
      MOV   EDATA1, A
      MOV   EDATA2, A
      MOV   EDATA3, A
      MOV   EDATA4, A
      MOV   DPCON, #1
      MOV   DPTR, #0                 ; clear shadow DPTR
      MOV   DPCON, A
      MOV   DPTR, #0                 ; clear main DPTR
      MOV   CFG834, A
      MOV   PLLCON, #3               ; run core at max speed
      MOV   T3CON, A
      MOV   T3CON, A
      MOV   SCON, A

      ; jump to 0000H
      LJMP  0000H
```

```
;====================================================================
;                          FUNCTIONS
;====================================================================
;____________________________________________________________________
                                                            ; RECBYTE
RECBYTE:        ; waits for a single ASCII character to be received
                ; by the UART.  places this character into A.

        JNB     RI,$
        MOV     A,SBUF
        CLR     RI

        RET

;____________________________________________________________________
                                                           ; SENDBYTE
SENDBYTE:       ; sends ASCII value contained in A to UART

        JNB     TI,$            ; wait til present char gone
        CLR     TI              ; must clear TI
        MOV     SBUF,A

        RET
;____________________________________________________________________
                                                   ; SENDACK/SENDNACK
SENDACK:
      MOV   A, #ACK
      AJMP  CONTSENDACK
SENDNACK:
      MOV   A, #NACK
CONTSENDACK:
      ACALL SENDBYTE

      AJMP  GETCOMMAND
;____________________________________________________________________

END
```