TMS320F243 Serial Boot Loader

1.0 Introduction

This document describes the implementation of a serial boot loader for the TMS320F243 Digital Signal Processor. The serial boot loader aims at in-situ programming of the flash array on the TMS320F243 device.

The TMS320F243 device has an on chip 8K Word (8Kx16bits) flash array, for use as program memory. The Flash array may be programmed via a JTAG interface, however the JTAG interface needs extra interface circuitry and also the existence of the JTAG connector on the target. The serial boot loader is designed to program the flash memory array by making use of the Serial Communication Interface(SCI). The F243 SCI is an industry standard RS232 serial communication port and allows easy interface to a host PC communication port via an appropriate RS232 driver circuit. This serial boot loader provides a convenient technique to program the flash array through the serial interface which may be used for user functions during normal operation.

1.1 Overview

For this operation a minimal amount of boot loader code must be resident in the flash array. In this implementation the F24x flash array has resident only the comms kernel and sequencing algorithm. The actual Clear, Erase & Program routines are received from the Host during the programming sequence & executed by the sequencing s/w as required. This allows the resident boot loader code to be small. Additionally, since the Clear, Erase, & Program algorithms are kept on the host, they can be updated in future version releases, without affecting the resident Flash s/w which is Programmed as part of Device Test.

Since the Comms + Sequencing s/w (Flash resident) is also Erased during the Programming sequence, it must be “re-loaded” after the User’s code. This is conveniently done by allowing the Host utility to send a copy of the Boot Loader code to the DSP after completion of the User’s code programming.

The F24x devices have a total of 544 Words of RAM, of which 256W can be used as Program memory. The entire Programming sequence must therefore be completed by making use of only the available 256W of program memory. This requires that the size of the comms kernel plus any of the Clear, Erase or Programming routines must execute entirely from 256W of program memory.

A summary of the Programming Sequence is given below. This is with the F24x DSP perspective:

1. After boot-up, s/w control passes to the Comms kernel if the BIO pin is set high.
2. If BIO is high, execute the Baud rate-detection routine to synchronize to Host comms baud rate.
3. Comms kernel (excluding Baud rate-detection portion) “copies itself” to RAM B0.
4. Receive CLEAR algorithm from Host & transfer it to RAM B0
5. Execute CLEAR routine.
6. Receive ERASE algorithm from Host & transfer it to RAM B0 over-writing previous CLEAR routine.
7. Execute ERASE routine.
8. Receive PROGRAM algorithm from Host & transfer it to RAM B0 over-writing previous ERASE routine.
9. Execute PROGRAM routine. This is done by receiving data blocks from the Host, storing them temporarily in RAM B1 & then executing the PROGRAM routine. This is repeated until all Data blocks have been downloaded from the Host.
10. Receive Serial Boot loader code as another block & program it at address 1F00h.

Notes:

1. The Serial Boot loader code block is actually treated as just another User code block with it's own destination address & length parameters, & hence it is transparent to the Comms Kernel & Sequencer running in B0.
2. **The programming sequence must not be interrupted at any point after the baud rate lock to the completion of the programming sequence. If this happens, the Serial Boot Loader in the Flash may be left in a corrupted state and then must be restored as discussed in section 2.1.**
3. All algorithms and files for the user and factory version of the boot loader are available as part of the Serial Boot Loader package.
4. The Serial Boot Loader package is also available on Texas Instruments website.

2.0 Operation sequence - In detail

1. At Boot-up the Reset vector causes a branch to comms kernel at location: 1F00h (this allows up to 256 words for complete Boot loader including baud rate-detection routine & initialization(1F00 to 1FFF =256Words).

   If BIO pin is high (+5V) initiate comms session with Host in preparation for Flash array Programming cycle. (i.e. proceed to #2)

   If BIO pin is low (0V) pass control over to user's code at 0040h.

2. Auto-baud session with Host.
   Comms kernel The comms kernel on the F243 is indeed a simplified version of the F240 version: it simply emulates the protocol for the F240. This means that the kernel simply does a hand-shake which synchronizes the comms sesssion, no actual baud rate *detect* is performed.

Host side:

Host keeps sending same character (i.e. 0Dh, CR character) and waits until the correct acknowledge character (0AAh) from the Slave is sent. Host waits for the acknowledge character for a defined time out period, after which the Host gives up & the comms session is aborted.

Slave side:

Slave sets baud rate control at the final rate, setting and attempts to communicate on the serial line by receiving a character & comparing it to an expected character. If different, the baud rate is increased to the next defined PLL setting. Once a reliable "match" is made, an acknowledge character (0AAh) is sent to the Host to signify a baud rate lock has occurred.

Assumption:

CLKIN **must** be 5 MHz for the factory programmed boot-loader to operate with the present PC Host & the Host baud rate is 38.4 Kb/s
Figure 2 shows a flow chart for the Baud rate-detection process & Table 1 below shows the allowable CLKIN frequencies that can be used by the user.

Table 1. Allowable CLKIN frequencies.

Clkin frequency	PLL x	DSP Clkout
5	4	20

3. Self copy of Comms kernel to B0:
   Main portion of Comms kernel plus the sequencer is copied from Flash array to B0. The Auto-baud portion is no longer required and can be abandoned to save memory. B0 is then selected as program memory at FF00h and control then passes to B0 in preparation for the CLEAR operation.

4. Receive CLEAR Algorithm from Host (Filename : "Clr24_x1.hex").
   Host sends: - Start address (dummy only)

- Length (i.e. number of bytes to transfer)
- Actual algorithm code words, split into bytes(LSByte first).

The start address is not required for the clear, erase and program routines, but a dummy start address is included to maintain the same header format, the routines always go into B0 for execution.
Slave packs bytes into words & stores CLEAR routine in B0 just after the comms kernel itself. Control is then passed to the CLEAR routine.

5. Execute CLEAR routine:
   If successful, send "OK" to Host, and proceed with ERASE.
   If failure to Clear, send error code to Host & abort process.

6. Receive ERASE algorithm from Host (Filename : "Era 24_x1.hex").
   Host sends: - Start address (dummy only)
   - Length (i.e. number of bytes to transfer)
   - Actual algorithm code words, split into bytes(LSByte first).

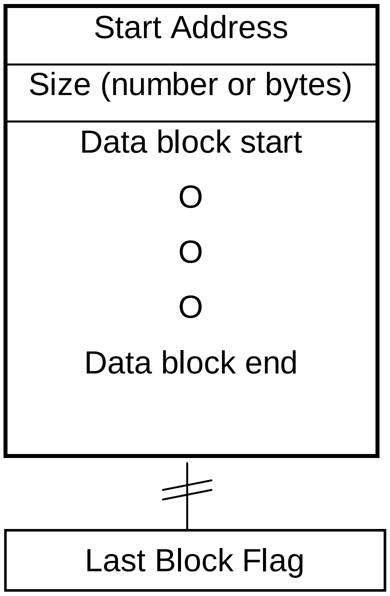
   Slave packs bytes into words & stores ERASE routine in B0 just after the comms kernel itself, over-writing the previous CLEAR routine. Control is then passed to the ERASE routine.

7. Execute ERASE routine
   If successful, send "OK" to Host, and proceed with PROGRAM.
   If failure to Erase, send error code to Host & abort process.

8. Receive PROGRAM algorithm from Host:
   Host sends: - Start address (dummy only)
   - Length (i.e. number of bytes to transfer)
   - Actual algorithm code words, split into bytes(LSByte first).

Slave packs bytes into words & stores PROGRAM routine in B0 just after the comms kernel itself, over-writing the previous CLEAR routine. Control is then passed to the PROGRAM routine.

9. Execute PROGRAM sequence:
   i) Indicate to Host - “Ready to receive Data block”
   Data is retrieved from the hex file generated by your program.
   ii) Host sends Data block in following format:



Note:

- Max Data block size is 200 words.
- Start address is destination address in Flash array.
- Data block is received in bytes then packed into words & stored in B1.

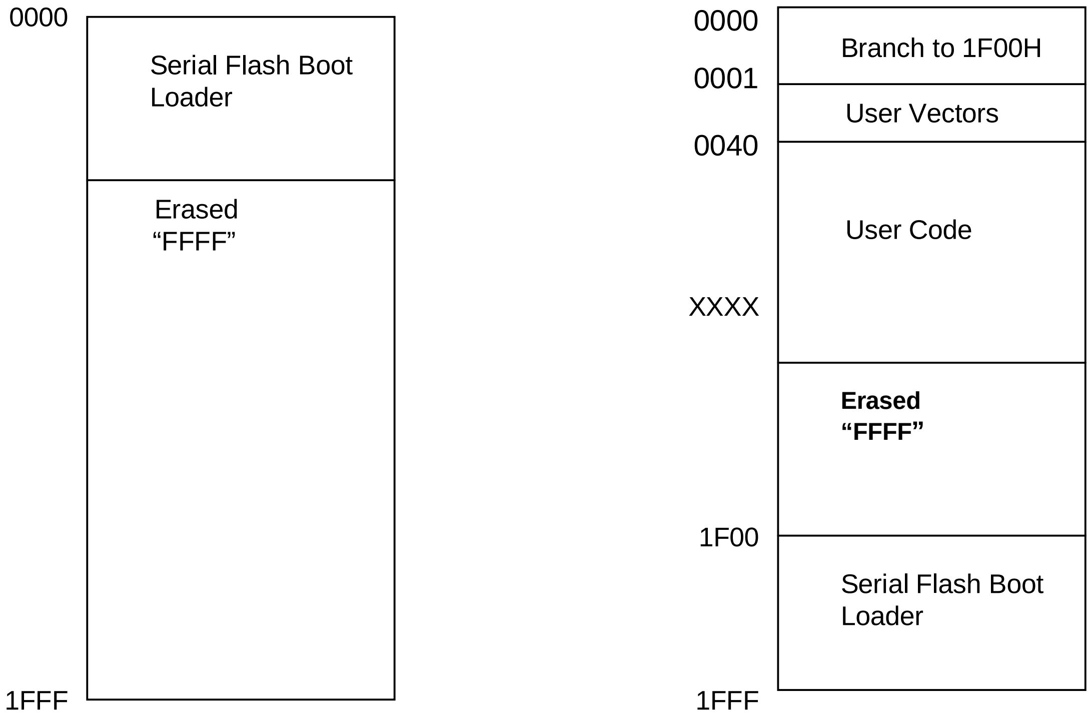
iii) Pass the Start address & Size (i.e. length of block) to the PROGRAM routine & execute.

iv) Send Pass / Fail status to Host. If Fail, host should abort the process.
v) Once the host gets a ‘success’ status, the host sends the ‘last block word’; the target acts upon this and goes through one more programming cycle if non-zero. If the last block flag (word) is zero, then the target terminates in a endless loop.

The Memory Maps given in Fig 1, show the memory locations of the algorithms for a new factory programmed device and for a device programmed via the serial boot loader. The device as shipped has a slightly different configuration, in that the boot loader is programmed at the beginning of the flash with no user code. (Fig 1a) Once the device is programmed the first time, the configuration is as shown in Fig 1b, with the boot loader resident at the high end of the flash.

2.1 Restoring the Serial Boot Loader Code after a program sequence failure

The programming sequence moves the boot loader code into RAM, while the flash is cleared, erased and reprogrammed with the users code and the boot loader itself. If the programming sequence is interrupted for any reason, such as comms link failure or power supply interruption, then essentially the device would be left with corrupted boot loader code in the flash. If this occurs the device must be reprogrammed by means of the JTAG programming utilities. To do this a JTAG connector must be available in the system. JTAG tools are then used to program the “sfpe_b.out” (included in the serial boot loader release) into the flash to restore uncorrupted boot loader code into the flash. The device then may be reprogrammed via the serial link as usual.



(a) Memory Map for factory programmed device

(b) Memory Map for serial programmed device

Figure 1. Serial Flash Boot Loader Memory Configuration.

3.0 Flowcharts for the Serial Boot Loader

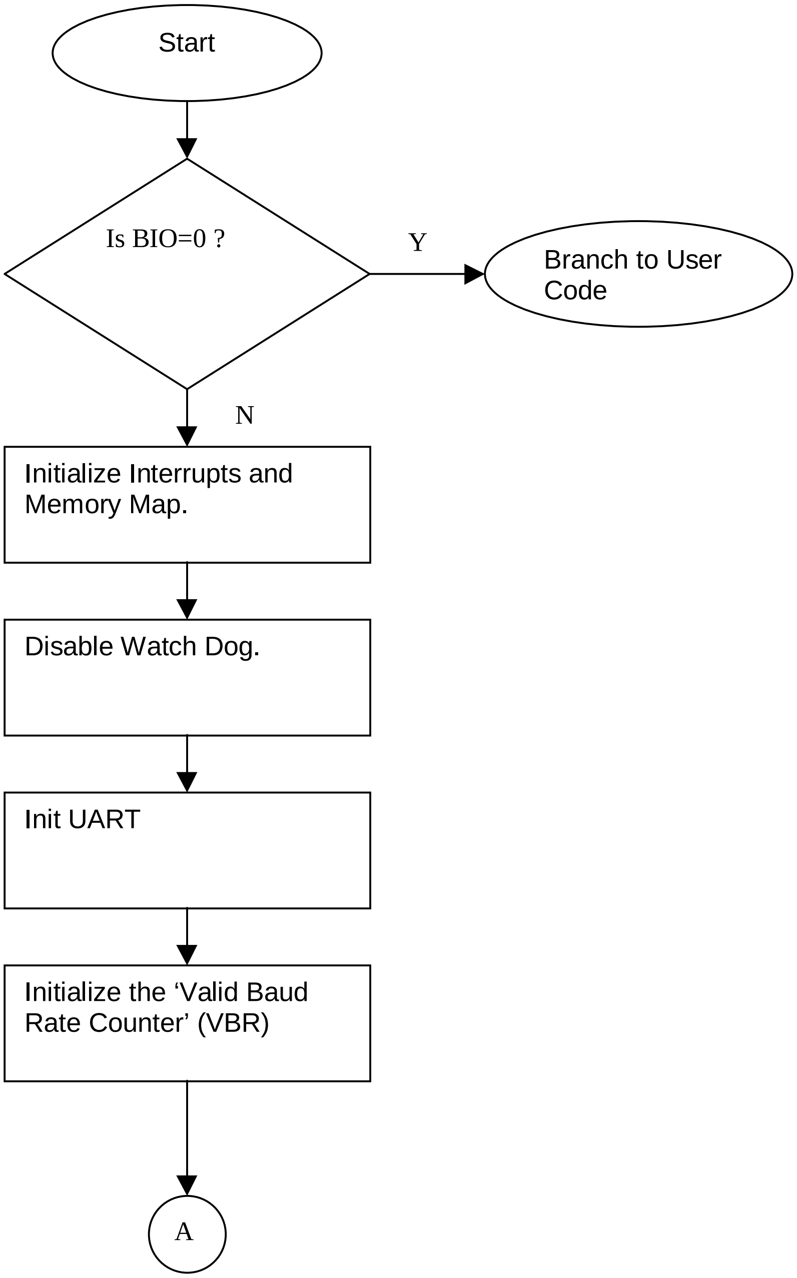

Start
Is BIO=0 ?
Y
Branch to User Code
N
Initialize Interrupts and Memory Map.
Disable Watch Dog.
Init UART
Initialize the 'Valid Baud Rate Counter' (VBR)
A

Figure 2. Flowchart for the Serial Boot Loader Initialization.

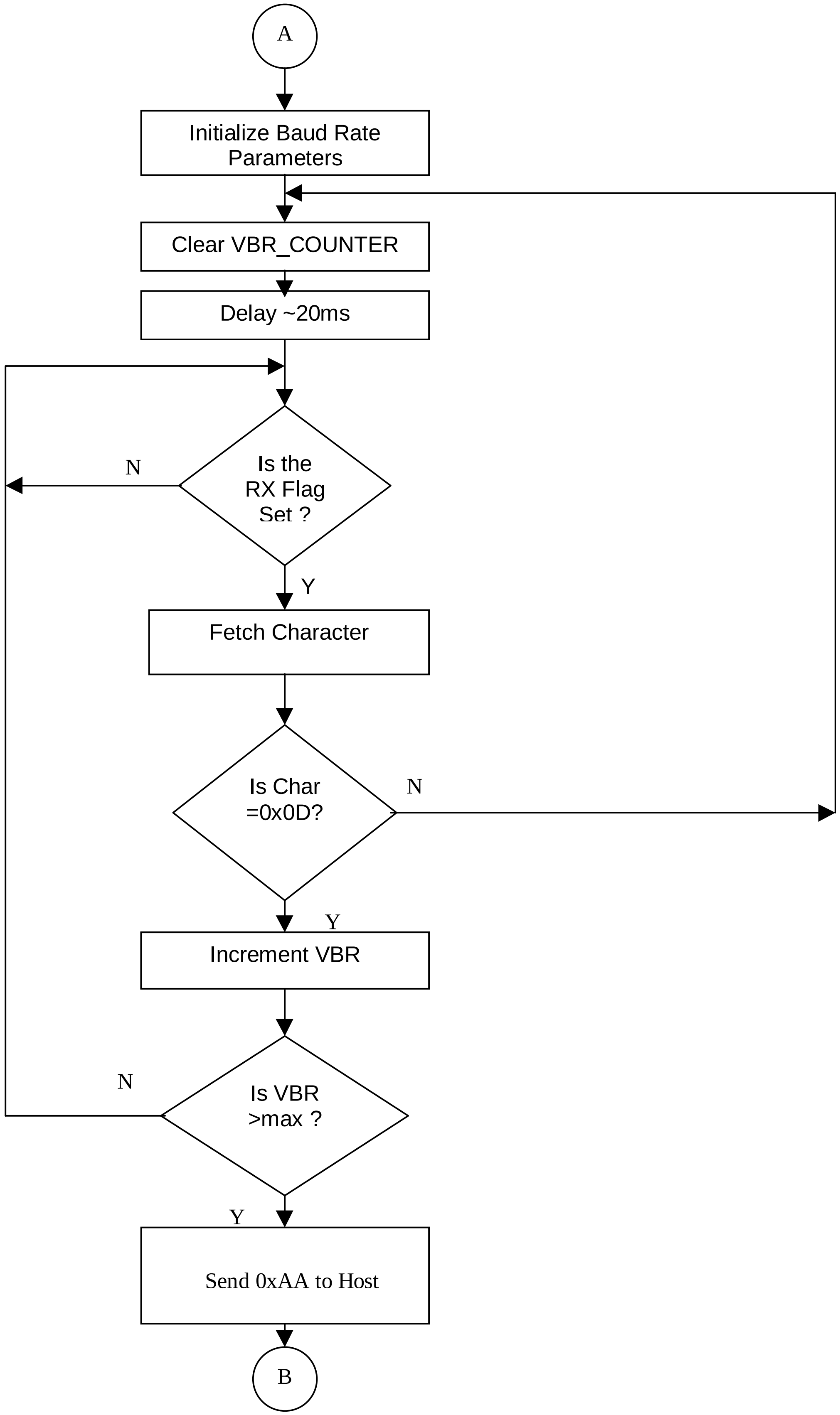


Figure 3. Flowchart for the Serial Boot Loader (contd.)
Baud Rate Detection Algorithm.

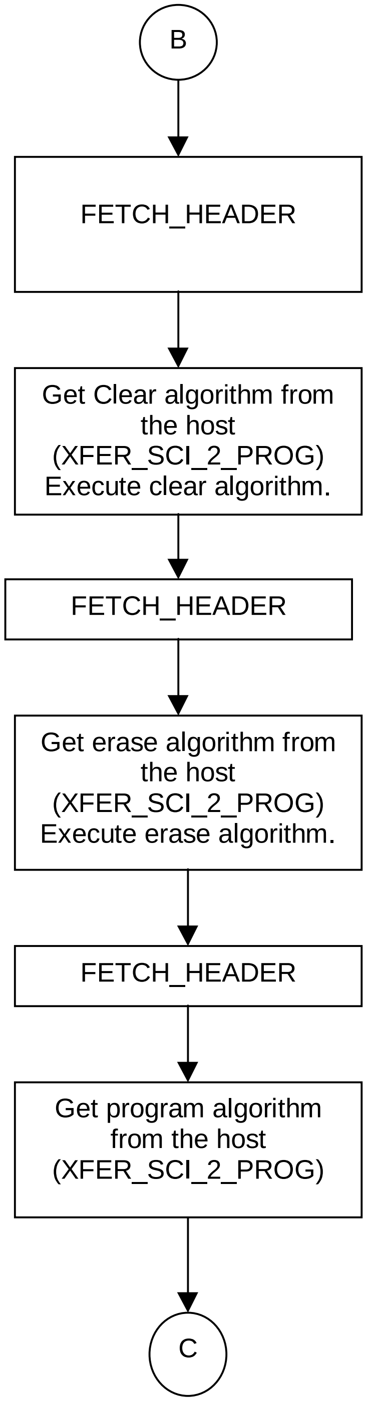

B
FETCH_HEADER
Get Clear algorithm from the host (XFER_SCI_2_PROG) Execute clear algorithm.
FETCH_HEADER
Get erase algorithm from the host (XFER_SCI_2_PROG) Execute erase algorithm.
FETCH_HEADER
Get program algorithm from the host (XFER_SCI_2_PROG)
C

Figure 4 Flowchart for the Serial Boot Loader (contd.).

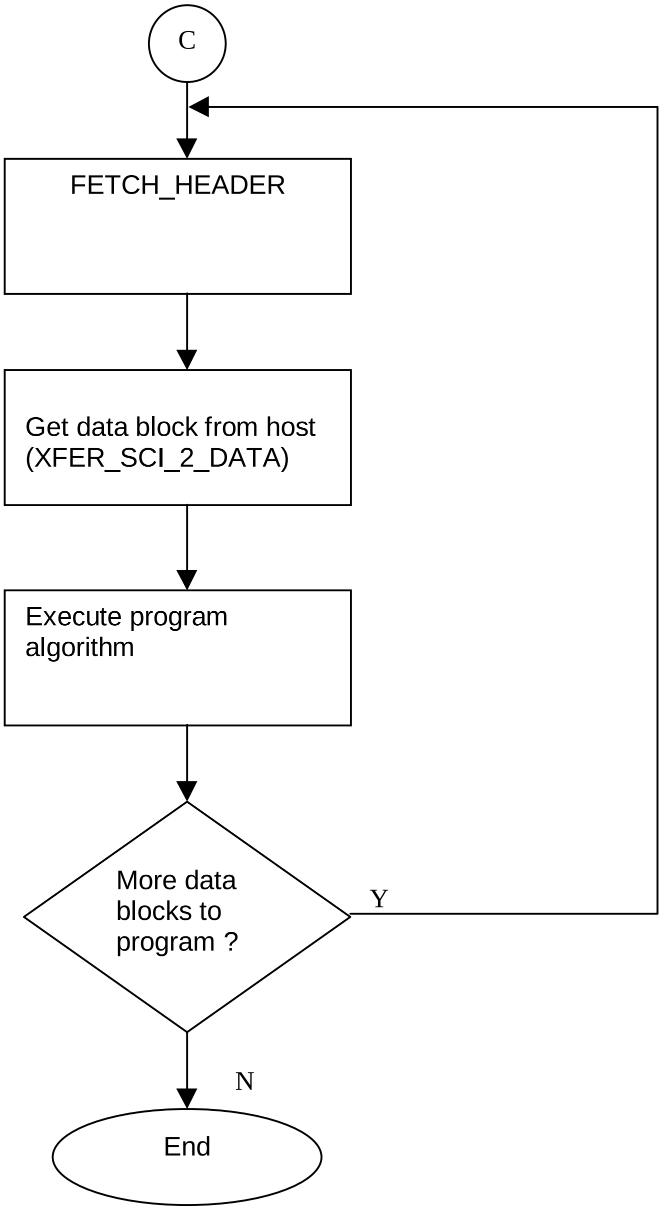
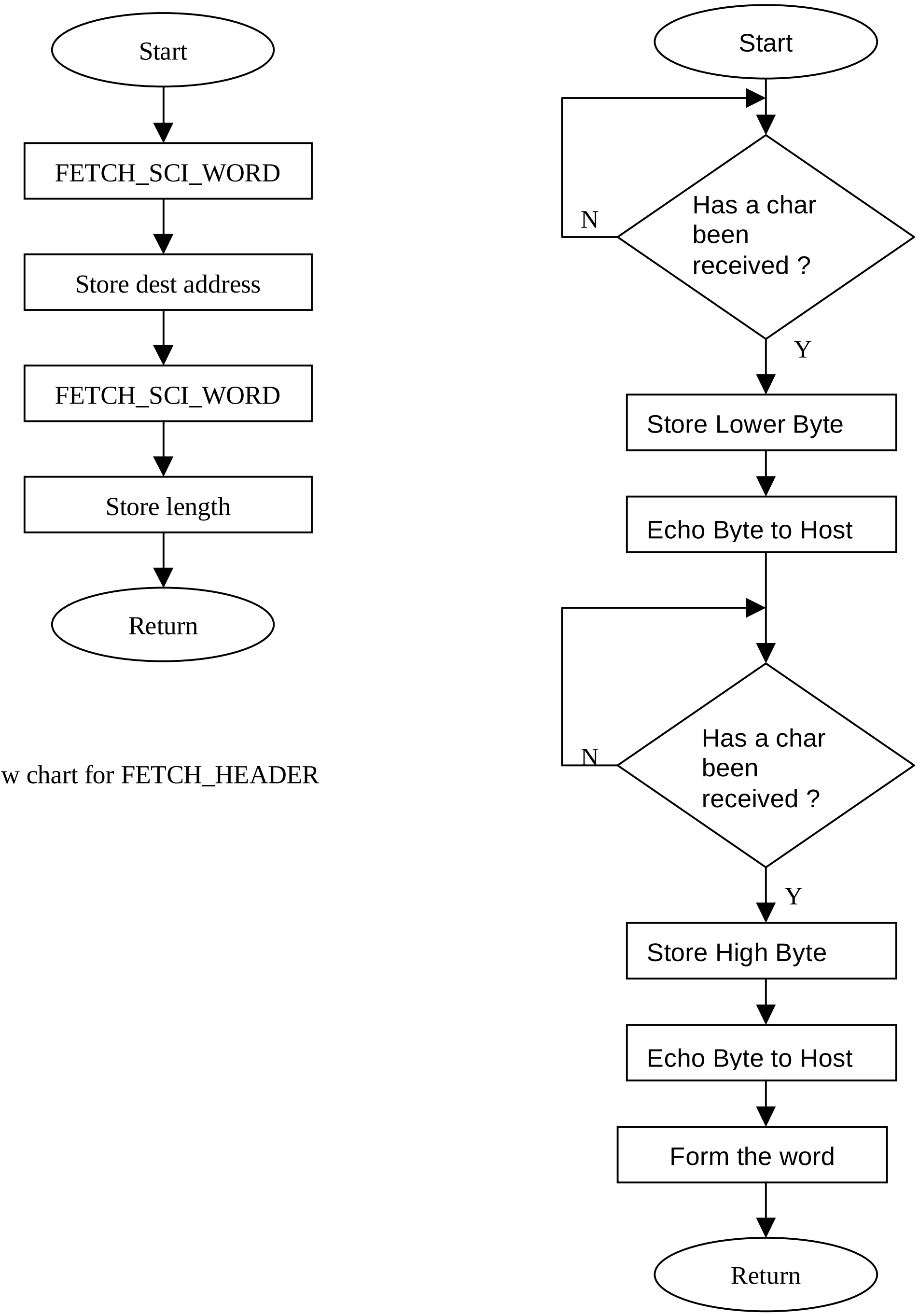


Figure 5 Flowchart for the Serial Boot Loader (contd.).



(a) Flow chart for FETCH_HEADER

(b) Flow chart for FETCH_SCI_WORD

Figure 6 Flowchart for the Serial Boot Loader (contd.)

4.0 Assembly Code Listing

```
;Conditional assembly directives
USER_VERSION    .set          1      ;Set to 1 if for User version
                              ;Set to 0 if for PE version
;**********************************************************************
;
; File Name:   SFLSH_X8.ASM
;
; Project:     F243 Serial Boot loader
; Originator:  TMS320F24x Applications Group, Texas Instruments.
;
; Target Sys:  F241/3 with Serial Communication Hardware.
;
; Description: Serial Boot loader using an Intelligent Windows based ;
               PC Host communicating via the SCI on the target F243.
;
;              This is the 241/3 version of the Serial Boot Loader.
;              This has been derived from the 'F240 Serial Boot Loader.
;
;Status:       Release 2.12
;
;
;
;
;Last Update:  August 1, 1999
;
;**********************************************************************
;
;----------------------------------------------------------------------; Debug directives
;----------------------------------------------------------------------
       .def    GPR0           ;General purpose registers.
               .def    GPR1
               .def    GPR2
               .def    GPR3
               .def    data_buf
               .def    length
               .def    dest_addr
               .def    B2_0
               .def    B2_1
               .def    B2_2
               .def    B2_3
               .def    B2_4
               .def    B2_5
               .def    B2_6
               .def    SPAD1
               .def    SPAD2

               .def    DUMMY
               .def    DLY10
               .def    DLY100
               .def    DLY3K3
               .def    SEG_ST
               .def    SEG_END
               .def    PROTECT

               .def    FL_ADRS
               .def    FL_DATA
               .def    ERROR

;Variables for ERASE and CLEAR
               .def    RPG_CNT
               .def    FL_ST
               .def    FL_END

;Variables for PROGRAM
               .def    PRG_paddr
               .def    PRG_length
               .def    PRG_bufaddr
               .def    BYTE_MASK

               .def    VBR_CNTR
               .def    DELAY
               .def    CHAR_RETRY_CNTR
               .def    ABSOLUTE_CHAR_CNT


                .include        x24x.h
```

```
;Miscellaneous
mSEC_K  .set    1000            ;Delay constant
BUF_SADDR       .set    0328h           ;Start address for Data buffer
WSGR            .set    0FFFFh
VBR_MAX         .set    09h             ;# times valid char needs to be received
CRC_MAX         .set    03h             ;# retries at each PLL setting before giving up.
B0_SADDR        .set    00200h           ;Block B0 start address
USER_START      .set    0040h


;-----------------------------------------------------------------------------
; Variable Declarations for on chip RAM Blocks
;-----------------------------------------------------------------------------
                .bss    GPR0,1          ;General purpose registers.
                .bss    GPR1,1
                .bss    GPR2,1
                .bss    GPR3,1
                .bss    stk0,1
                .bss    stk1,1
                .bss    data_buf,1
                .bss    length,1
                .bss    dest_addr,1
                .bss    PRG_paddr,1
                .bss    PRG_length,1
                .bss    PRG_bufaddr,1
                .bss    BYTE_MASK,1

                .bss    B2_0,1
                .bss    B2_1,1
                .bss    B2_2,1
                .bss    B2_3,1
                .bss    B2_4,1
                .bss    B2_5,1
                .bss    B2_6,1
                .bss    SPAD1,1
                .bss    SPAD2,1

                .bss    DUMMY,1
                .bss    DLY10,1
                .bss    DLY100,1
                .bss    DLY3K3,1
                .bss    SEG_ST,1
                .bss    SEG_END,1
                .bss    PROTECT,1

                .bss    FL_ADRS,1       ;Flash load address.
                .bss    FL_DATA,1       ;Flash load data.
                .bss    ERROR,1         ;Error flag register.

;Variables for ERASE and CLEAR
                .bss    RPG_CNT,1       ;Program pulse count.
                .bss    FL_ST,1         ;Flash start addr/Seg Cntrl Reg.
                .bss    FL_END,1        ;Flash end address.

;Misc variables
                .bss    VBR_CNTR,1
                .bss    DELAY,1
                .bss    CHAR_RETRY_CNTR,1
                .bss    ABSOLUTE_CHAR_CNT,1

;-----------------------------------------------------------------------------
; M A C R O - Definitions
;-----------------------------------------------------------------------------
SBIT0           .macro  DMA, MASK       ;Clear bit Macro
                LACC    DMA
                AND     #(0FFFFh-MASK)
                SACL    DMA
                .endm

SBIT1           .macro  DMA, MASK       ;Set bit Macro
                LACC    DMA
                OR      #MASK
                SACL    DMA
                .endm


KICK_DOG        .macro                  ;Watchdog reset macro
                LDP     #00E0h
                 SPLK    #05555h, WDKEY
                 SPLK    #0AAAAh, WDKEY
                LDP     #0h
                 .endm


 POINT_0        .macro
                LDP     #00h
                .endm

 POINT_B0       .macro
                LDP     #04h
                .endm

 POINT_B1       .macro
                LDP     #06h
                .endm

 POINT_PF1      .macro
                LDP     #0E0h
                .endm

 ;==============================================================================
 ; Start here after Reset.
 ;==============================================================================
                .text

 START:
                POINT_0

                .if (USER_VERSION)
                BCND    USER_START, BIO       ;If BIO=0 go to User's code
                .endif                        ;Else continue with Flsh prg.


                SETC    INTM                  ;Disable interrupts
                SPLK    #0h, IMR              ;Mask all Ints
                SPLK    #0FFh, IFR            ;Clear all Int flags
                CLRC    SXM                   ;Clear Sign Extension Mode
                CLRC    OVM                   ;Reset Overflow Mode
                CLRC    CNF                   ;Config Block B0 to Data
                SETC    XF                    ;set XF Hi (i.e. assert CTS on EVM )


                LDP     #WDCR>>7
                SPLK    #006Fh,WDCR           ;Disable WD if VCCP=5V
                KICK_DOG

 ;===========================================================================
 ;Uart Initialisation
 ;===========================================================================
 UART_INIT:     POINT_B1
                SPLK    #0h, CHAR_RETRY_CNTR  ;Clear retry counter
                SPLK    #0h, ABSOLUTE_CHAR_CNT ;TEST ONLY
                SPLK    #0FFFFh, DELAY

 ;==============================================================================
 ;SCI Initialization
 ;==============================================================================
 SCI_INIT:       LDP     #OCRA>>7
                 SPLK   #0FFFFh,OCRA          ; Set up pins as SCI pins.
                 SPLK   #0FEF3h,OCRB

                 LDP    #SCICCR>>7                   ;1 stop bit,no parity,8 char bits,
                 SPLK    #0007h, SCICCR       ;async mode, idle-line protocol
                 SPLK    #0013h, SCICTL1      ;Enable TX, RX, internal SCICLK,
                 SPLK    #0000H, SCICTL2      ;Disable RX Int, TX Int.
                 SPLK    #0000h, SCIHBAUD
                 SPLK    #0065, SCILBAUD      ;Baud Rate=1200 b/s (10 MHz SYSCLK)


                 SPLK    #0000H, SCIPRI
                 SPLK    #0023h, SCICTL1      ;Relinquish SCI from Reset.

 CLR_VBR_CNTR:  POINT_B1
                SPLK    #0, VBR_CNTR          ;Clear valid baud rate counter

 UI01:          LDP     #SCIRXST>>7
                BIT      SCIRXST,BIT6         ;Test RXRDY bit
                 BCND    UI01,NTC             ;If RXRDY=0,then repeat loop

         ;Check if Char is as expected
                LACC    SCIRXBUF
```

```
CHECK_CHAR:    AND    #0FFh                 ;Clear upper byte
               SUB    #00Dh                 ;Compare with "CR"
               BCND   CLR_VBR_CNTR, NEQ
               LDP    #0

INC_VBRC       POINT_B1
               LACC   VBR_CNTR              ;Inc VBR counter
               ADD    #1h
               SACL   VBR_CNTR
               SUB    #VBR_MAX              ;Is VBR counter > max value ?
               BCND   UI01, NEQ             ;No! fetch another char

SND_ECHO       LACC   #0AAh                 ;Yes!
               LDP    #SCITXBUF>>7
               SACL   SCITXBUF              ;Indicate to Host Baudrate locked

               B      FLSH_INIT


;============================================================================
;Init Flash parameters
;============================================================================
FLSH_INIT      POINT_B1
;              SPLK   #0200, DLY10          ;Delay val for 10uS
;              SPLK   #2000, DLY100         ;Delay val for 100uS
;              SPLK   #0FFFFh, DLY3K3       ;Dealy val for 3.3mS

               SPLK   #0h, SEG_ST           ;Start addr of Flash array
               SPLK   #1FFFh, SEG_END       ;End Addr of Flash array
               SPLK   #0FF00h, PROTECT      ;Enable all segments
               SPLK   #0h, ERROR            ;Clear error flag
               SPLK   #0h, FL_ST            ;Select Flash array 0



               B      XFER_KERNEL



;============================================================================
;Transfer Comms Kernel + sequencer to B0 & set CNF=1 (i.e. B0 = FE00)
;============================================================================
XFER_KERNEL    MAR    *, AR1
               LAR    AR1, #B0_SADDR
               LACC   #kernel_end
               SUB    #kernel_strt
               SACL   GPR1
               LACC   #kernel_strt
               RPT    GPR1
               TBLR   *+
               SETC   CNF
               B      MAIN

;============================================================================
;M A I N   P R O G R A M
;============================================================================
               .sect  "kernel"
               .label kernel_strt
MAIN:
       ;Load & Execute CLEAR
M00            CALL   FETCH_HEADER
               CALL   XFER_SCI_2_PROG


               CALL   ALGO_START
               LACC   ERROR
               CALL   SEND_CHAR      ;Indicate to host Clear finished.

       ;Load & Execute ERASE
M01            CALL   FETCH_HEADER
               CALL   XFER_SCI_2_PROG
               CALL   ALGO_START
               LACC   ERROR
               CALL   SEND_CHAR      ;Indicate to host Erase finished.

       ;Load & Execute PROG
M02            CALL   FETCH_HEADER
               CALL   XFER_SCI_2_PROG

M03            CALL   FETCH_HEADER          ;Get info on Data block
               LACC   dest_addr
               SACL    PRG_paddr              ;Pass Flash dest addr
               LACC    length
               ADD     #01h                   ;adjust for actual length
               SACL    PRG_length             ;Pass Data block length
               SPLK    #BUF_SADDR, PRG_bufaddr ;Pass Data buffer start addr

M04            CALL    XFER_SCI_2_DATA                ;Transfer Data block to B1
               CALL    ALGO_START             ;Execute PROG routine
               LACC    ERROR
               CALL    SEND_CHAR

               CALL    FETCH_SCI_WORD         ;Check if more blocks to come.
               LACC    data_buf               ;If non-zero, then loop
               BCND    M03, NEQ               ;If zero then finish up.


DEND           B       DEND                   ;TEST ONLY

;============================================================================
; Routine Name: F E T C H _ H E A D E R            Routine Type: SR
;
;============================================================================
FETCH_HEADER:  CALL    FETCH_SCI_WORD
               LACC    data_buf
               SACL    dest_addr
               CALL    FETCH_SCI_WORD
               LACC    data_buf
               SACL    length
               RET


;============================================================================
; Routine Name: X F E R _ S C I _ 2 _ P R O G              Routine Type: SR
;============================================================================
XFER_SCI_2_PROG:
               MAR     *, AR0
               LAR     AR0, length
               LACC    #ALGO_START            ;ACC=dest address

XSP0           CALL    FETCH_SCI_WORD
               TBLW    data_buf               ;data_buff-->[ACC]
               ADD     #01h                   ;ACC++
               BANZ    XSP0                   ;loop "length" times
               RET

;============================================================================
; Routine Name: X F E R _ S C I _ 2 _ D A T A              Routine Type: SR
;============================================================================
XFER_SCI_2_DATA:
               MAR     *, AR1
               LAR     AR0, length            ;AR0 is loop counter
               LAR     AR1, #BUF_SADDR        ;Dest --> B1 RAM

XSD0           CALL    FETCH_SCI_WORD
               LACC    data_buf
               SACL    *+, AR0
               BANZ    XSD0, AR1
               RET


;============================================================================
; Routine Name: F E T C H _ S C I _ W O R D        Routine Type: SR
;
; Description: Version which expects Lo byte / Hi byte sequence from Host &
;              also echos byte
;============================================================================
FETCH_SCI_WORD:        POINT_B1
               SACL    stk0
               LDP     #SCIRXST>>7
FSW0           BIT   SCIRXST,BIT6             ;Test RXRDY bit
               BCND  FSW0, NTC                        ;If RXRDY=0,then repeat loop
               LACC    SCIRXBUF               ;First byte is Lo byte
               SACL    SCITXBUF               ;Echo byte back
               AND     #0FFh                  ;Clear upper byte

FSW1           BIT   SCIRXST,BIT6             ;Test RXRDY bit
               BCND  FSW1, NTC                        ;If RXRDY=0,then repeat loop
               NOP
               ADD     SCIRXBUF,8             ;Concatenate Hi byte to Lo
               SFL                            ;used because 7 is max in SACH
               SACH    SCITXBUF,7              ;Echo byte back (after SFL 8)

               POINT_B1
               SFR                             ;restore ACC as before
               SACL    data_buf                ;Save received word

               LACC    stk0
               RET


;================================================================================
; Transmit char to host subroutine.
;================================================================================
SEND_CHAR      LDP     #SCITXBUF>>7
               SACL    SCITXBUF                ;Transmit byte to host.
               POINT_B1
               RET

;================================================================================
;Down-loaded algorithms start here.
;================================================================================
ALGO_START      .word  0
               .label  kernel_end
```