TMS320F240 Serial Boot Loader

1.0 Introduction

This document describes the implementation of a serial boot loader for the TMS320F240 Digital Signal Processor. The serial boot loader aims at in-situ programming of the flash array on the TMS320F240 device.

The TMS320F240 device has an on chip 16Kword 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 F240 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 effecting 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 3E00h.

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: 3E00h (this allows up to 512 words for complete Boot loader including baud rate-detection routine & initialization(3E00 to 3FFF =512Words).

   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 has no knowledge of F24x CLKIN frequency & hence the baud rate selection register bits must be determined by a Baud rate-detection algorithm, as follows:

   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 “some” low initial setting and attempts to converge on the correct baud rate 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 is in the range of 2 MHz to 20 MHz & 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	Actual BR	% ERROR
20	1	20	37.88	1.36
12	1.5	18	38.79	-1.02
10	2	20	37.88	1.36
8	2.5	20	37.88	1.36
6	3	18	38.79	-1.02
5	4	20	37.88	1.36
4	4.5	18	38.79	-1.02
3.5795	5	17.8975	38.57	-0.45
2	9	18	38.79	-1.02

1. 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.

2. 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.

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

4. 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.

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

6. 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.

7. 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:

Start Address
Size (number or bytes)
Data block start O O O Data block end
Last Block flag

Note:

- Max Data block size is 200 words.
- Start address is destination address in Flash array.
- If “Last block” flag is set then no further blocks are expected after this block.
- 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) If “Last block” flag is not set goto i), else Programming sequence is complete.

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.

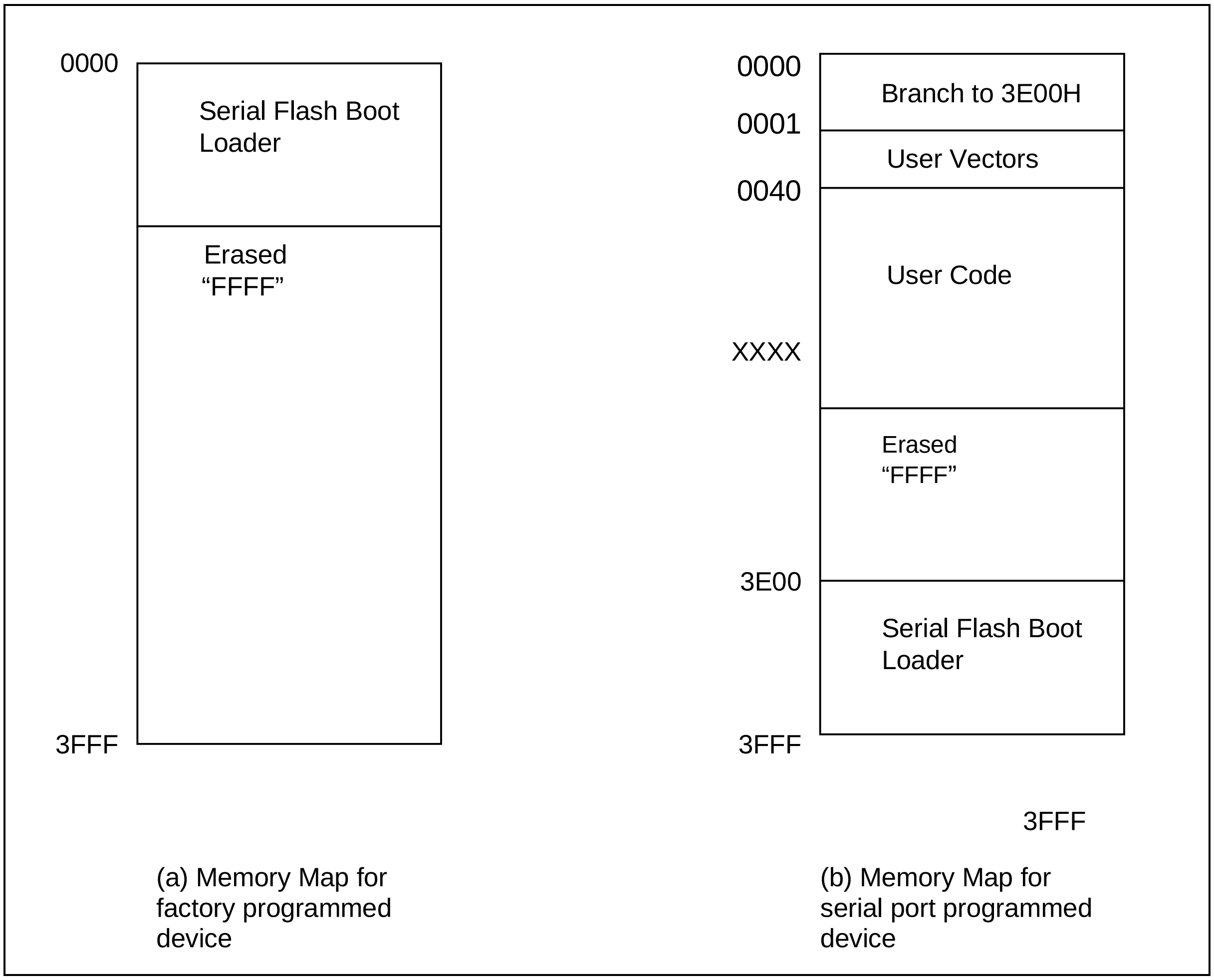


Figure1. Serial Flash Boot Loader Memory Configuration.

3.0 Flowcharts for the Serial Boot Loader

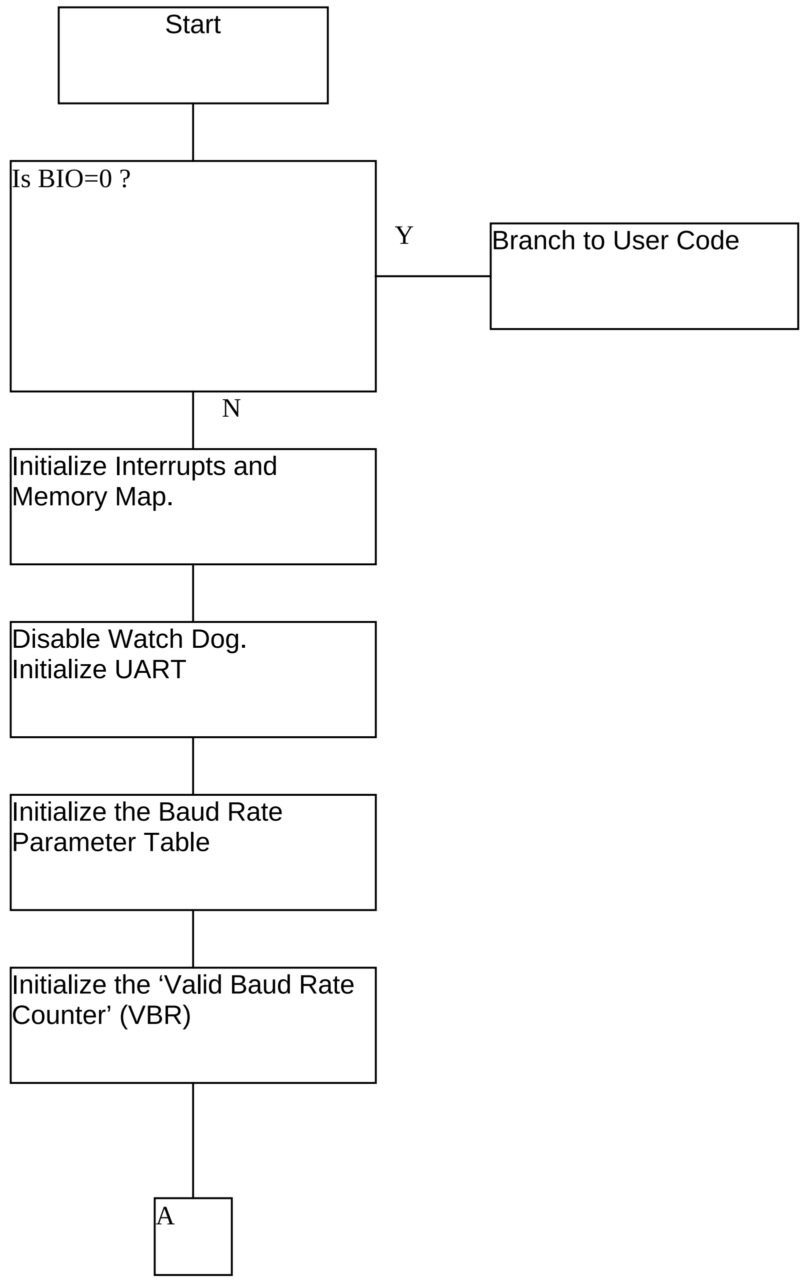


Figure 2. Flowchart for the Serial Boot Loader Initialization.

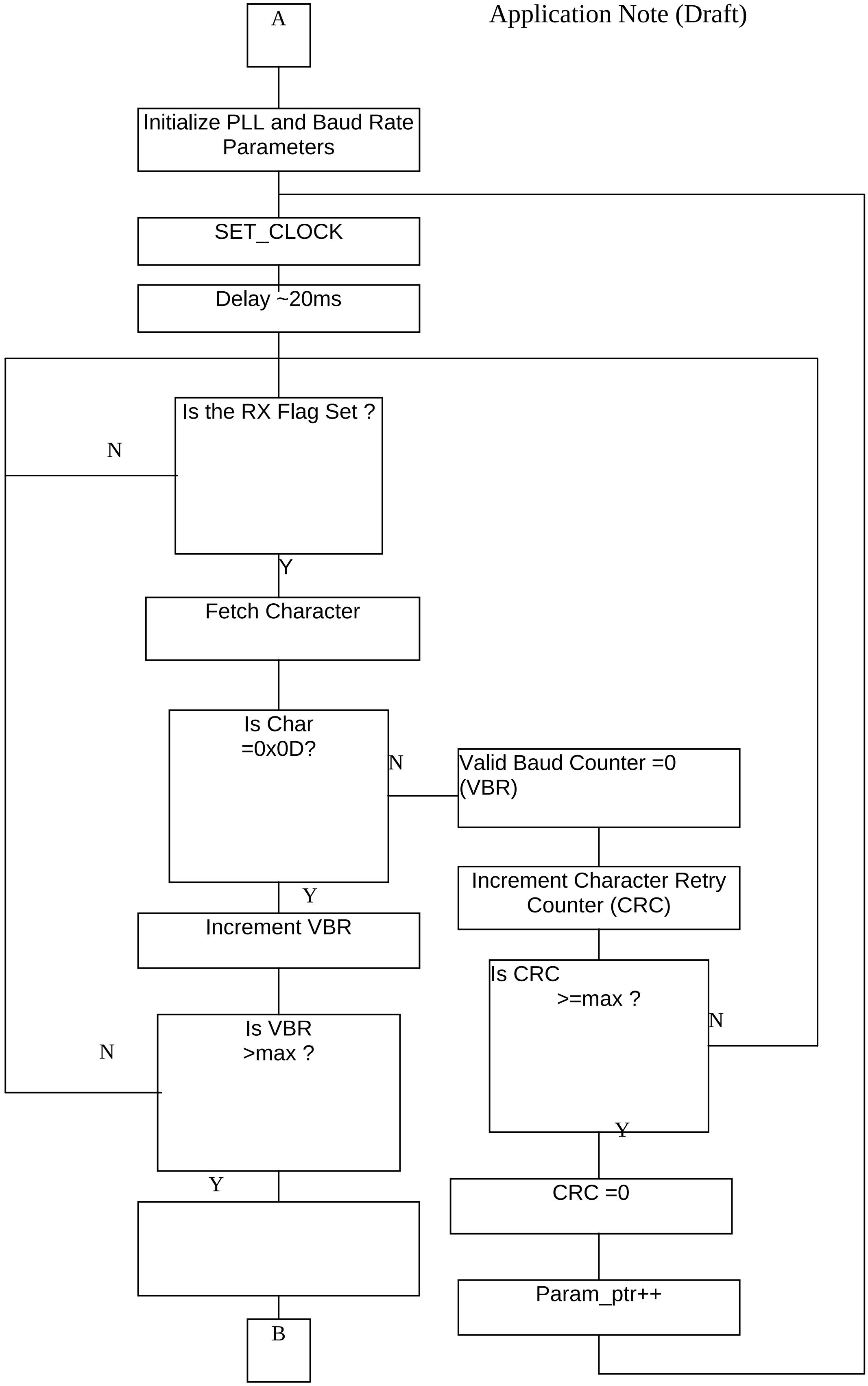


Figure 3. Flowchart for the Serial Boot Loader (contd.)
Baud Rate Detection Algorithm.

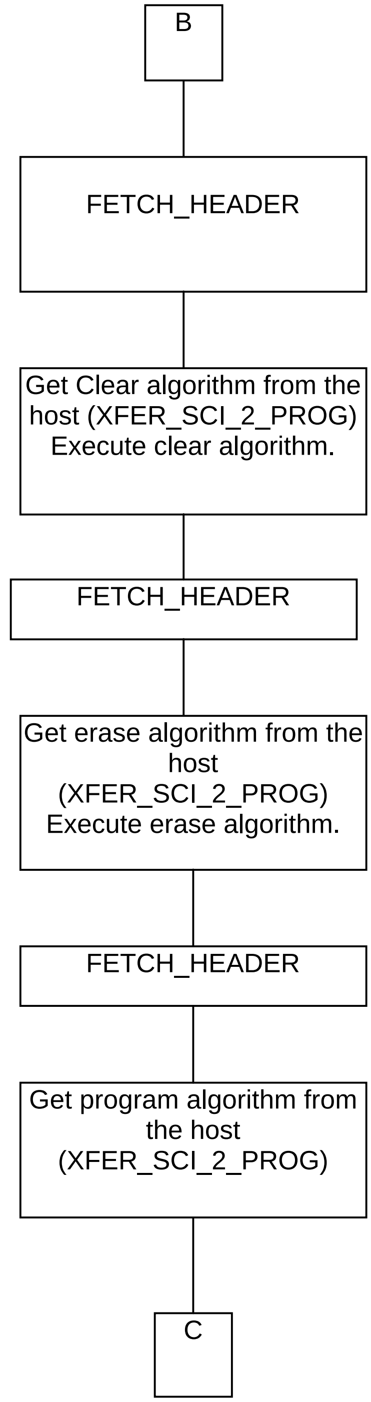


Figure 4 Flowchart for the Serial Boot Loader (contd.).

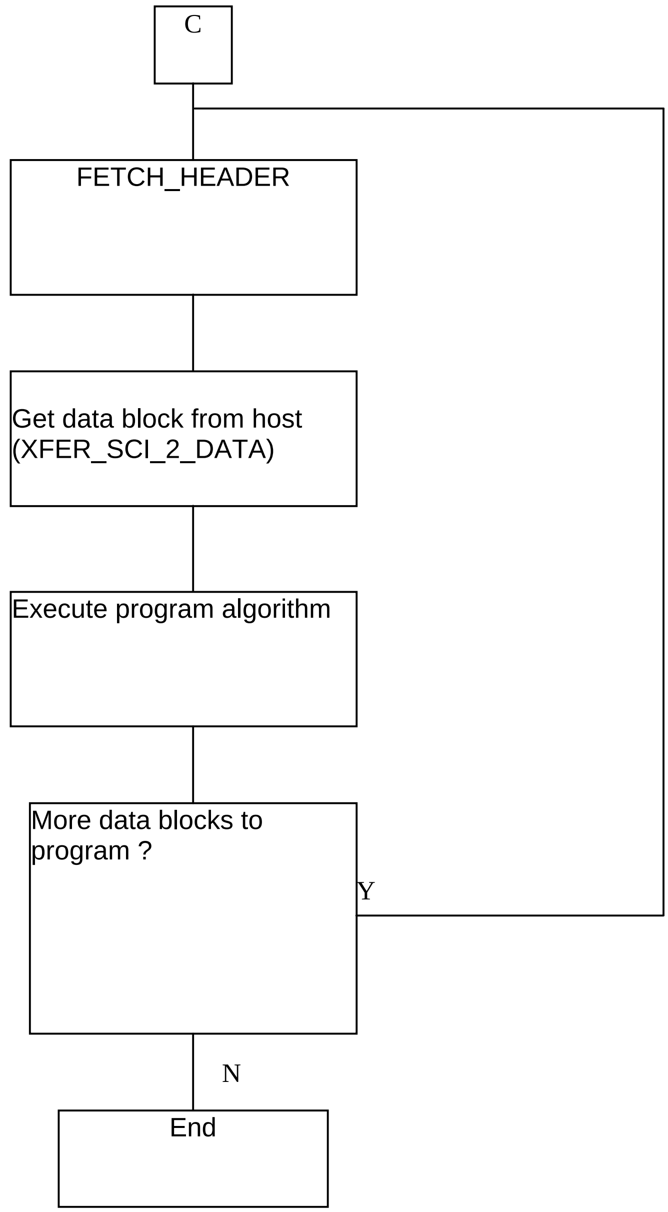
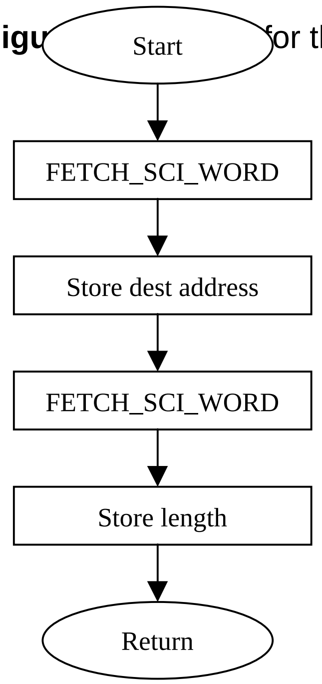
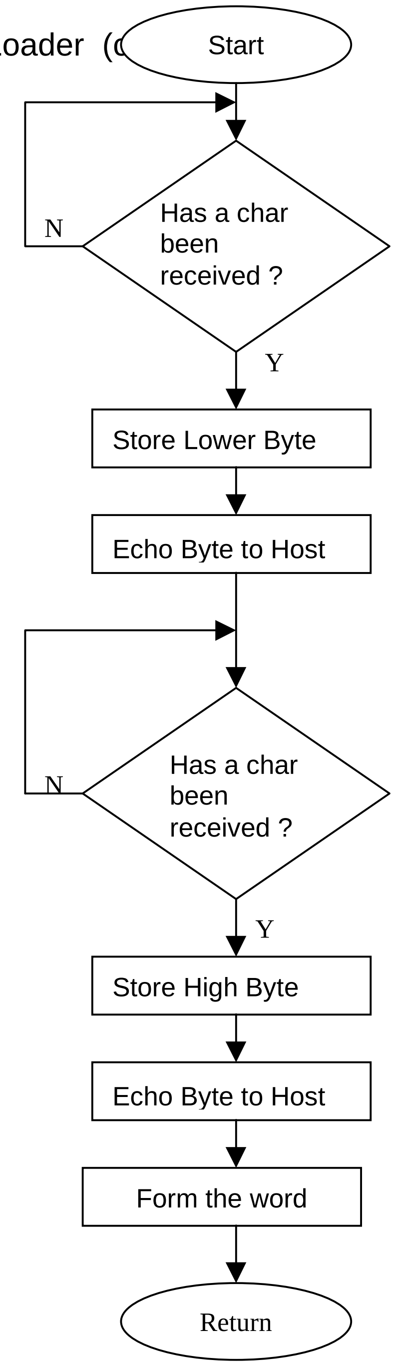


Figure 5 Flowchart for the Serial Boot Loader (contd.).

Figu... for the Serial Boot Loader (c...



(a) Flow chart for FETCH_HEADER



(b) Flow chart for FETCH_SCI_WORD

4.0 Assembly Code Listing

```
;*********************************************************************
; File Name:      SFLSH_B0.ASM
; Project:        F240 Serial Boot loader
;
; Description:    Serial Boot loader using an Intelligent Windows based
;                 PC Host communicating via the SCI on the target F240.
;                 The Serial Boot Loader Receives clear, erase and
;                 program algorithms from the host as well as data
;                 blocks containing user code. These blocks are
;                 programmed into flash.
;*********************************************************************
;
;Conditional assembly directives
USER_VERSION      .set  1           ;Set to 1 if for User version
                                    ;Set to 0 if for PE version

;---------------------------------------------------------------------
; 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


            .include    C240app.h
```

```
;Miscellaneous
BUF_SADDR   .set  0330h       ;Start address for Data buffer
WSGR        .set  0FFFFh
USER_START  .set  040h        ;Start of User's code
VBR_MAX     .set  09h         ;# times valid char needs to be received
CRC_MAX     .set  03h         ;# retries at each PLL setting before
giving up.

;----------------------------------------------------------------------
; 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

;----------------------------------------------------------------------
; 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, WD_KEY
            SPLK  #0AAAAh, WD_KEY
            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 - NOTE: NO reset vector!
;======================================================================
            .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
            CLRC  XF                ;set XF low (used to assert CTS on
                                    ;EVM 0nly)


            POINT_PF1
            SPLK  #006Fh, WD_CNTL         ;Disable WD if VCCP=5V
            KICK_DOG

            B     UART_INIT

;PLL & BAUD param table
PARAM_TBL   .word 0060h       ;PLL x 1 mode
            .word 0020h  ;20MHz     Baudrate value for BRR
            .word 00AAh       ;PLL x 1.5 mode
            .word 001Ch  ;12MHz
            .word 00B1h       ;PLL x 2.0 mode
            .word 0020h  ;10MHz
```

```
            .word 00CCh       ;PLL x 2.5 mode
            .word 0020h  ; 8MHz
            .word 00D2h       ;PLL x 3.0 mode
            .word 001Ch  ; 6MHz
            .word 00E3h       ;PLL x 4.0 mode
            .word 0020h  ; 5MHz
            .word 00EDh       ;PLL x 4.5 mode
            .word 001Ch  ; 4MHz
            .word 00E4h       ;PLL x 5.0 mode
            .word 001Ch  ; 3.58MHz
            .word 00F5h       ;PLL x 9.0 mode
PARAM_TBL_END     .word 001Ch  ; 2MHz

;======================================================================;
Uart Initialisation
;======================================================================
UART_INIT:
      ;Copy Parameter table to B1
            POINT_B1
            MAR   *, AR1
            LAR   AR1, #BUF_SADDR
            LACC  #PARAM_TBL_END
            SUB   #PARAM_TBL
            SACL  GPR1
            LACC  #PARAM_TBL
            RPT   GPR1
            TBLR  *+

            SPLK  #0h, VBR_CNTR     ;Clear valid baud rate counter
            SPLK  #0h, CHAR_RETRY_CNTR    ;Clear retry counter
            SPLK  #0FFFFh, DELAY

            POINT_PF1
            SPLK    #0017h, SCI_CCNTL ;One stop bit,No par,8 chr/word
            SPLK    #0013h, SCI_CNTL1 ;Enable TX, RX, internal SCICLK
                                      ;Disable RX ERR, SLEEP, TXWAKE
      ;Enable TXD & RXD pins
            SPLK    #0022h, SCI_PORT_C2

      ;Disable RX & TX INTs
            SPLK    #0000h, SCI_CNTL2

;======================================================================
;Baudrate lock protocol with Host
;======================================================================
      ;Set PLL to x1 mode initially
            LAR   AR1, #BUF_SADDR
UI00        CALL  SET_CLOCK

UI01        BIT   SCI_RX_STAT, 9    ;Test RXRDY bit
            BCND  UI01, NTC         ;If RXRDY=0,then repeat loop
            LACC  SCI_RX_BUF        ;First byte is Lo byte

      ;Check if Char is as expected
CHECK_CHAR  AND   #0FFh             ;Clear upper byte
            SUB   #00Dh             ;Compare with "CR"
            BCND  BAUD_RETRY, NEQ

INC_VBRC    POINT_B1
            LACC  VBR_CNTR          ;Inc VBR counter
            ADD   #1h
            SACL  VBR_CNTR
            SUB   #VBR_MAX          ;Is VBR counter > max value ?
            POINT_PF1
            BCND  UI01, NEQ         ;No! fetch another char
```

```
SND_ECHO     LACC  #0AAh              ;Yes!
             SACL  SCI_TX_BUF         ;Indicate to Host Baudrate locked
             B     FLSH_INIT

BAUD_RETRY   POINT_B1
             SPLK  #0h, VBR_CNTR
             LACC  CHAR_RETRY_CNTR    ;Inc CRC counter
             ADD   #1h
             SACL  CHAR_RETRY_CNTR
             SUB   #CRC_MAX           ;Is CRC > max value ?
             BCND  INC_TBL_PTR, GEQ   ;Yes! try next baudrate
             POINT_PF1
             B     UI01               ;No! fetch another char

             INC_TBL_PTR
             MAR   *+,AR1
             SPLK  #0h, CHAR_RETRY_CNTR
             B     UI00

;=======================================================================
;Init Flash parameters
;=======================================================================
FLSH_INIT    POINT_B1
             SPLK  #0h, SEG_ST        ;Start addr of Flash array
             SPLK  #3FFFh, 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
                                      ; F240 only has Array 0 present
             B     XFER_KERNEL


;=======================================================================
; Routine Name: S E T _ C L O C K          Routine Type: SR
;
; Note:     Assumes AR1 is pointer to PARAM_TBL in Data mem
;
;=======================================================================S
ET_CLOCK:
             POINT_PF1
      ;Configure PLL for
             SPLK  #0041h,PLL_CNTL1  ;Disable PLL first
             LACC  *+
             SACL  PLL_CNTL2
             SPLK  #0081h,PLL_CNTL1  ;Enable PLL
             SPLK  #40C0h,SYSCR      ;CLKOUT=CPUCLK

      ;Set the Baud Rate
             SPLK    #0000h, SCI_HBAUD
             LACC  *
             SACL  SCI_LBAUD

      ;Relinquish SCI from Reset.
             LACL    SCI_CNTL1
             OR      #20h
             SACL    SCI_CNTL1
             RET

;=======================================================================
;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              ;Stay here until reset.

;======================================================================
; 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
            POINT_PF1
FSW0        BIT   SCI_RX_STAT, 9    ;Test RXRDY bit
            BCND  FSW0, NTC         ;If RXRDY=0,then repeat loop
            LACC  SCI_RX_BUF        ;First byte is Lo byte
            SACL  SCI_TX_BUF        ;Echo byte back
            AND   #0FFh             ;Clear upper byte

FSW1        BIT   SCI_RX_STAT, 9    ;Test RXRDY bit
            BCND  FSW1, NTC         ;If RXRDY=0,then repeat loop
            NOP                     ;TEST ONLY
            ADD   SCI_RX_BUF,8      ;Concatenate Hi byte to Lo
            SFL                     ;used because 7 is max in SACH
            SACH  SCI_TX_BUF,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   POINT_PF1
            SACL  SCI_TX_BUF        ;Transmit byte to host.
            POINT_B1
            RET

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