TMS320LF240x Flash Programming

Serial Port Flash Programming Utility

Texas Instruments

Table of Contents

Table of Figures

1. Introduction

This document describes the Serial Asynchronous Port based Flash Programming Utility for the TMS320LF240x DSP Controllers. This utility leverages the Boot ROM on these DSP controllers to provide a stand alone flash programming capability independent on the previous contents of the flash. The serial port flash programming utility can be applied to in-system programming for the DSP controller. The Serial Flash Programming Utilities share the core flash programming algorithms with the JTAG Based Flash Programming Utilities. Another major difference from the F24x Serial Programming is that the RAM resident kernel is not fixed, copied from flash. It is downloaded run-time, resulting in added flexibility.

The TMS320LF240x devices have a on-chip asynchronous serial port (SCI - serial communication interface). This chip communicates with standard RS232 compatible devices, via external level translation hardware. The programming does not tie up the serial communication port, this can be used for normal operation at other times.

1.1. Overview

The Boot ROM on the 'LF240x devices has two loaders in it. The serial flash programming utility requires that upon device reset the Boot_EN / XF and the SPISIMO pins be pulled low. This vectors the control of the device to the Boot ROM Asynchronous port loader. A baud rate match protocol is then followed, synchronizing the communications port on the target and the host. Once the communications are up-and-running , the host downloads a kernel to the target. This kernel has the sequencing built into it, besides containing the interface routines to the asynchronous serial port. This kernel will from this point onwards, be the sole interface between the target flash algorithms and the host, the Boot ROM will not be accessible during the flash programming process, and the routines in the ROM cannot be executed. The kernel will be described in Sections 2.2 and 2.3. Once the kernel is initialized correctly, the Clear, Erase and Program Algorithms are downloaded and executed.

2. Operation

A step by step description of the process appears below.

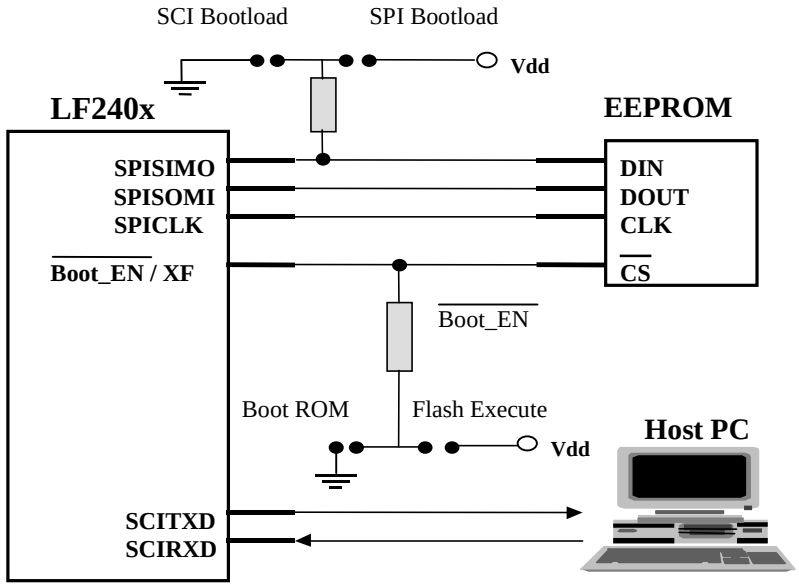
2.1. DSP Controller Initialization

The device is placed in Microcontroller mode and control transferred to the Boot ROM (See Figure 1). To do this the following things are required:

Microcontroller Mode The ‘LF24xx device must be placed in microcontroller mode, by pulling the MP/MC* pin LOW.

Boot ROM Loader Invocation The boot loader is invoked by pulling the BOOT_EN* / XF pin low through a resistor, prior to device reset. This causes the control to transfer to the boot load program located in the on-chip ROM. At reset time internal logic takes a 'snapshot' of this pin and if this pin is a low level then the Boot ROM appears in the memory map as shown in Figure 2.

Otherwise the on-chip flash memory is enabled and the program counter begins execution at 0x0000. It is suggested that this pin be driven from a jumper through a resistor, allowing control whether the processor enters the boot loader or commences normal execution from



internal program memory. Alternately this pin may be controlled from a host processor, allowing it to control the boot sequence of the DSP. The resistor must be present, since the XF pin is an output at all other times.

SPI or SPI Selection The boot loader code selects the source of the incoming code depending on the state of the SPISIMO pin, on the device.

Figure 1. Serial Flash Utility Device Initialization

The code takes a snapshot of this code after being invoked, and determines which loader (SPI or SCI) to invoke based on the status of this pin.

- If SPISIMO/IOPC2 is pulled low, an SCI transfer is commenced, otherwise
- If SPISIMO/IOPC2 is pulled high an SPI transfer is commenced

For the Serial port flash programming the first option, to commence SCI transfer must be selected.

At this point the control is transferred into the Boot ROM SCI control program. This program resident in the Boot ROM enables the transfer of the kernel into RAM on the device.

Communications Initialization The baud rate over the communication link is always 38400 bps. The baud rate protocol is necessary because the 'LF240x device may be operated at different speeds. The underlying assumption for the baud rate matching is that the device is clocked at a clock frequency belonging to a given set. This set is determined by the parameters in ROM. The host is required to send 'probe' characters, with the hexadecimal value 0x0D. The target listens in on the serial port, at the set speeds, in succession. Every time a character is detected, it is compared to 0x0D. If more than three characters do not match, the target tries a new baud rate. If the baud rate is correct and the character matches 0x0D, then the target expects to receive nine correct characters back to back. If any other character is received the baud match fails. Once the nine chars are received correctly then the target sends an acknowledge character. Once the acknowledge character is sent, from this point on each and every character is bounced back to the host to ensure data transfer integrity. All the communications are with 8 bits/char. 1 stop bit and no parity. The communications initialization protocol is flowcharted in **Figure 4**. Once the communications are locked, the data transfer commences.

2.2. Kernel Transfer

The kernel is transferred over the communications channel split into bytes. The LS Byte is transferred first, followed by the MS Byte. The data packet format is shown below:

Host sends:

- Start address : One Word (16 bits) , split into bytes, LS Byte first.
- Length (i.e. number of words to transfer): One Word (16 bits) , split into bytes, LS Byte first.
- Actual kernel code, split into bytes, LS Byte first.

As part of the initialization process, once the kernel is transferred into RAM, the kernel sends a 'error code' to indicate its initialization status. A 0x0 code indicates success whereas a failure to return any code or the return of an incorrect code would cause the host to abandon the transfer.

Note: Once the kernel is initialized it is possible to change the communication parameters, however the custom kernel must verify that the last transfer on the SCI is complete before attempting to change the parameters. Additionally the host must take this change of parameters into account.

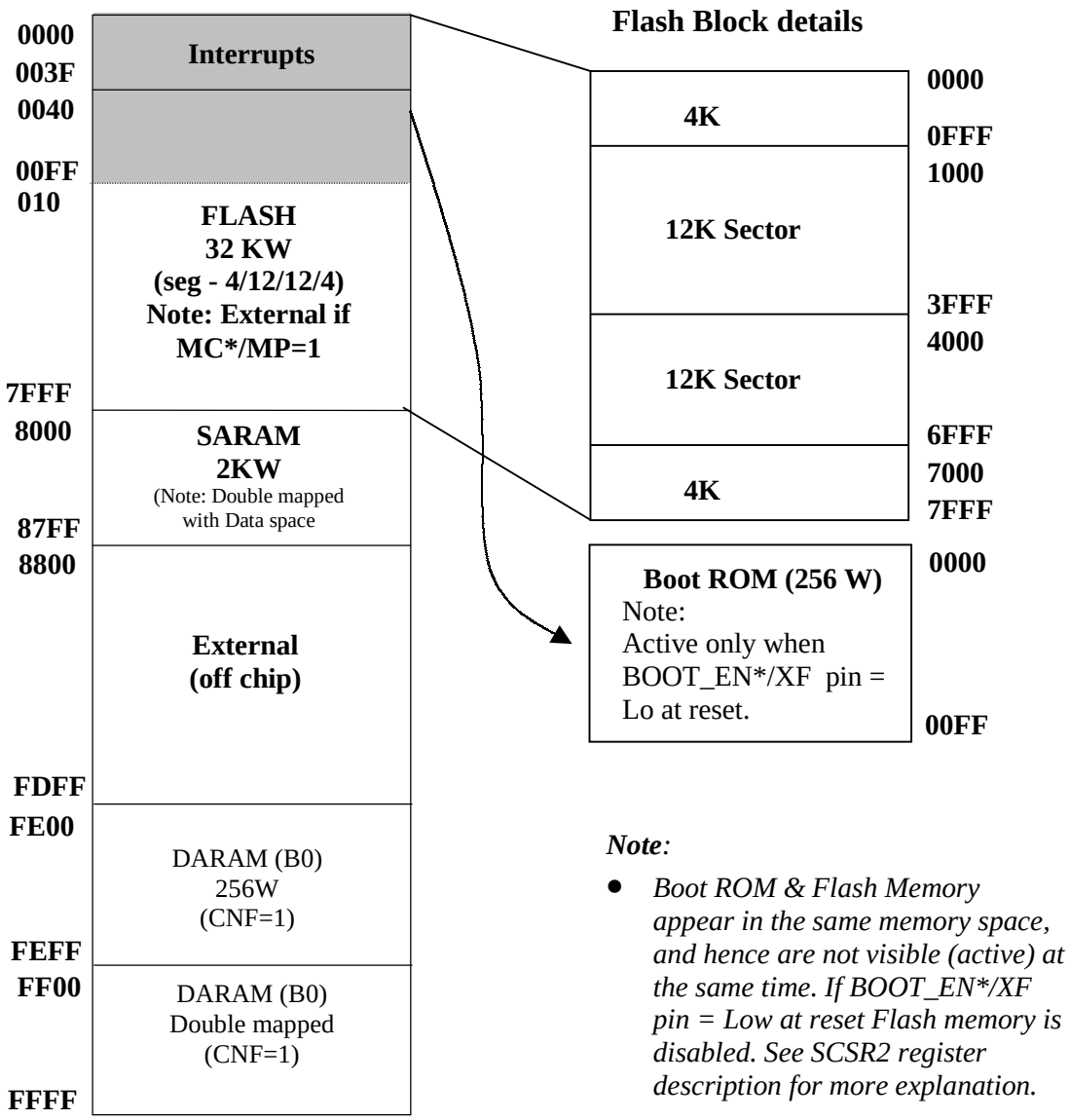


Figure 2. Memory Maps for the LF 24xx Devices in Microcontroller Mode.

2.3. Kernel Operation

Once the kernel is successfully initialized, the kernel controls the sequencing of the transfer over the link. The TI kernel requires the algorithms to satisfy certain conditions/guidelines. These are:

a. The algorithm must be less than 0x100 words, and must be assembled/linked to execute between 0xfe00 and 0xfeff. In addition the entry point must be at 0xfe00.
b. The algorithm must return status in the variable ERROR_FLAG. See SVAR.H in the algos\include directory for details.
c. The algorithm must return control by means of a return instruction. Also it should not destroy any variables the kernel uses. The kernel variables are in the file indicated in (b).
d. The algorithm should also not interfere with the communications port, this may interrupt the protocol.
e. The kernel expects the algorithms to be Clear, Erase and Program in that sequence. The clear, erase and program algorithms are described in step (5),(6) and (7).
f. The kernel must of course be located at an address with valid RAM available. No check is made to see if there is RAM available, so the kernel must be assembled and linked to ensure that this happens correctly.

2.4. Clear

The clear algorithm is the first algorithm downloaded to the target. It performs the pre-condition operation on the flash, clearing(setting to zero) all the bits in the main and secondary arrays. This readies the flash for erase. Upon a successful clear the algo returns a zero, otherwise non-zero.

2.5. Erase

Erase is the downloaded next. This erases all four sectors of the flash. The erase algorithm also performs a compaction check on the flash, compacting any depleted columns. Upon a successful erase the algo returns a zero, otherwise non-zero.

2.6. Program

Following the erase and clear the programming algorithm is downloaded next. However the programming algorithm is not executed immediately, instead first block of programming data is downloaded. Once the programming data is available a call is made to the programming algorithm, with the destination address and the length are available. The programming algo reads in data from the data buffer and programs the data into the flash array. The programming operation puts a pattern of zeros into the flash array.

2.7. Repeat program / Exit Programming

Once the programming operation for one block is completed the programming algorithm returns a status in ERROR_FLAG. Once the kernel regains control, it passes the status to the host. The host, upon receiving a success status, send a "last block flag" - one word. The target looks at this and either returns to receive one more program data block, or terminates in a blind loop.

The TI kernel terminates in a blind loop, however if the system allows, a custom kernel may allow the control to return to the newly programmed code in the flash. It would also be possible to extend the kernel to run further algorithms, after programming, performing other functions.

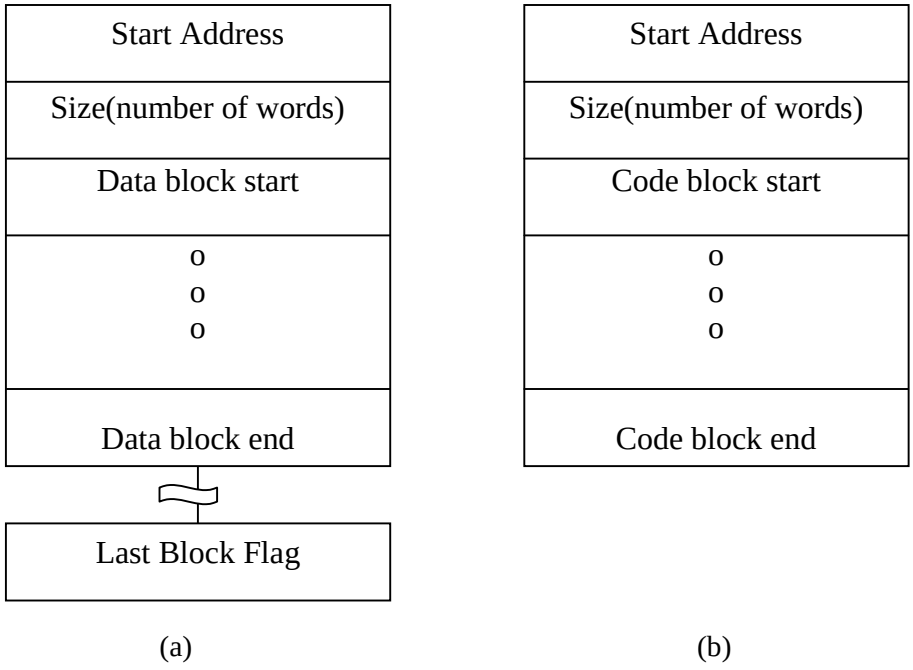


Figure 3. Transfer Packet Formats for the Programming data blocks (a) and the three operational algorithms(b).

3. Working with the Serial Programming Utility

3.1. Preparing code for programming into flash.

To use the serial port flash programming utility compile, assemble and link your code to run out of flash. Prepare your COFF file (flashcode.out) for programming the flash. This COFF file can contain up to 32K words for the 'F24xx. The only restriction is that the COFF file must not include anything other then the code to be programmed in the flash. Never include data sections in the COFF file that will be used to program the flash. For more information on COFF and working with sections, refer to the TMS320C1x/C2x/C2xx/C5x Assembly Language Tools User's Guide.

3.2. Setting up the programming utilities

The programming utilities are distributed as a compressed file. To set up the tool unzip this file. Ensure that the directory structure is restored. For correct operation this required.

3.3. Invoking the serial loader

Verify that the serial link on your target is functioning by means of echoing characters back from a terminal program. By connecting RX and TX together on the target side for example) Once this is done restore the connections.

Choose loader1.bat or loader2.bat depending on whether your serial cable is connected to COM1 or COM2. Copy this batch file to another batch file say, prog.bat. Edit this file to point the command line parameter for spf24xb?.exe to your hex file. Once this is done, run this batch file to invoke the loader

3.4.

4. Configuring the target clock frequency

For the utilities as released the target clock frequency MUST be 30MHz.
The serial boot ROM loader will lock at other clock frequencies as well, but the flash algorithms must be first configured for the NEW frequency first. Programming the flash using wrongly configured algorithms can cause permanent damage to the target device and/or undefined operation.

4.1. Adjusting Clock Frequency

The programming algorithms for the 'F240x on-chip flash include software delays that must be adjusted according to the instruction rate of the target device. This chapter describes how to modify the programming utility for use with different clock frequencies.

WARNING! If the design will be using a variable CLKOUT, (i.e. CLKOUT will be varied by the application) then the flash should be erased at the highest possible CLKOUT rate. This is important to ensure adequate read-back margin throughout the life of the application. If for instance the CLKOUT may be any of 5,10,20 MHz, then the flash must be erased using the CLKOUT at 20Mhz.

If the frequency of CLKOUT will be different for some reason, eg. the CLKIN is not 7.5 MHz, then it is necessary to re-configure the flash programming utilities to take this into account. To do this following things must be accomplished.

a. **PLL Multiplier Configuration**

b. **Scaling the Timing Parameters.**

c. **Generating a timing set.**

d. **Re-building the algorithms.**

4.1.1. (NOT) Configuring the PLL Multiplier Ratio

The configuration set in the algos\include\var.h is ignored, since the utilities depend on the Boot ROM loader to configure the PLL.

4.1.2. Scaling the Timing Parameters

Timing parameters are selected based on the clock frequency the CPU is running at. A few timing sets are distributed with the tools. Each set of timings is contained in a file called timings.xx. For example the file containing loop timings calibrated at 28MHz is called timings.28.In STEP 2 in VAR.H choose the appropriate timing set.

4.1.3. Generating a timing set

Lets assume that a system needs timings to be calibrated at 28 MHz. For this purpose the Excel worksheet timings.xls is provided. To generate a new set of timings open this file in Microsoft Excel, and see the instructions in the worksheet.

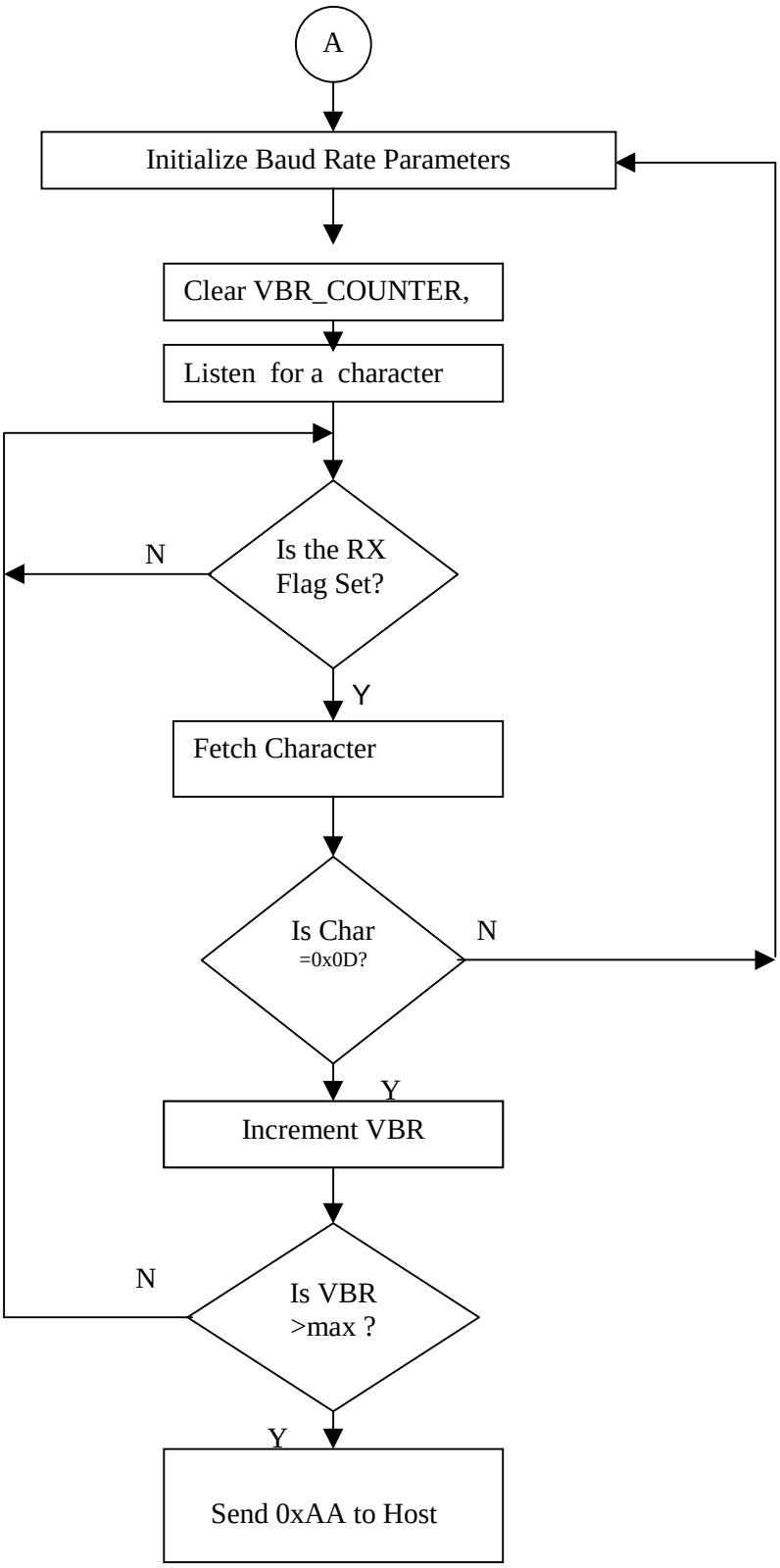


Figure 4. Flowchart for the ROM Bootloader SCI Protocol Communications Synchronization.

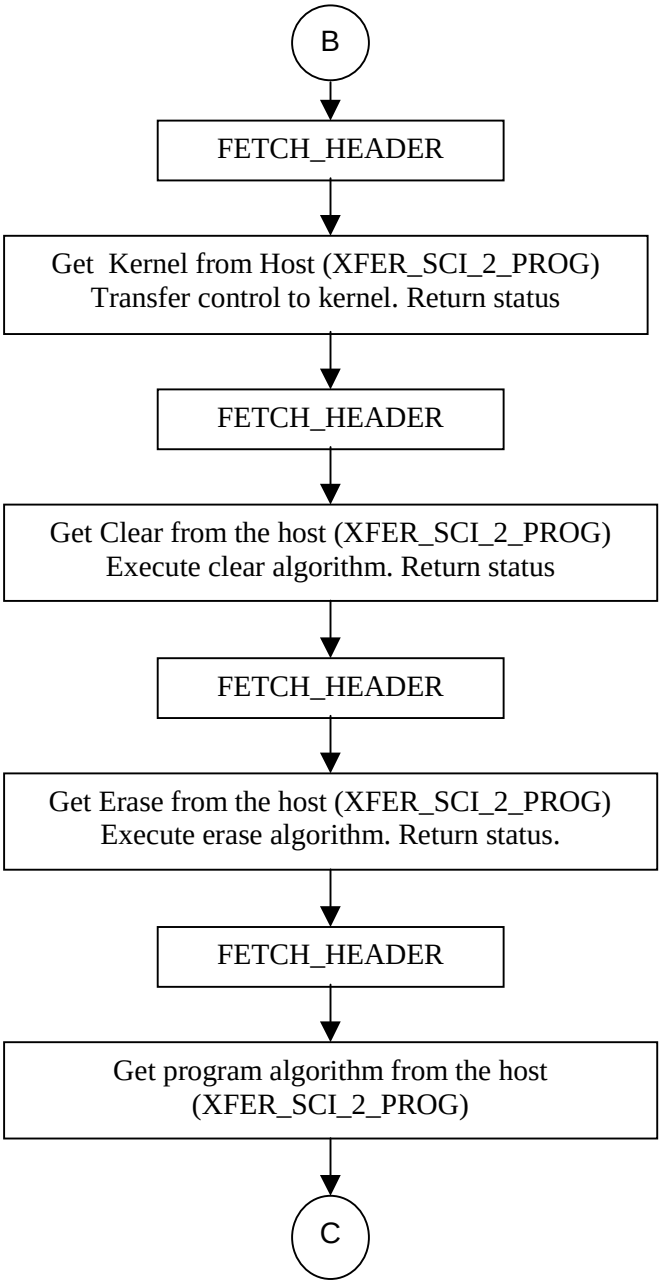


Figure 5. Flowchart for Serial Flash Programming Utility Algorithm execution sequence.

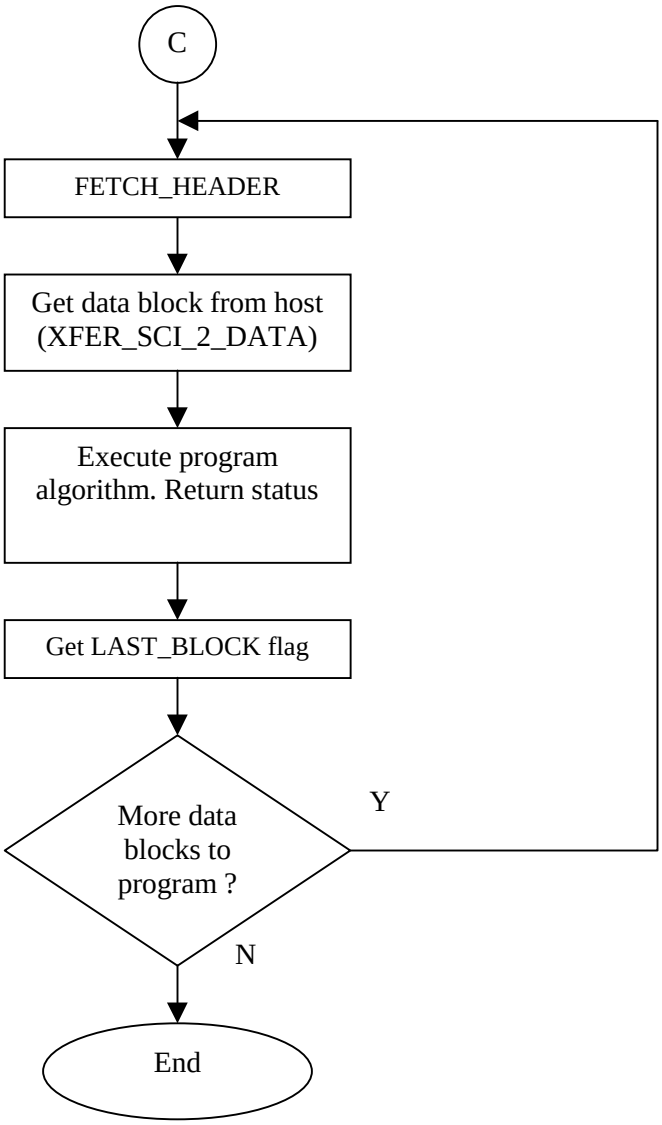


Figure 6. Flowchart for Serial Flash Programming Utility Programming Sequence.

```
;***************************************************************************
; File Name:   Kernel.ASM  Build 2.
;
; Project:     F24XX Serial Boot Loader
; Originator:  DSP Digital Control Systems Group, Houston (Texas Instruments)
;
; Target Sys:  F240x.
;***************************************************************************
               .include       ..\include\svar.h
               .include       ..\include\x24x.h

ALGO_START     .set  0fe00h
               .def           ALGO_START

;Miscellaneous
BUF_SADDR       .set    0328h           ;Start address for Data buffer
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

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

;===========================================================================
; This is the entry point and will be at 0x8000.
;===========================================================================
               .text
START:         LDP     #WDCR>>7
               SPLK    #006Fh,WDCR    ;Disable WD
;===========================================================================
;Init Kernel
;===========================================================================
FLSH_INIT      POINT_B1
```

```
                SPLK    #0,ERROR_FLAG
;=============================================================================
                LACC    ERROR_FLAG      ;Send Zero error to host to indicate to
                CALL    SEND_CHAR       ;host kernel successfully initialized.
;=============================================================================
        ;Load & Execute CLEAR
M00             CALL    XFER_SCI_2_PROG
                CALL    RUN_ALGO
;=============================================================================
        ;Load & Execute ERASE
M01             CALL    XFER_SCI_2_PROG
                CALL    RUN_ALGO
;=============================================================================
                ;Load & Execute PROG
M02             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    RUN_ALGO
;=============================================================================
                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                    ;
;=============================================================================
; 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:
                CALL    FETCH_HEADER
                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
               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


RUN_ALGO       CALL   ALGO_START
               LACC   ERROR_FLAG
               CALL   SEND_CHAR      ;Indicate to host Clear finished.
               RET
;=============================================================================
               .end
```