Application Note

Self-Programming Examples

For NEC Electronics 78K0/Kx2 Microcontrollers

Document No. U18291EU1V0AN00
©July 2006. NEC Electronics America, Inc.
All rights reserved.

The information in this document is current as of July 2006, however the information is subject to change without notice. For actual design-in, refer to the latest publications of NEC Electronics data sheets or data books, etc., for the most up-to-date specifications of NEC Electronics products. Not all products and/or types are available in every country. Please check with an NEC Electronics sales representative for availability and additional information.

No part of this document may be copied or reproduced in any form or by any means without prior written consent of NEC Electronics. NEC Electronics assumes no responsibility for any errors that may appear in this document.

NEC Electronics does not assume any liability for infringement of patents, copyrights or other intellectual property rights of third parties by or arising from the use of NEC Electronics products listed in this document or any other liability arising from the use of such NEC Electronics products. No license, express, implied or otherwise, is granted under any patents, copyrights or other intellectual property rights of NEC Electronics or others.

Descriptions of circuits, software and other related information in this document are provided for illustrative purposes in semiconductor product operation and application examples. The incorporation of these circuits, software and information in the design of customer's equipment shall be done under the full responsibility of customer. NEC Electronics no responsibility for any losses incurred by customers or third parties arising from the use of these circuits, software and information.

While NEC Electronics endeavors to enhance the quality, reliability and safety of NEC Electronics products, customers agree and acknowledge that the possibility of defects thereof cannot be eliminated entirely. To minimize risks of damage to property or injury (including death) to persons arising from defects in NEC Electronics products, customers must incorporate sufficient safety measures in their design, such as redundancy, fire-containment and anti-failure features.

NEC Electronics products are classified into the following three quality grades: “Standard”, “Special” and “Specific”.

The "Specific" quality grade applies only to NEC Electronics products developed based on a customer-designated “quality assurance program” for a specific application. The recommended applications of NEC Electronics product depend on its quality grade, as indicated below. Customers must check the quality grade of each NEC Electronics product before using it in a particular application.

"Standard": Computers, office equipment, communications equipment, test and measurement equipment, audio and visual equipment, home electronic appliances, machine tools, personal electronic equipment and industrial robots.

"Special": Transportation equipment (automobiles, trains, ships, etc.), traffic control systems, anti-disaster systems, anti-crime systems, safety equipment and medical equipment (not specifically designed for life support).

"Specific": Aircraft, aerospace equipment, submersible repeaters, nuclear reactor control systems, life support systems and medical equipment for life support, etc.

The quality grade of NEC Electronics products is “Standard” unless otherwise expressly specified in NEC Electronics data sheets or data books, etc. If customers wish to use NEC Electronics products in applications not intended by NEC Electronics, they must contact NEC Electronics sales representative in advance to determine NEC Electronics 's willingness to support a given application.

Notes:

1. "NEC Electronics" as used in this statement means NEC Electronics Corporation and also includes its majority-owned subsidiaries.
2. "NEC Electronics products" means any product developed or manufactured by or for NEC Electronics (as defined above).

M8E 02.10

Revision History

Date	Revision	Section	Description
July 2006	—	—	First release

Contents

1. Introduction 7
   - 1.1 Overview 7
   - 1.2 Self-Programming Features 7
   - 1.3 Requirements 8
2. Microcontroller Flash Architecture 8
   - 2.1 Flash Blocks 8
   - 2.2 Boot Clusters 9
   - 2.3 Flash Words 9
   - 2.4 Flash Banks 9
   - 2.5 Boot-Swap Feature 9
   - 2.6 Interrupt Handling 9
   - 2.7 Hardware Requirement (FLMD0 Pin) 9
3. Sample Self-Programming Libraries 11
   - 3.1 Flow of Self-Programming When Using Sample Library 12
4. Self-Programming Example 13
   - 4.1 Introduction of Bootloader 13
   - 4.2 Elements of Bootloader 14
     - 4.2.1 Start-up Signal 14
     - 4.2.2 Execution Signal 14
     - 4.2.3 Transferring Code 14
     - 4.2.4 Flash Self-Programming 14
     - 4.2.5 Transferring Control to a Valid Application Program 15
   - 4.3 Bootloader Configuration and Operation 15
     - 4.3.1 Demonstration Platform 15
     - 4.3.2 Bootloader Communication 15
     - 4.3.3 Bootloader Operation 16
     - 4.3.4 Boot prompt for Loading Application 17
     - 4.3.5 Loading Application 17
     - 4.3.6 Receiving Data 17
     - 4.3.7 Processing the Received Data 17
     - 4.3.8 Blank Checking and Erasing 18
     - 4.3.9 Programming 18
     - 4.3.10 Verifying 18
     - 4.3.11 Storing Valid-Application Checksum 19
     - 4.3.12 Executing Application 19
     - 4.3.13 Flash Self-Programming Errors 19
     - 4.3.14 Incorrect Line-Checksum Error 20
     - 4.3.15 Boot Area Overwrite Error 20
     - 4.3.16 Boot Swapping 20
5. Developing Self-Programming Code Using NEC Electronics Tools 22
   - 5.1 Code Structure 22
   - 5.2 Development Flow 22
   - 5.3 Tool Setup 23
     - 5.3.1 Procedure to Generate Boot Code 24
     - 5.3.2 Procedure to Generate Application (Flash) Code 24

6. Sample Code 26

7. Appendix A — Flash Self-Programming Routines 28

7.1 bloader.c 28
7.2 epvdata.c 32
7.3 getdata.c 39
7.4 uart6.c 47
7.5 adcbu6.c 51
7.6 dicebu6.c 56
7.7 UART6.h 63
7.8 epvdata.h 64
7.9 getdata.h 66
7.10 option.asm 78

1. Introduction

The 78K0/Kx2 family of microcontrollers (MCUs) can program their own flash memory. This application note provides an overview of the flash self-programming capability, using the µP78F0537 microcontroller as the target device.

To facilitate development of applications requiring self-programming capabilities, NEC Electronics has developed a sample library implementing all the necessary functions. You can obtain the self-programming sample library in binary format, and sample source code is provided for reference in the Appendix of this application note. The sample code includes a bootloader program, which allows you to download code to an MCU, as well as self-programming application-code examples.

For additional information about the 78K0/Kx2 flash self-programming process, refer to the *78K0/Kx2 Flash Memory Self-Programming User's Manual* (document U17516EJxV0UM).

1.1 Overview

To program its flash memory, 78K0/Kx2 MCUs executes code located in hidden ROM. You can call these hidden functions to check for unused space (blank check) and to erase or write code to any memory location available in the MCU. To take advantage of these capabilities, your application program must include code to access the hidden firmware. Thus, you must pre-program the MCU's flash memory with this code in order to do self-programming.

1.2 Self-Programming Features

- The self-programming function takes advantage of NEC Electronics' single-voltage flash memory process, and requires only the MCU's V_{CC} power.
- Self programming works with all of the MCU's flash memory.
- You can input data through any MCU interface so long as the data passes through internal RAM.
- The minimum write unit is four bytes.
- The maximum write unit at one time is 256 bytes.
- The minimum erase or blank-check unit is one flash-memory block (1 KB).
- The MCU provides two 4-KB clusters that can store a bootloader program. A built-in boot-swap feature lets you update the bootloader itself.
- Interrupt acknowledgement is possible while executing major self-programming functions.
- Software and hardware conditions guard against accidental manipulation of flash memory.
- The CPU uses an 8-MHz internal oscillator when executing hidden-ROM functions.

1.3 Requirements

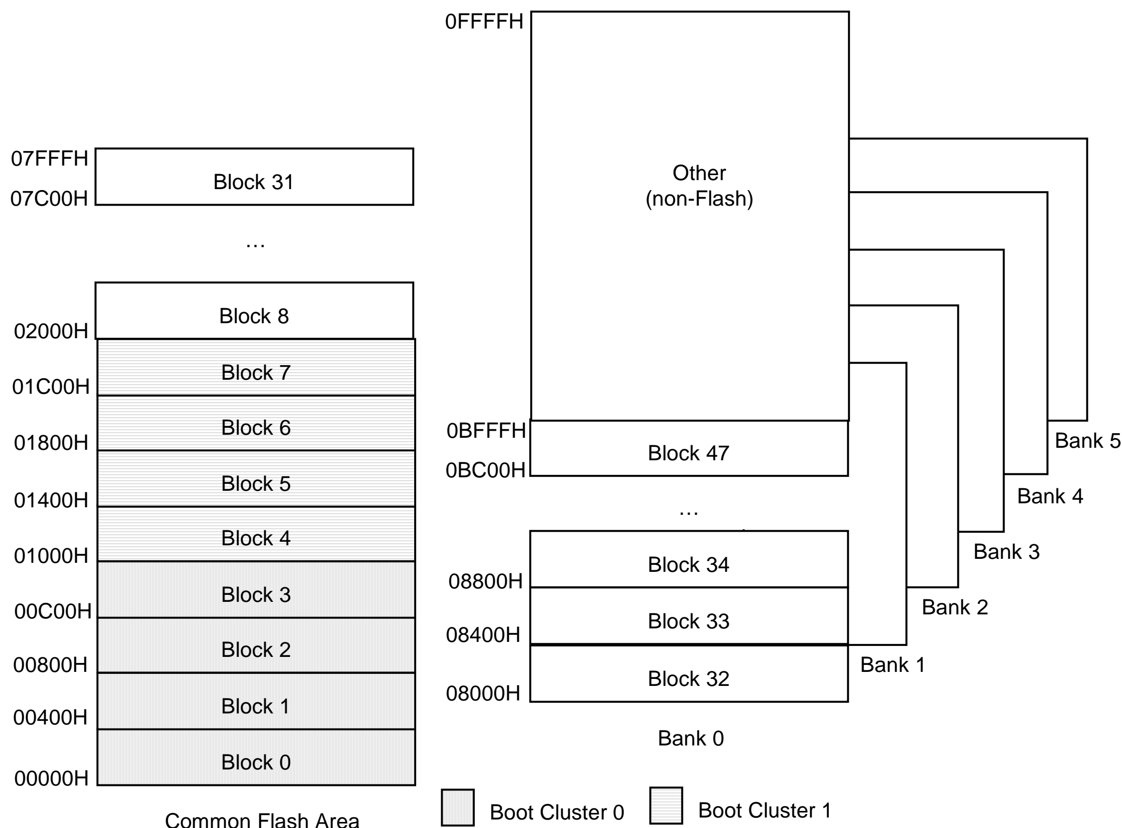
- Use of General Purpose Register Bank 3
- 100 bytes of RAM (entry RAM) for a work area for the hidden-ROM functions.
- Use of pin FLMD0 to set the MCU to self-programming mode
- From 4- to 256 bytes of RAM for a data buffer
- Maximum of 39 bytes of RAM for a hidden-ROM function stack
- Hidden-ROM functions called by an application program located between addresses 0000H and 7FFFH

2. Microcontroller Flash Architecture

2.1 Flash Blocks

As shown in the figure below, the µPD78F0537 MCU's flash memory is divided into 1-KB blocks. This is the smallest amount of memory that can be blank-checked, erased, or verified.

Figure 1. Flash Memory Structure of µPD78F0537



2.2 Boot Clusters

The first eight blocks of flash memory make up two 4-KB boot clusters (Boot Cluster 0 and Boot Cluster 1). These clusters work with the MCU's boot-swapping feature to allow a bootloader to update itself.

2.3 Flash Words

One four-byte word is the smallest amount of memory you can write into the data buffer, which holds a maximum of 64 words (256 bytes).

2.4 Flash Banks

The 78K0/Kx2 family includes products with flash memories ranging from 8 to 128 KB. Products with more than 60 KB of flash memory have a bank configuration. Those with 60 KB or less do not have banks.

For some self-programming operations, such as write, erase or blank-check, you must specify both bank and block numbers.

2.5 Boot-Swap Feature

The 78K0/Kx2 self-programming capability supports a boot-swap function that replaces the default boot-program area (Boot Cluster 0) even if the programming is interrupted by electrical noise or power loss. In a typical application with self-programming, Boot Cluster 0 contains bootloading code with all the necessary procedures to execute self-programming. To update the bootloader using self-programming, the bootloader programs its new version into boot cluster 1, then uses the boot-swap feature to have the CPU boot-up using the new code.

2.6 Interrupt Handling

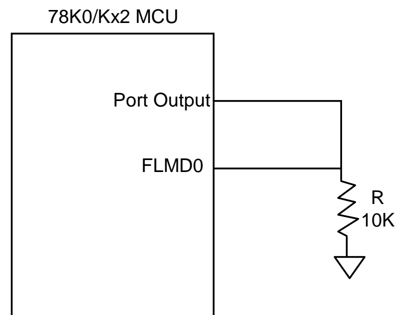
The MCU's interrupt logic can interrupt self-programming processes such as write or erase. However, unlike ordinary processing, the interrupt-response time includes a delay to complete processing of the self-programming library before acknowledging the interrupt. By reading the return value of the self-programming library, you can determine if you need to retry the procedure stopped by the interrupt.

2.7 Hardware Requirement (FLMD0 Pin)

The flash self-programming function uses a single-voltage process, requiring only the MCU's V_{CC} power. The application program manages the FLMD0 pin (which sets the MCU to self-programming mode) as part of the flash self-programming process. Pin FLMD0 normally is low but must be pulled high for programming. As shown in the figure below, you can tie FLMD0 to a port pin configured as an output and

to ground via a resistor. In this recommended configuration, you set FLMD0 high or low by setting or clearing the output-port pin.

Figure 2. FLMD0 Connection



3. Sample Self-Programming Libraries

Table 1. NEC Electronics Sample Self-Programming Library Routines

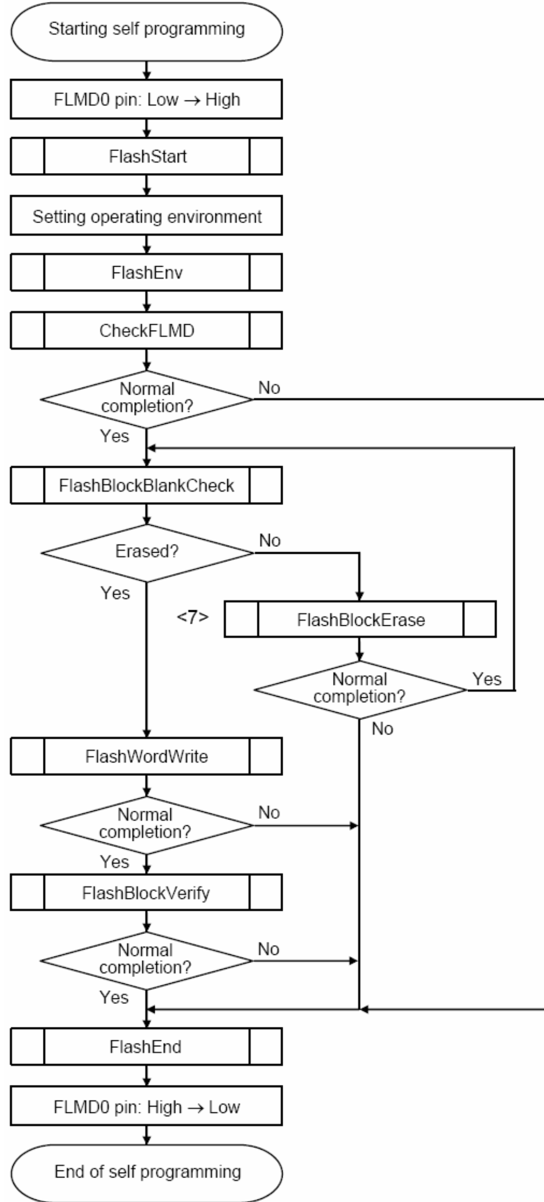
Library Name	Call Example (C language) / Call Example (assembly language)	Note
Self-Programming Start Library	FlashStart(); CALL !_FlashStart	1
Initialize Library	FlashEnv(&EntryRAM[0]); CALL !_FlashEnv	2
Mode-Check Library	Status = CheckFLMD( ); CALL !_CheckFLMD	3
Block-Blank Check Library	Status = FlashBlockBlankCheck(BlankCheckBANK,BlankCheckBlock); CALL !_FlashBlockBlankCheck	4
Block-Erase Library	Status = FlashBlockErase(EraseBANK, EraseBlock); CALL !_FlashBlockErase	5
Word-Write Library	Status = FlashWordWrite(&WordAddr, WordNumber,&DataBuffer); CALL !_FlashWordWrite	6
Block-Verify Library	Status = FlashBlockVerfy(VerifyBANK, VerifyBlock); CALL !_FlashBlockVerify	7
Self-Programming End Library	FlashEnd( ); CALL !_FlashEnd	8
Get-Information Library	Status = FlashGetInfo(&GetInfo, DataBuffer); CALL !_FlashGetInfo	9
Set-Information Library	Status = FlashSetInfo(SetInfoData) CALL !_FlashSetInfo	10
EEPROM-Write Library	Status = FlashEEPROMWrite(&WordAdder,WordNumber, &DataBuffer); CALL !_EEPROMWrite	11

Notes:

1. Declares start of self-programming
2. Initializes entry RAM
3. Checks FLMD0 voltage level
4. Checks erasing of specified block (1KB)
5. Erases specified library (1KB)
6. Writes 1- to 64-word data to specified address
7. Verifies specified block (1KB) (internal verification)
8. Declares end of self-programming
9. Reads flash information
10. Changes setting of flash information
11. Writes 1- to 64-word data to specified address (during EEPROM emulation)

3.1 Flow of Self-Programming When Using Sample Library

Figure 3. Self-Programming Flowchart



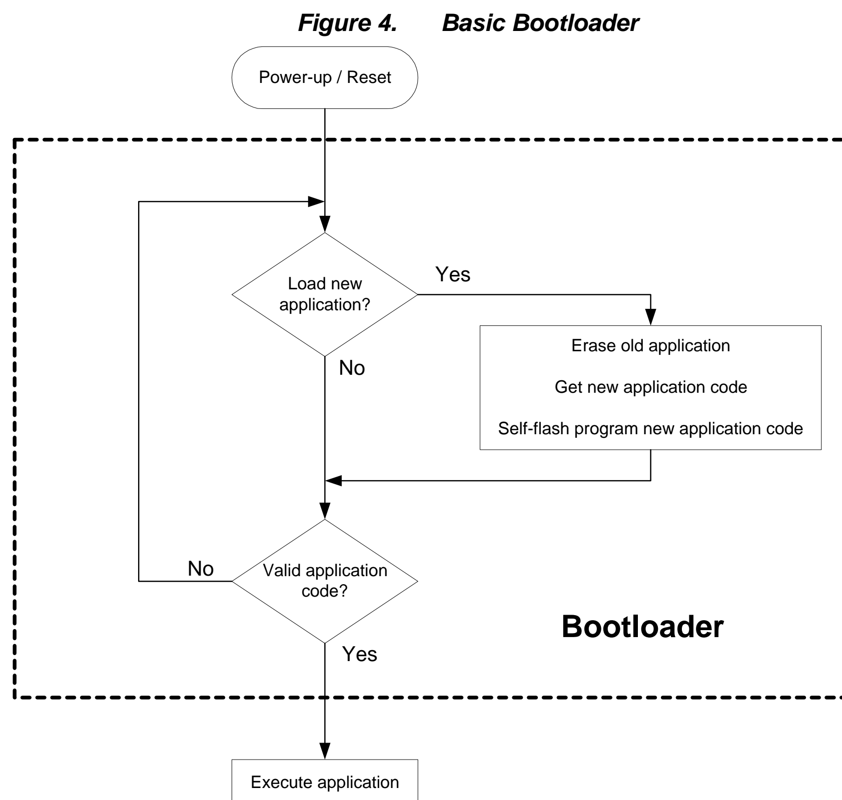
4. Self-Programming Example

This section describes the code developed by NEC Electronics America to demonstrate the main features of 78K0/Kx2 self-programming. This example uses the UPD78F0537 microcontroller and consists of a bootloader program, which allows you to update the contents of flash memory by downloading sample application code from a host PC.

4.1 Introduction of Bootloader

Boot code comprises the instructions that execute when the MCU starts up or boots. The term loading, in this case, refers to writing new application code into the MCU's flash memory. Below is the flowchart of a simple bootloader used to self-program the MCU's flash memory.

Figure 4. Basic Bootloader



A bootloader lets you update or replace application code without an external programmer, and makes it possible to update code remotely—over a phone line or Internet connection.

For example, if you have 5,000 MCU-based pay phones and the phones require a firmware update, the phone company's service person could manually reprogram the individual telephones—a time-consuming effort—or use a bootloader to reprogram all 5,000 phones remotely from a central location.

4.2 Elements of Bootloader

The sample bootloader consists of five main elements:

- ♦ A signal to start the bootloading process
- ♦ A signal to execute the bootloader
- ♦ Transfer of new code into the MCU
- ♦ Flash self-programming of the new code
- ♦ Transfer of control to a valid application program

4.2.1 Start-up Signal

If you have hundreds of vending machines connected to the Internet and want to update their firmware, you would need to generate a signal to trigger the MCUs to start the bootloading process. The signal could be an interrupt, a command byte sent over a serial channel, or something else that would cause the program to reset and run the bootloader code.

4.2.2 Execution Signal

Upon startup, the MCU loads a new application program or executes an existing one, depending on the external signal. You can program the signal to come from a port pin upon power-up, using this signal to control whether the MCU loads or executes. You could base the signal on a character received by the UART or a reading taken by an analog-to-digital (A/D) converter.

4.2.3 Transferring Code

You can transfer the programming data over an RS-232, I^2C, serial port or parallel port. Since the amount of data transferred typically exceeds the size of the MCU’s RAM, you need some provision to control the data flow. For an RS-232 serial port you could use a slow baud rate to give the MCU time to process the data and self-program without being overrun. Or you can use hardware handshaking, using clear to send (CTS) and request to send (RTS) lines to control the data flow. Another option is software handshaking using the XON/XOFF protocol.

You can format the code as you choose, but it must contain addressing information as well as checksums for error processing. Most applications use a standard such as Intel hexadecimal format.

4.2.4 Flash Self-Programming

Each time an MCU receives a new batch of data, the device needs to program the correct flash-memory locations. If the locations are not already blank, the MCU must erase them before programming. Typically you want to verify the memory contents during or after programming.

4.2.5 Transferring Control to a Valid Application Program

After receiving and programming the new code, the bootloader writes a checksum or other unique byte sequence to a fixed memory location. The bootloader then checks for this value. If it is present, the bootloader transfers control to the application.

4.3 Bootloader Configuration and Operation

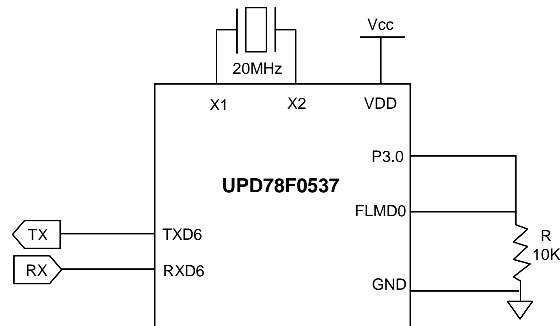
This section describes the bootloader's configuration and operation while loading new application code, using a demonstration example.

4.3.1 Demonstration Platform

The hardware platform used to demonstrate the bootloader includes:

- UPD78F0537 MCU
- 20-MHz clock
- FLMD0 pin tied to ground via a 10K-ohm resistor
- P3.0 port-output controls the state of FLMD0 pin
- UART6 serial interface

Figure 5. Demonstration Hardware



4.3.2 Bootloader Communication

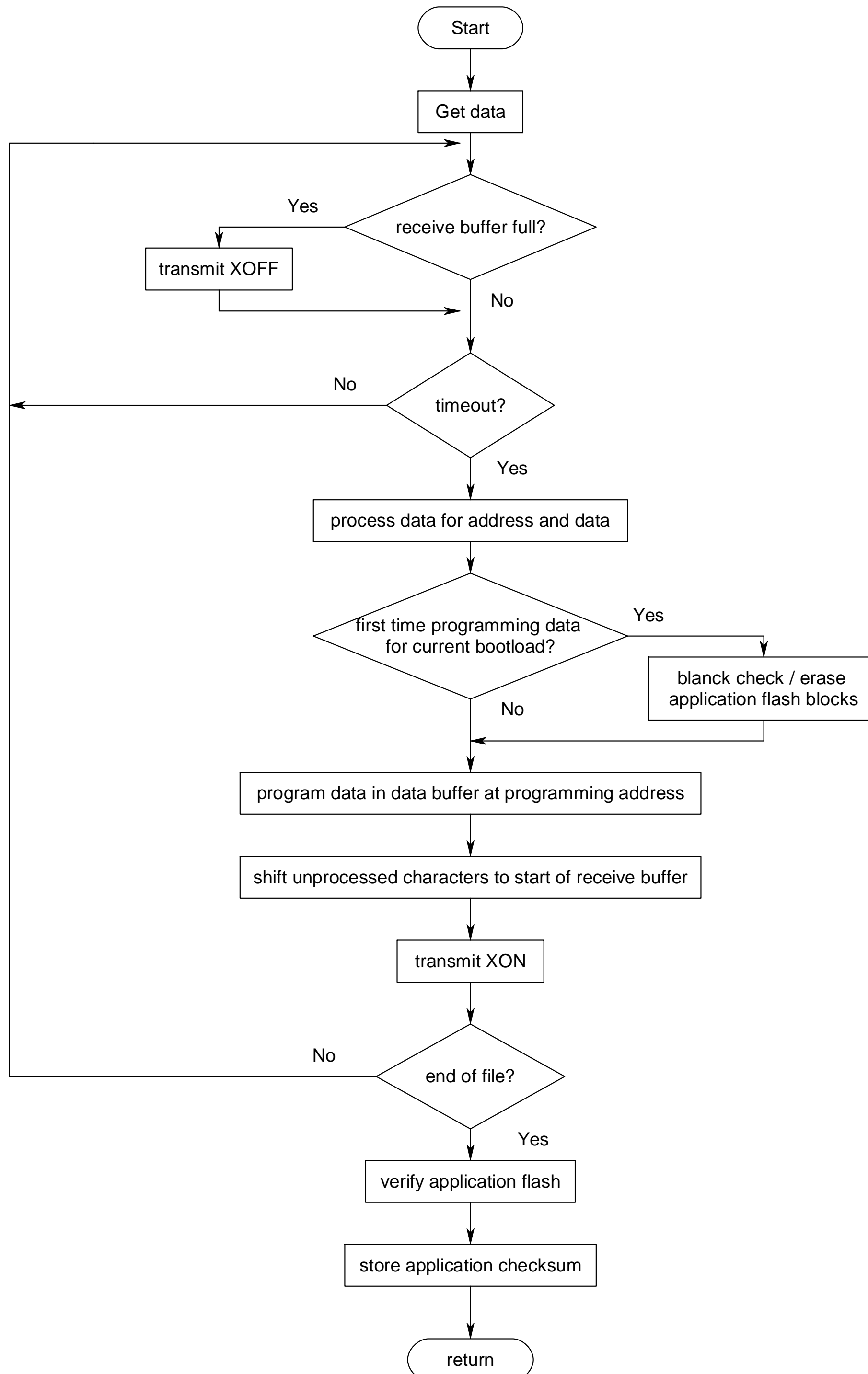
The UART6 used by the MCU for transmitting and receiving serial data is configured as follows:

- 115200 baud
- 8 data bits
- No parity
- One stop bit
- XON/XOFF flow control

4.3.3 Bootloader Operation

The figure below outlines the **Erase**, **Program**, and **Verify** commands of the application code using XON/XOFF flow control to download self-programming addresses and data.

Figure 6. Bootloader Flow



4.3.4 Boot prompt for Loading Application

When the MCU boots up, either on power-up or after a reset, the device displays the following prompt via an attached terminal:

```
NEC Electronics America Bootloader Ver. 1
Load Y/N?
```

If you enter **Y**, the device prompts you to download the new application file. The device expects a file in standard Intel hexadecimal 16 code, as detailed in Appendix C. If you enter **N,** or if there is no entry for a number of seconds, the MCU checks for valid application code to execute

4.3.5 Loading Application

When you enter **Y** at boot-up, the bootloader prompts you to send the file:

```
Send File
```

You then can use the **Send** or **Transfer File** command in your terminal program to send the file.

4.3.6 Receiving Data

The bootloader loads the incoming data into a receive buffer. When the buffer is full, the bootloader transmits the XOFF character to halt the flow of data and continues to process any remaining data until no more characters are received. A timeout causes an exit from the receive loop and begins data processing. Note that the same timeout mechanism comes into play if received characters stop before the receive buffer is full.

4.3.7 Processing the Received Data

The bootloader examines the data in the receive buffer and executes tasks as follows:

- Checks that the line checksums are correct
- Checks for breaks in line addresses (non-contiguous addresses)
- Sets the programming address
- Extracts the data to be programmed into the data buffer
- Programs the data
- Shifts the unprocessed data to the start of the receive buffer
- Checks for end of file
- Transmits XON before returning to receive loop

The bootloader processes the receive buffer and stores a start address for programming. The bootloader then stores data from contiguous addresses into the data buffer. After processing all the data from the receive buffer or after encountering a break in the address sequence, the bootloader programs the data in the data buffer. Afterward, the bootloader shifts the data in the receive buffer to move any unprocessed data to the start of the buffer. The bootloader then transmits the XON character and returns to the receive loop where it left off. This XON/XOFF receive/programming sequence continues until the bootloader detects the end of the file.

Caution: The example bootloader provided by NEC Electronics can handle only '00' (data) as the record type in the Intel Hex 16 format. Refer to Appendix C for details on Intel Hex 16 format. Any other value in the record type location causes the program to malfunction.

4.3.8 Blank Checking and Erasing

After processing the first data set from the incoming file, the bootloader executes a **Blank Check/Erase** command before programming the application code into flash memory. Up to this point, if an error in the process causes the bootloader to reset, the application flash still valid. The bootloader blank-checks the flash memory first and only erases blocks that are not blank. If blocks are erased, then the bootloader outputs the numbers of erased blocks in hexadecimal format:

```
Erasing 02
Erasing 03
Erasing 1D
```

4.3.9 Programming

The bootloader programs the data in the buffer by specifying the address and the number of words to be programmed. Each time the bootloader programs data, it outputs the starting addresses so that you can monitor the progress, as follows:

```
Programming at ..
101C
102C
112C
Etc.
```

4.3.10 Verifying

After receiving and programming the whole file without errors, the bootloader verifies the flash memory and outputs the message:

```
Verifying..
```

4.3.11 Storing Valid-Application Checksum

When application code is successfully verified, the bootloader stores a 4-byte checksum at the end of flash memory. This checksum is summed over the entire flash application area, from the start of FIRST_FLASH_BLOCK_C to the end of LAST_FLASH_BLOCK_B, excluding the four bytes used for checksum storage. Since the UPD78F0537 microcontroller has 128 KBytes of flash memory, the last block (47) of bank 5 stores the checksum:

```
#define FIRST_FLASH_BLOCK_C      4           // first application block
#define LAST_FLASH_BLOCK_B       47          // last application block
```

After programming the checksum in normal mode, the bootloader verifies the block:

```
Checksum 00DC8195 at..
BFFC
Verifying..
```

Note that in this example, the checksum is calculated by summing the data bytes from 1000H to 7FFFH (common area), 8000H to BFFFH from banks 0 to 4, and 8000H to BFFB from bank 5. The bootloader stores the checksum as a 4-byte word in the last four locations of block 47, bank 5 (BFFCH, BFFDH, BFFEH, BFFFH).

4.3.12 Executing Application

Immediately after successfully loading the new application code, the MCU executes the code. Whether the code is written to run immediately after a bootload, power-up or reset, the bootloader always checks for a valid application checksum before executing the application code. The bootloader calculates a checksum for the flash application memory, compares the calculated and stored checksums, then outputs the information in this form:

```
Stored checksum    = 00DC8195
Calculated checksum = 00DC8195
```

If the values are equal, the MCU executes the code. Otherwise, the program resets and prompts you to download a new application. This valid-application check guards against execution of invalid code, which could happen, for example, if you program an MCU with the bootloader but it has not yet received application code. Invalid execution can also occur if noise or power loss causes an MCU reset or if one of these conditions occurs in the middle of a download.

4.3.13 Flash Self-Programming Errors

If the bootloader encounters errors, it outputs this type of message:

```
SFP: Function = 04   Return = 05
```

The bootloader then enters an endless loop and allows the watchdog timeout to force a reset.

4.3.14 Incorrect Line-Checksum Error

While processing characters in the receive buffer, the bootloader examines each line for the correct checksum. If an incorrect checksum is found, the bootloader outputs the message:

```
Line checksum!
```

The bootloader then enters an endless loop and allows the watchdog timeout to force a reset.

4.3.15 Boot Area Overwrite Error

When bootloading a new application and before programming the received data, the bootloader examines each address to make sure that it falls outside the boot cluster 0 area (0000H–0FFFH). Programming an address inside this range would corrupt the bootloader code. If an address falls inside this range, the bootloader outputs the message:

```
Boot Area!
```

The bootloader then enters an endless loop and allows the watchdog timeout to force a reset.

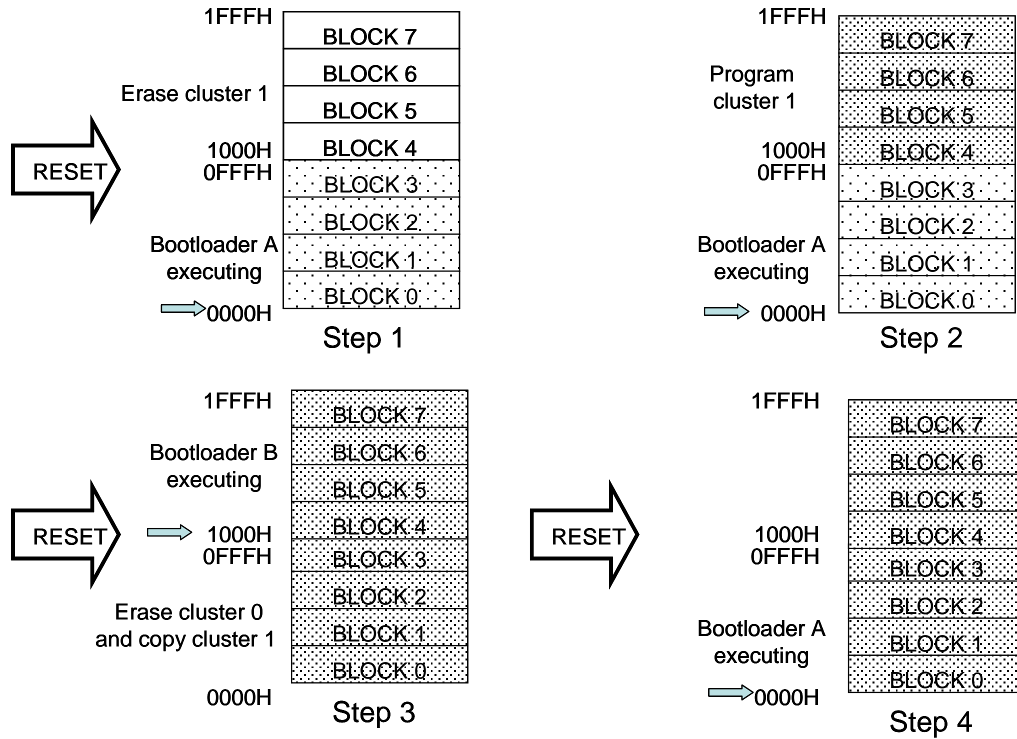
4.3.16 Boot Swapping

Replacing the bootloader in a microcontroller requires use of a procedure called boot swapping. This procedure involves two bootloaders—one already stored in the MCU’s boot cluster and the new bootloader that needs to be stored in the boot cluster. The following boot-swapping steps refer to these bootloaders as A and B, respectively:

- Bootloader A examines the first address of the file being sent. If the address is the reset address of 0000H, the bootloader knows that the file contains new bootloader code rather than an application. Bootloader A then erases boot cluster 1 (blocks 4 to 7).
- Although Bootloader B (the new bootloader) has addresses in the 0000H–0FFFH range (Boot Cluster 0), the Bootloader A must store the new code in the 1000H–1FFFH range (Boot Cluster 1). Bootloader A thus adds the offset 1000H to all addresses before programming.
- After programming area 1000H to 1FFFH, Bootloader A prompts you to replace the bootloader code. If you decide to replace Bootloader A, it sets the boot-swap flag and forces the MCU to reset by allowing the watchdog timer to overflow.
- On reset, the MCU sees that the boot-swap flag is set and starts execution from Boot Cluster 1 (Bootloader B).
- Bootloader B checks the boot-swap flag. If the flag is set, Bootloader B erases Boot Cluster 0 (Bootloader A) and copies itself into that area.
- After copying from Boot Cluster 1 to Boot Cluster 0, Bootloader B clears the boot-swap flag and allows the watchdog timer to overflow to force another reset.

- Upon this new reset, the MCU sees that the boot-swap flag is cleared and starts execution from Boot Cluster 0 (Bootloader B).
- Based on the boot-swap flag, the application starts at 0000H or 1000H.
- This example does not include any interrupt servicing.

Figure 7. Boot-Swapping Steps

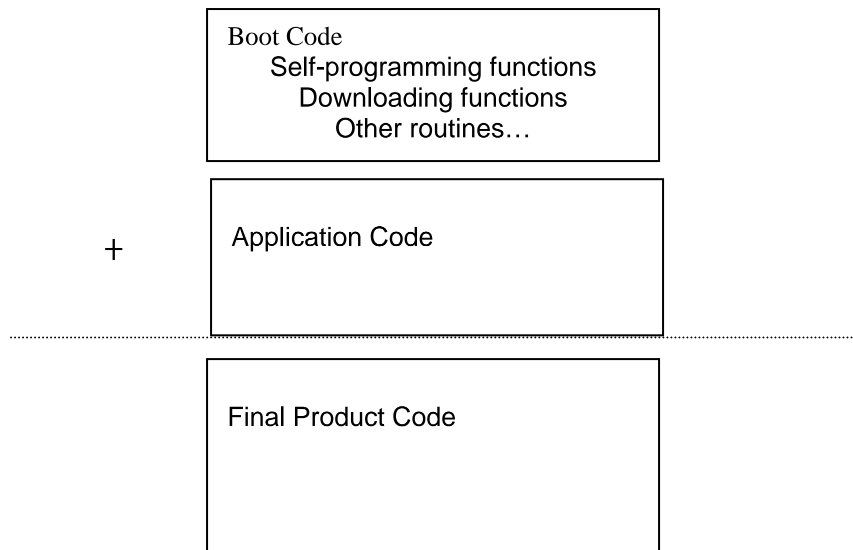


5. Developing Self-Programming Code Using NEC Electronics Tools

5.1 Code Structure

To take full advantage of flash self-programming, separate your code into two sections: the boot area and the application area. This separation allows you to update the application code without disrupting a bootloader program. In addition, you can update the bootloader itself with the MCU's boot-swapping capability.

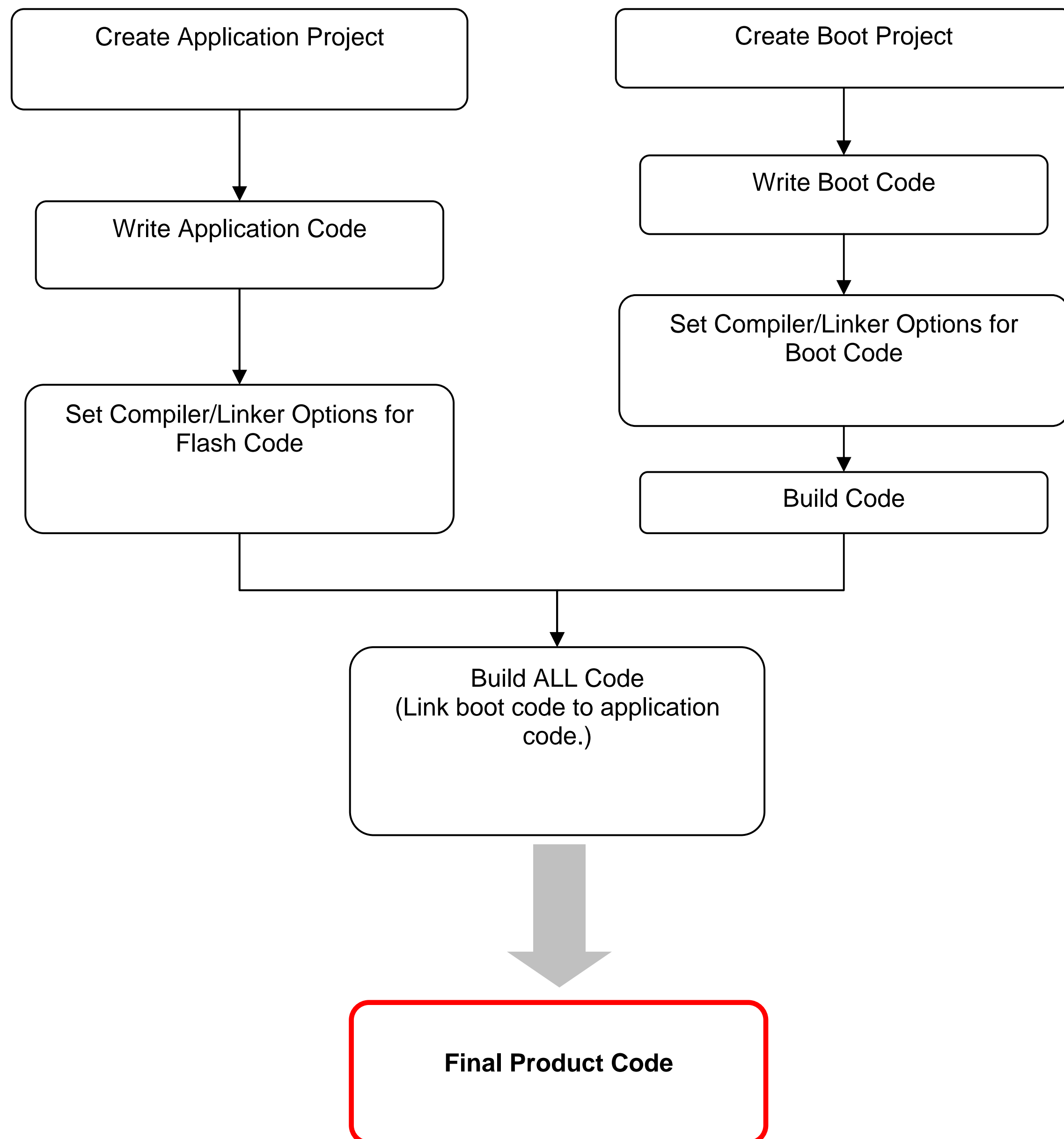
Figure 8. Self-Programming Code Structure



5.2 Development Flow

The typical development flow for generating code with self-programming capabilities using NEC Electronics tools is shown below.

Figure 9. Code Development Flow



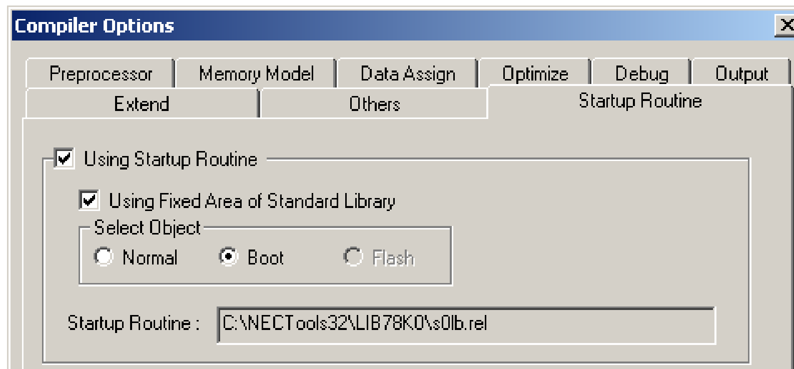
5.3 Tool Setup

NEC Electronics' tool suites provide everything necessary to develop code with self-programming functionality. You can easily create self-programming code with the compiler, linker and object converter options provided by NEC Electronics' PM Plus Integrated Development Environment (IDE) user interface. Below is a description of the minimum number of steps that you can follow to generate boot and application code with NEC Electronics tools' default options. For details on this procedure, refer to the CC78K0 and RA78K0 Language and Operation Manuals.

5.3.1 Procedure to Generate Boot Code

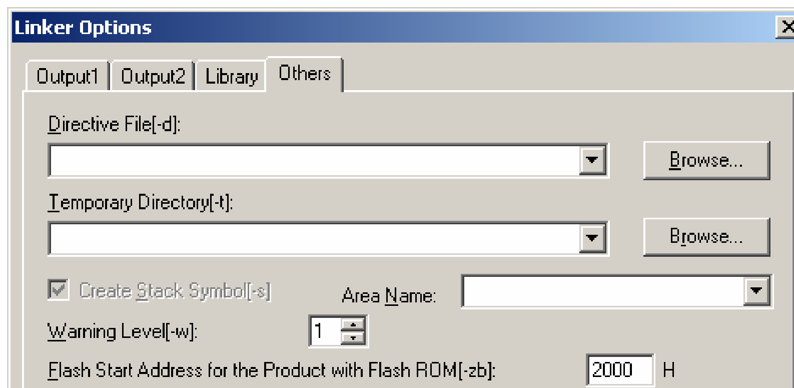
1. Name the Main() function **boot_main()**.
2. Specify **C Startup Routine** for **Boot** in C compiler options.

Figure 10. Selecting Compiler Options For Boot Code



3. Specify the starting address of application code (default value should be 2000H).

Figure 11. Selecting Linker Options for Boot Code



Notes:

- If the application code starts at an address other then 2000H, you must rebuild the NEC Electronics run-time libraries. Read the steps described in the *CC78K0 Language User's Manual* for more details.
- The bootloader example developed by NEC Electronics America uses 1000H as the application code start address.

5.3.2 Procedure to Generate Application (Flash) Code

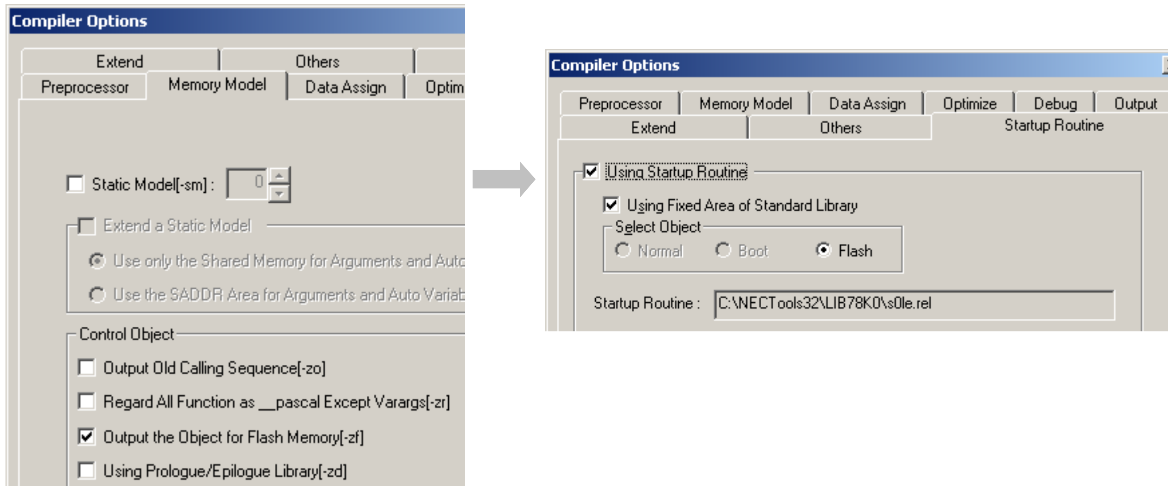
1. Specify (in code) the application code's starting address. At the very top of the C language source file, write the following:

```
#pragma ext_table 0x2000            //using address 2000H as example
```

Note that this directive defines the first address of the application code, which is used by the C startup routines and interrupt functions.

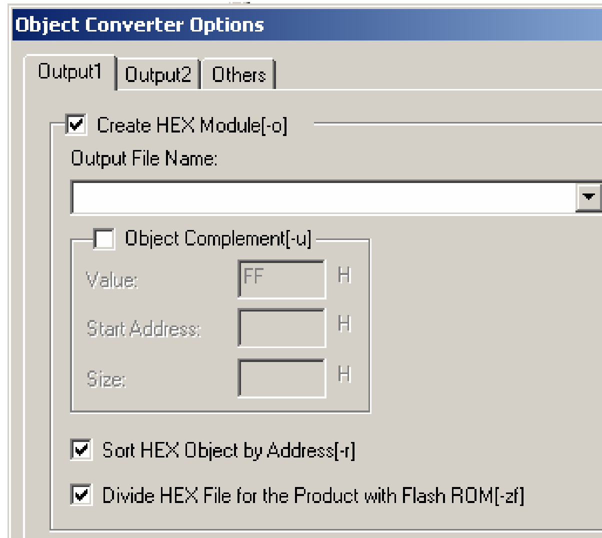
2. Select the setting in PM Plus to output object code for flash memory (application code). This setting automatically selects the correct C startup routine for flash code, as shown below.

Figure 12. Compiler Options for Application Code



3. In the **Object Converter** options, specify that the tool generate two separate HEX files, one for boot code and one for application (flash) code.

Figure 13. Select Object Converter Options for Application Code



The file extensions are:

- *.HXB (hex file for boot code)
- *.HXF (hex file for flash/application code)

Note that file *.HXB should match file *.HEX generated from boot code build.

6. Sample Code

The demonstration program consists of software modules made up of the files shown in the table below. The table indicates which files the Applilet generates and which you have to modify.

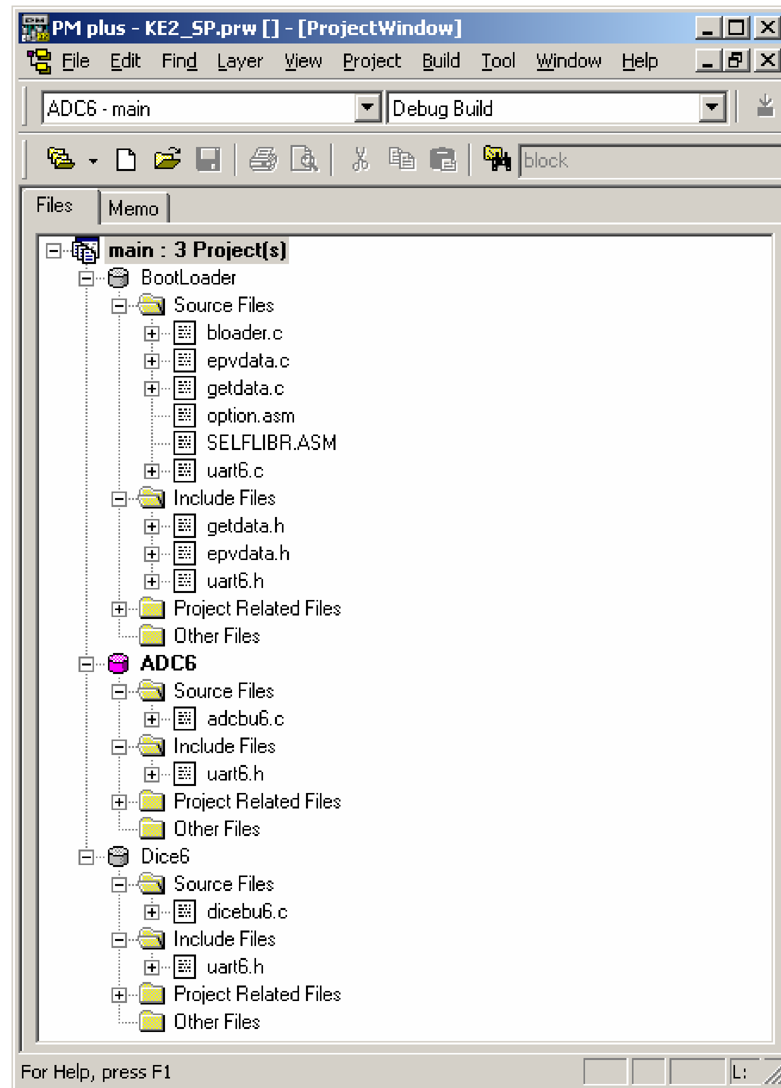
The listings for these files are located in the Appendix.

Table 2. Demonstration Program Software Modules

File	Purpose	Generated By Applilet	Modified By User
bloader.c	Boot loading related routine	No	Yes
evpdata.c	Erase verify and program routine	No	Yes
getdata.c	Get data from user input routine	No	Yes
uart6.c	RS232 port communication to host routine	No	Yes
adcbu6.c	Analog to digital conversion demo routine	No	Yes
dicebu6.c	Dice demo routine	No	Yes
uart6.h	RS232 port communication to host routine declarations	No	Yes
epvdata.h	Erase, verify and program routine declarations	No	Yes
getdata.h	Get data from user input routine declarations	No	Yes
SELFLIBR.ASM	Self programming library asm routine	No	Yes
option.asm	Option byte setting	No	Yes

Three sub-project files are associated with this application. To configure them, create three sub-directories under the main directory for dice6, ADC6 and bootloader. Copy adcbu6.c and uart6.h to the dice directory. Copy adcbu6.c and uart6.h to the ADC directory. Copy the rest of the files (including uart6.h) to the bootloader directory. In the main directory, create the main project and add new projects individually into the main project. After making the project, you get the project file settings shown below.

Figure 14. Project File Settings



7. Appendix A — Flash Self-Programming Routines

7.1 bloader.c

```
/******************************************************************************
*
* FILE                    : bloader.c
* DATE                    : February 2006
* DESCRIPTION             : boot main file for M-78F0537
* CPU TYPE                : 78K0/KE2 - uPD78F0537D
*
* Notes:          :
*
******************************************************************************/

#pragma sfr
#pragma NOP
#pragma EI
#pragma DI

// defines
#define VERSION 3
#define SSEG_L  0xc7

/*=======================================
; include files
;======================================*/

#include "getdata.h"
#include "epvdata.h"
#include "uart6.h"

unsigned long storedAppWord;      // stored application checksum
unsigned int wordData;                        // general purpose int
const unsigned char
led_array[16]={0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10,0x08,0x03,0x46,0x21,0x06,0x0e};

/******************************************************************************/
// void hdwinit(void) - set up initial hardware
//         note - this is called by startup code before boot_main() is called
void hdwinit ()
{
        /* note that WDTON in option byte (option.asm) is 1, so */
        /* watchdog timer is enabled on reset */

        DI();               // disable interrupts

        WDTE=0xac;                          // clear watchdog timer

        IXS     = 0x00;         /* enable 6144 bytes IXS RAM (uPD78F0537) */
        IMS     = 0xCC;         /* set memory size register for uPD78F0537 (128KB)*/
                                        /* 32K common area + 5 x 16K bank */
        BANK = 0x00;    /* set for bank area 0 */
```

```
    /* setup clock generator, switch to X1/X2 clock */
    /* set: EXCLK (OSCCTL.7) = 0, OSCSEL (OSCCTL.6) = 1 for X1/X2 crystal */
    /*      EXCLKS (OSCCTL.5) = 0, OSCSELS (OSCCTL.4) = 1 for XT1/XT2 crystal */
    /*      AMPH (OSCCTL.0) = 0 for X1/X2 clock > 10 MHz */
    OSCCTL = 0x50;

    /* set MSTOP (MOC.7) = 0 to start X1/X2 oscillator */
    MSTOP = 0;

    /* wait for X1 oscillator to stabilize */
    while (OSTC.0 == 0){
        WDTE=0xac;                      // clear watchdog timer
        ;
    }

    /* set for clock from X1 input instead of  Internal Oscillator */
    MCM = 0x05;         /* XSEL (MCM.2) = 1, MCM0 (MCM.0) = 1 */

    /* wait for clock to switch to X1 clock */
    while (MCS == 0){
        WDTE=0xac;                      // clear watchdog timer
        ;
    }


    LVIM = 0x00;        //prohibit LVI detection
    LVIS = 0x00;

    PCC = 0x00;         /* set for fastest CPU execution */

}


/**************************************************/
void boot_main(void)            // reserved word "boot_main"
{
    unsigned char delay;


    WDTE=0xac;                  // clear watchdog timer

    P4=0x03;                    // "b"
    P5=0x08;
    P7=led_array[VERSION];      // version of bootloader example

    PM4=0x00;
    PM5=0x00;
    PM7=0x00;   // set port to all outputs

    InitUART6_8N1(BAUD_115200_20);      // initialise UART6
    Enable_SFP();                       // enable self-flash programming

    bootSwapFlag=CheckBootSwapFlag();   // read boot swap flag
```

```
if(bootSwapFlag)                          // if boot swap flag is set
{
        Tx_String("\n\rBootSwapping..");  // transmit boot swap message
        performBootSwap();                // perform the boot swap
        ToggleBootSwapFlag();             // toggle flag to 0
        Disable_SFP();                    // disable self-flash programming
        while(1);                         // endless loop to allow watchdog timeout
}
// transmit program description
Tx_String("\n\r\n\rNEC Electronics America Bootloader Ver. ");
Tx_Character(VERSION+0x30);               // transmit version
Tx_Character(X_ON);                       // transmit XON
timeOut=MAX_COUNT;
while(timeOut>0)
{
        if(SRIF6)
        {
                rx_char=RXB6;
                SRIF6=0;
                timeOut=MAX_COUNT;
        }
        timeOut--;
}
Tx_String("\n\rLoad Y/N? ");              // query user if file load required?

if(ConfirmOnPrompt())                     // if load confirmed
{
        P7=SSEG_L;                        // display L on m-station LED2
        GetFile();                        // bootload the file
}
Disable_SFP();                            // disable self-flash programming

P7=led_array[VERSION];                    // redisplay version on m-station
P4=0x03;
P5=0x08;

WDTE=0xac;                                // clear watchdog timer

Tx_CRLF();

BANK=CS_FLASH_BANK;                       //setting Flash Bank to #5 for checksum processing
validApp_ptr=(unsigned long *)CS_ADDRESS; // point to checksum address
storedAppWord=*validApp_ptr;              // get stored checksum

BANK=0;                                   //setting Flash Bank to default

Tx_String("\n\rStored checksum     = ");   // transmit stored checksum
wordData=(unsigned int)(storedAppWord>>16);
Tx_Word(wordData);
wordData=(unsigned int)(storedAppWord&0x0000ffff);
Tx_Word(wordData);

validAppWord=CalculateAppCheckSum();      // calculate checksum
Tx_String("\n\rCalculated checksum = ");   // transmit calculated checksum
wordData=(unsigned int)(validAppWord>>16);
```

```
    Tx_Word(wordData);
    wordData=(unsigned int)(validAppWord&0x0000ffff);
    Tx_Word(wordData);


    if(validAppWord!=storedAppWord)                  // if checksums don't match
    {
        P4=0x06;                                             // "C"
        P5=0x0C;
        P7=led_array[0x0e];                          // "E"

        while(1);                                            // endless loop to allow
watchdog timeout
    }
    else
    {
        for(delay=0;delay<0x00ff;delay++) WDTE=0xac;   // add delay before Application start
    }
}
```

7.2 epvdata.c

```
/***************************************************************************
*
* FILE              : epvdata.c
* DATE              : February 2006
* DESCRIPTION       : bootload file for M-78F0537
* CPU TYPE          : 78K0/KE2 - uPD78F0537D
*
* Notes:        : supporting routines for erase program verify routines
*                   : plus checksum calculation and storage
*
***************************************************************************/
#define EPVDATA_C
#pragma sfr
#pragma NOP                     /* key word for NOP instruction */
#pragma DI
#include "epvdata.h"
#include "uart6.h"

//two structures below required by self-programming library files
struct stWordAddress{
        USHORT WriteAddress;
        UCHAR WriteBank;
};

struct stGetInfo{
        UCHAR OptionNumber;
        UCHAR GetInfoBank;
        UCHAR GetInfoBlock;
};

struct stWordAddress WordAddr;
struct stGetInfo GetInfo;

//NEC libraries functions in selflibr.rel (self-programming library)
extern  void    FlashStart(void);
extern  void    FlashEnd(void);
extern  void    FlashEnv(USHORT EntryRAM);
extern  UCHAR FlashBlockErase(UCHAR EraseBank,UCHAR EraseBlock);
extern  UCHAR FlashWordWrite(struct stWordAddress *ptr, UCHAR WordNumber, USHORT myDataBuffer);
extern  UCHAR FlashBlockVerify(UCHAR VerifyBank, UCHAR VerifyBlock);
extern  UCHAR FlashBlockBlankCheck(UCHAR BlankCheckBank,UCHAR BlankCheckBlock);
extern  UCHAR FlashGetInfo(struct stGetInfo *ptr, USHORT myDataBuffer);
extern  UCHAR FlashSetInfo(UCHAR SetInfoData);
extern  UCHAR CheckFLMD(void);


unsigned int gp_i;
unsigned char myEntryRam[ENTRY_RAM_SIZE];
unsigned char returnValue;

unsigned short startAddress;
```

```
unsigned char *bootswap_ptr;

void programDataBuffer(void)
{
	WDTE=0xac;			// clear watchdog timer

	if(firstProgram)		// if this is the first time programming
	{
		firstProgram=FALSE;	// reset first time programming flag

		if(programmingAddress==0)				// if first programming address is at 0000H
		{
				bootDownLoad=TRUE;					// boot download flag set
to TRUE
				EraseFlashBlocks(0,0,FIRST_BOOT_BLOCK,LAST_BOOT_BLOCK); // blank check
and erase blocks 4-7 in Common Area
		}
		else
		{
			EraseFlashBlocks(0,0,FIRST_FLASH_BLOCK_C,LAST_FLASH_BLOCK_C); // blank check
and erase blocks 4-31 in Common Area
			EraseFlashBlocks(0,5,FIRST_FLASH_BLOCK_B,LAST_FLASH_BLOCK_B); // blank check
and erase blocks 32-47 in Banks 0-5
		}

		Tx_String("\n\r\n\rProgramming at.."); // start of programming

	}
	startAddress=(unsigned short)(programmingAddress);		// get write start address

	numberOfWords=db_index/4;	// odd bytes can be ignored
	Tx_CRLF();				// new line for write addresses
	if(bootDownLoad)			// if its a boot download
	{
		WDTE=0xac;			// clear watchdog timer

		startAddress+=0x1000;	// add 1000H to start address
		Tx_Word(programmingAddress);	// transmit "from" address
		Tx_Character('-');
		Tx_Character('>');
	}
	else					// else if its not a boot download
	{
		if(programmingAddress<0x1000)	// and address is in boot area
		{
			Tx_String("\n\rBoot Area!");	// transmit error
			while(1);				// allow watchdog timeout
		}
	}
	Tx_Word((unsigned int)startAddress);			// transmit data start address

	WriteDataBufferToFlash(0,startAddress,numberOfWords); // write data in buffer
}

void Enable_SFP(void)
```

```
{
        FLMD0_CONTROL_IO=OUTPUT;                        // set FLMD0 control pin to output
        FLMD0_CONTROL_PIN=1;                            // set   FLMD0 high

        FlashStart();
        FlashEnv((unsigned short)&myEntryRam);

        // check mode is ok
        returnValue=CheckFLMD();

        if(returnValue!=0)                                              // if not successful
        {
                Tx_String("\n\rSFP Error: FLMD0 CHECK: ");      // transmit function return code
                Tx_Byte(returnValue);                   // return from function
        }

}

void Disable_SFP(void)
{
                FLMD0_CONTROL_PIN=0;                            // set FLMD0 pin low
                FlashEnd();
}

void EraseFlashBlocks(unsigned char firstBank, unsigned char lastBank, unsigned char firstBlock, unsigned char
lastBlock)
{
        unsigned char blocknumber, banknumber;
        WDTE=0xac;              // clear watchdog timer

        // blank check and erase blocks that may be programmed
        for(banknumber=firstBank;banknumber<lastBank+1;banknumber++)
        {
                for(blocknumber=firstBlock;blocknumber<lastBlock+1;blocknumber++)
                {
                        WDTE=0xac;
                        returnValue=FlashBlockBlankCheck(banknumber,blocknumber);

                        if(returnValue!=0)                              // if not successful
                        {
                                if(returnValue==NOT_BLANK)              // if its not blank
                                {
                                        Tx_String("\n\rErasing block ");        // transmit erasing

                                        returnValue=FlashBlockErase(banknumber,blocknumber); // erase block

                                        if(returnValue!=0)                      // if erase not successful
                                        {
                                                Tx_String("\n\rSFP Error: BLOCK ERASE: ");      // transmit
function return code
                                                Tx_Byte(returnValue);                   // return from
function
                                        }
                                        else                                            // else if erase successful
                                        {
```

```
                    if(!bootSwapFlag)                // if not bootswap
                    {
                        Tx_Byte(blocknumber);    // transmit block number
                    }
                    else                                        // transmit 0,1 instead of
2,3
                    {                                                   // as switch in
internal addressing
                        Tx_Byte(blocknumber-2); // transmit block number
                    }

                }
            }
            else                                                // else if blank check
error
            {
                Tx_String("\n\rSFP Error: BLOCK BLANK CHECK: ");        // transmit
function return code
                Tx_Byte(returnValue);                           // return from function

            }
        }
    }
  }
}

void WriteDataBufferToFlash(unsigned char banknumber,unsigned short writeAddress,unsigned char wordCount)
{
    WDTE=0xac;                  // clear watchdog timer

    WordAddr.WriteAddress = writeAddress;
    WordAddr.WriteBank = banknumber;

    returnValue=FlashWordWrite(&WordAddr,wordCount,&myDataBuffer);          // write data to flash memory

    if(returnValue!=0)                                                  // if not successful
    {
        Tx_String("\n\rSFP Error: WORD WRITE: ");           // transmit function return code
        Tx_Byte(returnValue);                               // return from function

    }
}

void VerifyFlashBlocks(unsigned char firstBank, unsigned char lastBank, unsigned char firstBlock, unsigned char
lastBlock)
{
    unsigned char blocknumber, banknumber;
    WDTE=0xac;                  // clear watchdog timer
    Tx_String("\n\rVerifying..\n\r");

        // for all specified blocks
        for(banknumber=firstBank;banknumber<=lastBank;banknumber++)
        {
            for(blocknumber=firstBlock;blocknumber<=lastBlock;blocknumber++)
            {
```

```
                    WDTE=0xac;          // clear watchdog timer

                    returnValue=FlashBlockVerify(banknumber,blocknumber);   // verify block

                    if(returnValue!=0)                                      // if not
successful
                    {
                              Tx_String("\n\rSFP Error: BLOCK VERIFY: ");   // transmit
function return code
                              Tx_Byte(returnValue);                         // return from
function
                    }
               }
          }
}

unsigned long CalculateAppCheckSum(void)
{
     unsigned long appCheckSum;
     unsigned char *address_ptr, bank;
     unsigned int firstaddr,lastaddr;

     WDTE=0xac;          // clear watchdog timer

     appCheckSum=0;                          // clear variable to hold application checksum

     //sum of whole memory
     firstaddr = 0x0000;
     lastaddr = 0xBFFF;

     for(bank=0;bank<6;bank++){

          BANK=bank;

          if (bank==1) firstaddr = 0x8000;
          else if (bank==5) lastaddr=CS_ADDRESS-1;

          for(address_ptr=(unsigned char *)firstaddr;address_ptr<(unsigned char *)lastaddr;address_ptr++){
               appCheckSum+=*address_ptr;          // sum data to application checksum
               WDTE=0xac;                               // clear watchdog timer
          }
     }
     BANK = 0;
     return appCheckSum;                     // return calculated checksum
}

void WriteVerifyAppCheckSum(void)
{
     WDTE=0xac;                                             // clear
watchdog timer

     validApp_ptr=(unsigned long *)myDataBuffer;            // point to start of data buffer
     *validApp_ptr=validAppWord;                                 // store checksum in buffer
     Tx_Word((unsigned int)CS_ADDRESS);                     // transmit data start address
     WriteDataBufferToFlash(CS_FLASH_BANK,CS_ADDRESS,1);    // program checksum
```

```
        VerifyFlashBlocks(CS_FLASH_BANK,CS_FLASH_BANK,CS_FLASH_BLOCK,CS_FLASH_BLOCK);
        // verify block
        WDTE=0xac;                                              // clear watchdog timer
}

unsigned char ConfirmOnPrompt(void)
{

        for(i=0;i<25;i++)
        {
                WDTE=0xac;              // clear watchdog timer

                timeOut=MAX_COUNT;
                while(timeOut>0)        // wait for user to confirm
                {
                        if(SRIF6)
                        {
                                rx_char=Rx_Character();  // get character
                                Tx_Character(rx_char);   // echo character
                                if((rx_char=='y')||(rx_char=='Y'))   // if yes
                                {
                                        return(TRUE);                   // return true
                                }
                                else
                                {
                                        if((rx_char=='n')||(rx_char=='N'))   // if no
                                        {
                                                return(FALSE);          // return FALSE
                                        }
                                        else            // if input but not yes or no
                                        {
                                                i=0;    // restart timeout loop
                                        }
                                }
                        }

                        timeOut--;                      // decrement timeout
                }
        }

        return(FALSE);  // if no input return false after timeout

}

unsigned char CheckBootSwapFlag(void)
{
        // check for boot swap flag

        GetInfo.OptionNumber=0x04;
        GetInfo.GetInfoBank=0;          //don't care when Option = 4
        GetInfo.GetInfoBlock=0;         //don't care when Option = 4

        returnValue=FlashGetInfo(&GetInfo,&myDataBuffer);

        if(returnValue!=0)                              // if not successful
```

```
    {
        Tx_String("\n\rSFP Error: GET INFORMATION: ");// transmit function return code
        Tx_Byte(returnValue);                       // return from function
    }
    return(myDataBuffer[0]);  // return information
}

void performBootSwap(void)
{
    unsigned char blocknumber,j;

    EraseFlashBlocks(0,0,FIRST_BOOT_BLOCK,LAST_BOOT_BLOCK);     // erase destination blocks
    WDTE=0xac;                                  // clear watchdog timer

    bootswap_ptr=(unsigned char *)0x0000;       // set pointer to boot code
    startAddress=(unsigned long)0x1000;         // set start address to destination
    for(int_i=0;int_i<256;int_i++)              // 256 * 16 times = 4096
    {
        for(j=0;j<16;j++)
        {
            myDataBuffer[j]=*bootswap_ptr++;    // fill data buffer with 16 bytes
        }
        WriteDataBufferToFlash(0,startAddress,4);  // write 4 words
        startAddress+=16;                           // increment address by 16
        WDTE=0xac;                                      // clear watchdog timer

    }
}

void ToggleBootSwapFlag(void)
{
    returnValue=FlashSetInfo(0xFE);   // write 0 to boot swap flag (bit 0)

    if(returnValue!=0)                              // if not successful
    {
        Tx_String("\n\rSFP Error: SET INFORMATION: "); // transmit function return code
        Tx_Byte(returnValue);                       // return from function
    }
}
```

7.3 getdata.c

```
/*****************************************************************************
*
* FILE                    : getdata.c
* DATE                    : February 2006
* DESCRIPTION             : bootload file for M-78F0537
* CPU TYPE                : 78K0/KE2 - uPD78F0537D
*
* Notes:          :
*
*****************************************************************************/
#define GETDATA_C
#pragma sfr
#pragma NOP
#pragma DI
#include "getdata.h"
#include "epvdata.h"
#include "uart6.h"


// flags
unsigned char firstXOFF;          // flag for first XOFF
unsigned char endOfFile;          // flag for end of file
unsigned char breakFlag;          // flag for break in address

// receive data
unsigned char rx_buffer[RB_SIZE];          // buffer to hold received data
unsigned int  rb_index;                    // index for receive buffer
unsigned char flowState;          // flow state

// record
unsigned char recordCount;                 // number of data bytes in line
unsigned char msAddress;          // most significant byte of line address
unsigned char lsAddress;          // least significant byte of line address
unsigned char recordType;         // record type (00=data, 01=end of file)
unsigned char checkSum;                    // checksum for line

// processing data
unsigned int  pr_index;                    // index used for processing
unsigned int  pr_end;                      // end index for processing loop
unsigned char charPosition;                // character position in line
unsigned char nextByte;                    // next byte in buffer
unsigned char sumOfBytes;                  // sum of bytes to verify checksum
unsigned char dataCount;          // count of remaining data bytes in line
unsigned char sob_whole;          // sum of bytes up to last whole sfp word
unsigned int rb_whole;                     // index at last whole sfp word
unsigned char dc_whole;                    // data count at last whole sfp word

// addresses
unsigned char oddAddress;                  // bytes greater than sfp word boundary
unsigned int currentAddress;      // current address of data
unsigned int lineAddress;         // received address of current line
```

```
// general use
unsigned int int_j;                          // integer j


void GetFile(void)
{
        WDTE=0xac;              // clear watchdog timer

        //initialise variables
        firstProgram=TRUE;              // first time programming is true
        firstXOFF=TRUE;                         // first XOFF is true
        endOfFile=FALSE;                // end of file is false
        breakFlag=FALSE;                // address break flag is false
        bootDownLoad=FALSE;             // boot download is false
        rb_index=0;                             // receive buffer index is 0
        charPosition=0;                 // line character position is 0
        Tx_Character(X_ON);             // ensure in XON state
        flowState=X_ON;                         // set flow state to XON
        Tx_String("\n\rSend file\n\r");         // ask user to send file

        while(!SRIF6)                   // wait for user to send file
        {
                WDTE=0xac;                      // clear watchdog timer
        }


        while(!endOfFile)               // main program loop
        {
                WDTE=0xac;                      // clear watchdog timer
                timeOut=MAX_COUNT;              // set timeOut to max


                while(timeOut!=0)       // get data loop
                {
                        WDTE=0xac;              // clear watchdog timer
                        timeOut--;              // decrement timeOut counter
                        if(SRIF6)               // if character received
                        {
                                rx_buffer[rb_index++]=RXB6;     // put character in buffer
                                                                        // and increment index
                                SRIF6=0;                                // clear interrupt flag
                                timeOut=MAX_COUNT;                      // set timeOut to max
                                if(rb_index>=RB_DATA_SIZE)      // if at data limit
                                {
                                        TXB6=X_OFF;                     // transmit XOFF character
                                        flowState=X_OFF;                // set flow state to XOFF
                                }
                        }
                }

                // flowstate XOFF (or XON if data has finished being transmitted)
                programmingAddress=currentAddress;      // assign new programming address
                db_index=0;             // reset data buffer index
                for(int_i=0;int_i<MDB_SIZE;int_i++)
```

```
		{
			myDataBuffer[int_i]=0x00;		// clear data buffer
		}
		pr_index=0;					// reset process receive buffer index

		if(breakFlag)				// if theres a break in addresses
		{
			breakFlag=FALSE;		// clear break flag
			oddAddress=(unsigned char)(programmingAddress%4);	// get odd address count
			while(oddAddress!=0)			// while odd address count not 0
			{
				programmingAddress--;		// decrement programming address
				myDataBuffer[db_index++]=0xff;	// put pad byte in data buffer
				oddAddress--;				// decrement odd address count

			}
		}

		// initialise processing loop
		if(rb_index>=RB_DATA_SIZE)	// if data at or greater than limit
		{
			pr_end=RB_DATA_SIZE-2;		// set end index to two less
		}
		else
		{
			pr_end=rb_index-1;		// -1 to avoid end of file character
		}

		while(pr_index<pr_end)	// loop for processing data in receive buffer
		{
			WDTE=0xac;			// clear watchdog timer

			switch(charPosition)// main switch decision tree
			{
				case 0:
						nextByte=rx_buffer[pr_index++];	// get next byte
						if(nextByte==':')		// if its a colon
						{
							charPosition=1;			// next char position is 1

						}
						else
						{
							if(firstXOFF)			// if file has no leading
colon
							{
								while(1);			// wait for
watchdog timeout
							}
						}
						break;

				case 1:
						recordCount=GetHexByte();	// get record count
						charPosition=3;				// next char position is 3
```

```
                        sumOfBytes=recordCount;              // start summing bytes in
line
                        dataCount=recordCount;        // initialise data count
                        break;

                case 3:
                        msAddress=GetHexByte();              // get ms byte of address
                        charPosition=5;                      // next char position is 5
                        sumOfBytes+=msAddress;               // add to sum of bytes
                        break;

                case 5:
                        lsAddress=GetHexByte();       // get ls byte of address
                        charPosition=7;                      // next char position is 7
                        sumOfBytes+=lsAddress;        // add to sum of bytes
                        lineAddress=(unsigned int)msAddress<<8;  // put ms byte in line
address
                        lineAddress|=(unsigned int)lsAddress;    // put ls byte in line
address
                        if(firstXOFF)                        // is this the first XOFF
                        {

                                firstXOFF=FALSE;             // no longer first XOFF
                                programmingAddress=lineAddress;// initialise programming
address
                                currentAddress=lineAddress;          // initialise
current address
                                oddAddress=(unsigned char)(programmingAddress%4);
        // get odd address count
                                while(oddAddress!=0)    // put programming address on sfp
word boundary
                                {
                                        programmingAddress--;        // decrement
programming address
                                        myDataBuffer[db_index++]=0xff;  // put pad byte
in data buffer
                                        oddAddress--;                        //
decrement odd address count
                                }

                        }
                        else                                         // if its not the
first XOFF
                        {

                                while((currentAddress<lineAddress)&&(oddBytes!=0))
                                {
                                        myDataBuffer[db_index++]=0xff;  // put in pad
byte
                                        currentAddress++;       // increment current
address
                                        oddBytes++;                          // increment odd
bytes
                                        oddBytes=oddBytes%4;// see if they make up a
whole word
```

```
                }
                if((oddBytes==0)&& (currentAddress<lineAddress))

                {
                        breakFlag=TRUE;            // set break flag to true
                        currentAddress=lineAddress;          // set new
current address to break address

        for(int_i=0,int_j=pr_index;int_j<RB_SIZE;int_i++,int_j++)
                        {
                                rx_buffer[int_i]=rx_buffer[int_j];    // shift
buffer data
                        }
                        rb_index-=pr_index;        // shift buffer index
                        programDataBuffer();
                        pr_index=pr_end;                    // exit loop if
break flag set

                }
        }

        break;

                case 7:
        recordType=GetHexByte();            // get record type
        if(recordType!=0)endOfFile=TRUE;           // if not 0 then assume
end of file
        sumOfBytes+=recordType;                    // add to sum of bytes
        charPosition=9;                            // next char position is 9
        break;

                case 9:
        if(dataCount!=0)              // if still data bytes remaining in line
        {
                nextByte=GetHexByte();  // get next data byte
                sumOfBytes+=nextByte;   // add to sum of bytes
                myDataBuffer[db_index++]=nextByte;        // put byte in
data buffer
                currentAddress++;                   // increment current
address
                dataCount--;                                // decrement
data count
                oddBytes=(unsigned char)(db_index%4);                 //
caluculate odd bytes
                if(oddBytes==0)                             // if whole
number of sfp words
                {
                        rb_whole=pr_index;                  // save index
                        sob_whole=sumOfBytes;  // save sum of bytes
                        dc_whole=dataCount;                 // save data
count
                }

        }
        else
```

```
                    {
                        charPosition=11;                // checksum at position
>=11
                    }
                    break;

            case 11:
                    checkSum=GetHexByte();              // get checksum

                    sumOfBytes+=checkSum;               // add to sum of bytes
                    if(sumOfBytes!=0)                       // checksum
error if sum not 0
                    {
                        Tx_String("\n\rLine checksum!");  // transmit error
                        while(1);
    // wait for watchdog reset
                    }
                    else                                    // else if
checksum correct
                    {
                        charPosition=0;                 // next char position is 13
                    }
                    break;


            default:
                    NOP();                                  // no action

                }   // end switch

        }   // end while(pr_index<pr_end)

        if(!breakFlag)                                  // if its not an address
break
        {
            if(rb_index>=RB_DATA_SIZE)              // if index >= data limit
            {
                currentAddress-=oddBytes;           // move current address back to
whole word boundary
                sumOfBytes=sob_whole;               // restore saved sum at whole
word boundary
                dataCount=dc_whole;                     // restore saved data
count
                for(int_i=0,int_j=rb_whole;int_j<RB_SIZE;int_i++,int_j++)
                {
                    rx_buffer[int_i]=rx_buffer[int_j];  // shift buffer data
                }
                rb_index-=rb_whole;                     // shift buffer index
                charPosition=9;                         // shifting back to data
                programDataBuffer();                // program data buffer

            }
            else                                            // else if index <
data limit
            {
```

```
                        while(oddBytes!=0)                          // while odd bytes not 0
                        {
                            myDataBuffer[db_index++]=0xff;  // put in pad byte
                            currentAddress++;                       // increment current
address
                            oddBytes++;                                 // increment odd
bytes
                            oddBytes=oddBytes%4;            // see if they make up a whole
word
                        }
                        programDataBuffer();                // program data buffer
                    }
                }
    Tx_Character(X_ON);             // turn transmission back on
    flowState=X_ON;                     // and set flow state to XON

    } // end  while(!endOfFile)


    timeOut=MAX_COUNT;                          // set timeout duration
    while(timeOut>0)                            // wait for any end of file characters
    {
        WDTE=0xac;                                  // clear watchdog timer

        timeOut--;                                  // decrement timeOut
        if(SRIF6)                                   // if serial character
        {
            rx_char=Rx_Character();  // rx_char = received char
            timeOut=MAX_COUNT;                      // reset timeOut to max
        }
    }
    WDTE=0xac;              // clear watchdog timer

    VerifyFlashBlocks(0,0,FIRST_FLASH_BLOCK_C,LAST_FLASH_BLOCK_C); // verify flash area

                //(only common area for demonstration purposes)
    if(!bootDownLoad)
    {
        WDTE=0xac;              // clear watchdog timer

        Disable_SFP();    // disable self-flash programming
        validAppWord=CalculateAppCheckSum();  // calculate application checksum
        Tx_String("\n\rChecksum ");
        Tx_Word((unsigned int)(validAppWord>>16));
        Tx_Word((unsigned int)(validAppWord&0x0000ffff));
        Tx_String(" at..\n\r");
        Enable_SFP();               // get self flash programming ready
        WriteVerifyAppCheckSum();       // write and verify the application checksum
        Disable_SFP();              // disable self-flash programming
    }
    else                                    // else if a boot download
    {
        WDTE=0xac;              // clear watchdog timer
        Tx_String("\n\rReplace the boot code Y/N? ");         // transmit confirm message
        if(ConfirmOnPrompt())
```

```
                {
                        ToggleBootSwapFlag();    // set the boot swap flag
                        bootSwapFlag=CheckBootSwapFlag(); // check status of boot swap flag
                }
                Disable_SFP();                  // disable self-flash programming

                P4=0x0F;                        // "r"
                P5=0x0A;
                P7=0x86;                        // "E"

                while(1);                       // wait for watchdog timeout
        }

}
/*****************************************************************************
*
* Function:       GetHexByte
* Parameters:     none
* Returns:        hex byte from char array
*                                         rx_buffer[pr_index],rx_buffer[pr_index+1]
* Date:           January 24, 2005
* Notes:                  increments entry value of pr_index by 2
*
*****************************************************************************/

unsigned char GetHexByte(void)
{
        unsigned char hexByte,nibbleValue;

        nibbleValue=rx_buffer[pr_index++]-'0';      // get ms nibble
        if(nibbleValue>9)nibbleValue-=7;            // convert to binary
        hexByte=nibbleValue<<4;                                 // put ms nibble in byte
        nibbleValue=rx_buffer[pr_index++]-'0';      // get ls nibble
        if(nibbleValue>9)nibbleValue-=7;            // convert to binary
        hexByte|=(nibbleValue&0x0f);                       // put ls nibble in byte
        return(hexByte);
}
```

7.4 uart6.c

```
/*****************************************************************************
*
* FILE                    : uart6.c
* DATE                    : November 17, 2004
* DESCRIPTION             : UART file for M-78F0537
* CPU TYPE                : 78K0/KE2 - 78F0537D
*
* Notes:          : UART6 (8 data bits, no parity, 1 stop bit)
*
*****************************************************************************/
#define UART6_C
#pragma sfr
#pragma NOP                          /* key word for NOP instruction */
#include "uart6.h"

const char Hex_Values[]="0123456789ABCDEF";
/*****************************************************************************
*
* Function:       InitUART6_8N1
* Parameters:     unsigned char value to set baud rate
* Returns:        nothing
* Date:           February 2006
* Description:    Initialises UART6 at passed baud rate
*                                     ( 8 data bits, No parity, 1 stop bit)
* Notes:
*
*
*****************************************************************************/

void InitUART6_8N1(unsigned char baudRate)
{
        unsigned int i;

        PM1.4=1;                /* set RxD6 pin to input */
        P1.3=1;                 // set TxD6 pin high
        PM1.3=0;                /* set TxD6 pin to output */

        ASIM6=0x01;             // stop uart - set to reset state

        BRGC6=BAUD_115200_20;          /*set baud rate */

        CKSR6 = 0x00;
        ASIM6 |= 0xE5;

        POWER6=1;               // enable UART6
        for(i=0;i<1000;i++)     //wait for 2 BRGC6 clocks
        {
                NOP();
        }

        STIF6=0;                // clear Tx interrupt request
        TXE6=1;                         // enable transmission
```

```
        SRIF6=0;                // clear Rx interrupt request
        RXE6=1;                 // enable reception
}
/*****************************************************************************
*
* Function:       Tx_String
* Parameters:     pointer to string array of unsigned char
* Returns:        nothing
* Date:           February 26, 2004
* Description:    Transmits string via UART
* Calls:                  Tx_Character(unsigned char ascii_character)
* Notes:
*
*****************************************************************************/
void Tx_String(const char *puc)
{
        while(*puc != NULL_CHAR)
                Tx_Character(*puc++);
}
/*****************************************************************************
*
* Function:                       Tx_Word
* Parameters:             16-bit number unsigned int
* Returns:                nothing
* Date:           February 29, 2004
* Description:    Converts 16-bit value to 4 ASCII HEX characters
*                                         and transmits them via UART
* Notes:
*
*****************************************************************************/
void Tx_Word(unsigned int data_16_bit)
{
        Tx_Byte((unsigned char)(data_16_bit >> 8));             // transmit ms byte
        Tx_Byte((unsigned char)(data_16_bit & 0xff));   // transmit ls byte
}
/*****************************************************************************
*
* Function:                       Tx_Byte
* Parameters:             8-bit number
* Returns:                nothing
* Date:           February 27, 2004
* Description:    Transmits 8-bit value as 2 ASCII HEX characters
*
* Notes:
*
*
*****************************************************************************/
void Tx_Byte(unsigned char data_8_bit)
{
        Tx_Character(Hex_Values[data_8_bit>>4]);// transmit hex character for ms nibble
        Tx_Character(Hex_Values[data_8_bit&0x0f]);      // transmit hex character for ls nibble
}

/*****************************************************************************
```

```
*
* Function:           Tx_CRLF
* Parameters:     none
* Returns:            nothing
* Date:               February 28, 2004
* Description:    Send CR/LF
*
* Notes:
*
*****************************************************************************/
void Tx_CRLF(void)
{
        Tx_Character(CR_CHAR);          // transmit carriage return char
        Tx_Character(LF_CHAR);          // transmit line feed char
}

/*****************************************************************************
*
* Function:       Tx_Character
* Parameters:     unsigned ASCII character
* Returns:        nothing
* Date:           February 23, 2004
* Description:    Transmits character via UART
*
* Notes:
*
*
*****************************************************************************/
void Tx_Character(char ascii_character)
{
        char clear_char;

        if((SRIF6) && (ASIS6!=0))                               // if any error in status register
        {
                SRIF6=0;                        // clear error interrupt request flag
                clear_char=RXB6;        // read receive register to clear error
        }

        TXB6=ascii_character;   // load character to transmit register
        while(STIF6==0);                // wait while transmission in progress
        STIF6=0;                                // clear interrupt request


}

/*****************************************************************************
*
* Function:       Rx_Character
* Parameters:     none
* Returns:        received character
* Date:           February 23, 2004
* Description:    Receives character via UART
*
* Notes:              INTSER0 set if receive error
*                                     Reading RXB6 clears ASIS6 error flags
```

```
*                                   ready for next reception
*
*
***************************************************************************/
char Rx_Character(void)
{
	char rx_char;


	// if reception error return NULL character
	// note that receive register must be read to clear error condition
	if(SRIF6)
	{
		if(ASIS6!=0)				// if any error in status register
		{
			SRIF6=0;			// clear error interrupt request flag
			rx_char=RXB6;		// read receive register to clear error
		}
		else
		{
			rx_char=RXB6;		// get received char
			SRIF6=0;			// clear error interrupt request flag

		}
	}

	return(rx_char);			// return received character

}
```

7.5 adcbu6.c

```
/*****************************************************************************
*
* FILE                  : adcbu6.c
* DATE                  : February 2006
* DESCRIPTION           : adc demo file for M-Station M-78F0537
* CPU TYPE              : 78K0/KE2 - 78F0537
*
* Notes:        : For use with bootloader
*                               : assumes boot code has already initialized micro & UART
*
*****************************************************************************/


#pragma ext_table 0x2000
#pragma SFR
#pragma NOP
#pragma DI
#pragma EI
#pragma interrupt INTSR6 UartRx_Interrupt rb1
#include "uart6.h"

#define SWITCH_2 P3.1
#define SWITCH_3 P3.2

#define HEX_DISPLAY 0
#define VOLT_DISPLAY 1

void Delay(unsigned int delay);
const unsigned char
led_values[16]={0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10,0x08,0x03,0x46,0x21,0x06,0x0e};
unsigned char int_count;
unsigned char led1_value;
unsigned char led2_value;
unsigned char led_index;
unsigned char dp_LED1,dp_LED2;
unsigned int adc_result;
unsigned char adc_msbyte;
unsigned char volts;
unsigned int loop_count;
unsigned char display_state;
char received_char;
unsigned int timedowncount;
unsigned char tcnt;
unsigned char bootRequest;
unsigned char intRx;

unsigned char old_volts=0, old_adcval=0;

void main(){

        unsigned int i;
        DI();                   // disable interrupts
```

```
WDTE=0xac;                // clear watchdog timer

bootRequest=0;   // request for bootloader is false
intRx=0;         // receive interrupt flag is 0
ADCS=1;                          // enable ADC

P4=0xff;         // set seven segment led1 outputs off
P5=0xff;
P7=0xff;         // set seven segment led2 outputs off

PM4=0x00;                 // set seven segment led1 to all outputs
PM5=0x00;
PM7=0x00;                 // set seven segment led2 to all outputs
PM3|=0x06;                // set switch 2 and 3 lines to inputs

dp_LED2=0x80; // set value to keep decimal point 2 turned off
led1_value=0;    // reset led1 value
led2_value=0;    // reset led2 value
display_state=VOLT_DISPLAY;   // set display mode to volts

//Transmit Header message for Demo
Tx_String("\n\r\n\rNEC Electronics America, Inc. 2006\n\r");
Tx_String("ADC Display Demo on M-Station\n\r\n\r");
Tx_String("Press SW2 to change display to hex value (most significant 8-bits of ADC result)\n\r");
Tx_String("Press SW3 to change display to volts\n\r");
Tx_String("Potentiometer varies voltage from 0.0 to 5.0V \n\r\n\r");
Tx_String("(Hit any key to get bootloader prompt)\n\r\n\r");


SRIF6=0;         // clear Rx interrupt request
SRMK6=0;         // enable Rx interrupts

EI();            // enable interrupts

while(1){        // main loop
        WDTE=0xac;              // clear watchdog timer
        loop_count++;    // increment loop counter

        if(ADIF)                // if ADC ready
        {
                adc_result=ADCR;        // get result
                adc_msbyte=(unsigned char)(ADCR>>8);    // get ms byte of result
                ADIF=0;                 // clear ADC interrupt flag
        }

        if(display_state==VOLT_DISPLAY)                 // if display mode is volts
        {
                volts=adc_msbyte/5;                             // approximate volt value
                led1_value=volts/10;                    // get led1 volt units

                P4=(led_values[led1_value]) & 0x0F;
                P5=(led_values[led1_value]>>4) & 0x0F;
                P5.3=0;                                                 // display decimal point

                led2_value=volts%10;                    // get led1 volt tenths
```

```
                P7=(led_values[led2_value])|dp_LED2;       // display led2 value

                if (volts!=old_volts){
                        Tx_Character(CR_CHAR);
                        Tx_Character(0x30+led1_value);
                        Tx_Character('.');
                        Tx_Character(0x30+led2_value);
                        Tx_Character('V');
                        old_volts=volts;
                }
        }

        else                                                        // else if display mode is hex
        {
                led1_value= adc_msbyte>>4;                 // get led1 ms nibble
                P4=(led_values[led1_value]) & 0x0F;
                P5=(led_values[led1_value]>>4) & 0x0F;
                P5.3=1;                                             // turn off decimal point

                led2_value= adc_msbyte&0x0f;      // get led2 ls nibble
                P7=(led_values[led2_value])|dp_LED2;       // display led2 value

                if(adc_msbyte!=old_adcval){
                        Tx_Character(CR_CHAR);
                        Tx_Byte(adc_msbyte);
                        Tx_Character('H');
                        Tx_Character(' ');
                        old_adcval=adc_msbyte;
                }
        }

        if((SWITCH_2==0)&& (SWITCH_3==1))                // if only sw2 pressed
        {
                Delay(1000);

                if((SWITCH_2==0)&& (SWITCH_3==1))
                {
                        display_state=VOLT_DISPLAY;                // display mode is volts
                }

        }

        if((SWITCH_2==1)&& (SWITCH_3==0))                // if only sw3 pressed
        {
                Delay(1000);

                if((SWITCH_2==1)&& (SWITCH_3==0))
                {
                        display_state=HEX_DISPLAY;                 // display mode is hex
                }

        }

        if(intRx)                                                   // if character received
        {
```

```
DI();                                                            // disable interrupts

Tx_String("\n\rStart bootloader Y/N?\n\r");  // ask for confirmation

for(tcnt=0;tcnt<40;tcnt++)
{
        WDTE=0xac;                  // clear watchdog timer

        timedowncount=0xffff;       // set timeout count

        while(timedowncount>0)  // while not timed out
        {
                if(SRIF6)                              // if received character
                {
                        received_char=Rx_Character();       // get character
                        Tx_Character(received_char);        // echo character

                        if(received_char=='y'  || received_char=='Y')
                        {
                                bootRequest=1;   // if yes set boot request flag
                                tcnt=40;             // and set tcnt for exit
                        }
                        else                                       // if not yes
                        {
                                if(received_char=='n'  || received_char=='N')
                                {
                                        bootRequest=0;   // if no clear boot request flag
                                        tcnt=40;             // and set tcnt for exit
                                }
                                else
                                {
                                        bootRequest=0;   // clear boot request flag
                                }
                        }

                }
                timedowncount--;            // decrement timout count
        }
}

if(bootRequest)                 // if its a boot request
{                                                       // set led1 and led2 displays to rE
        P4=0x0f;                        // "r"
        P5-0x0a;
        P7=0x06;                        // "E"

        while(1);                       // loop to allow watchdog timer to timeout

}
else        // if not a boot request transmit returning to application message
{
        Tx_String("\n\rReturned to ADC Display Demo on M-Station\n\r");
        Tx_String("(Hit any key to get bootloader prompt)\n\r\n\r");
}
```

```
                    intRx=0;                                        // clear interrupt flag
                    EI();                                           // enable interrupts
                }

                WDTE=0xac;              // clear watchdog timer

                Delay(5000);        // delay
        }
}

void Delay(unsigned int delay)
{
        unsigned int i;

        for(i=0;i<delay;i++);
}

/*****************************************************************************
*
* Function:        UartRx_Interrupt
* Parameters:      none
* Returns:         nothing
* Date:            March 12, 2004
* Description:     Interrupt service routine for UART6 reception
*
* Notes:                        Used for bootloader request
*
*****************************************************************************/

__interrupt void UartRx_Interrupt(void)
{

        received_char=RXB6;     // get interrupting character
        intRx=1;                // set interrupt flag
}
```

7.6 dicebu6.c

```
/***************************************************************************
*
* FILE                  : dicebu6.c
* DATE                  : February 2006
* DESCRIPTION           : dice demo file for M-Station M-78F0537
* CPU TYPE              : 78K0/KE2 (uPD78F0537D)
*
* Notes:        : For use with bootloader
*                               : assumes boot code has already initialized micro & UART
*                               : Uses interrupts and includes library files
*
***************************************************************************/

// pragmas
#pragma ext_table 0x2000
#pragma SFR
#pragma NOP
#pragma DI
#pragma EI
#pragma interrupt INTSR6 UartRx_Interrupt rb1
#pragma interrupt INTTM50 TM50_Interrupt rb2

// included files
#include "uart6.h"
#include <stdlib.h>

// defines
#define SWITCH_2 P3.1
#define SWITCH_3 P3.2

// function declarations
void InitTimer50(void);
void StartTimer50(void);

// values for m-station seven segment leds (led1 and led2)
const unsigned char led_values[10]={0x40,0x79,0x24,0x30,0x19,0x12,0x02,0x78,0x00,0x10};

// variables
unsigned char int_count;
unsigned char led1_count;
unsigned char led2_count;
unsigned char led_index;
unsigned char dp_LED2;
unsigned int adc_result;
unsigned char adc_msbyte;
unsigned char ledcycle_speed;
unsigned int gp_int2;
unsigned int gp_int3;
unsigned int rand2;
unsigned int rand3;
unsigned int loop_count;
unsigned char doublesix;
```

```
unsigned char speedreported;
unsigned char highrollspeed;
char received_char;
unsigned char tcnt,bootRequest;
unsigned int timedowncount;
unsigned char intRx;

void main(){

    unsigned int i;
    DI();               // disable interrupts


    intRx=0;                    // clear receive interrupt flag
    received_char='Q';          // set receive char to Q
    bootRequest=0;              // boot load request is false
    ADCS=1;                                     // enable ADC

    PM3.1 = 1;                          // input for switch
    PM3.2 = 1;                          // input for switch

    P4=0xff;
    P5=0xff;
    P7=0x00;                            // set 7 segment led2 outputs off

    PM4=0x00;                           // set 7 segment led1 pins to outputs
    PM5=0x00;
    PM7=0x00;                           // set 7 segment led2 pins to outputs

    dp_LED2=0x80;               // set led2 decimal point off value
    led1_count=6;               // initialise led1 count to 6
    led2_count=6;               // initialise led2 count to 6
    doublesix=1;                // set double six flag to true
    speedreported=0; // set speed reported value to 0
    highrollspeed=0;  // set high roll speed to 0

    InitTimer50();              // initialise timer 50
    StartTimer50();             // start timer 50

    WDTE=0xac;                          // clear watchdog timer
    SRIF6=0;                            // clear Rx interrupt request
    ADMK=1;                                     // ADC interrupt is masked

    //Transmit Header message for Demo
    Tx_String("\n\r\n\rNEC Electronics America, Inc. 2006\n\r");
    Tx_String("Dice Rolling Demo on M-Station\n\r");
    Tx_String("SW2 rolls die displayed on seven segment display LED1\n\r");
    Tx_String("SW3 rolls die displayed on seven segment display LED2\n\r");
    Tx_String("Potentiometer controls the speed of the roll\n\r\n\r");
    Tx_String("What is the fastest roll you can throw that gets a double six?\n\r\n\r");
    Tx_String("(Hit any key to get bootloader prompt)\n\r\n\r");


    adc_result=ADCR;                                        // get adc result
    adc_msbyte=(unsigned char)(ADCR>>8);    // get ms byte of result
```

```
	ledcycle_speed=0xff-adc_msbyte;			// calculate cycle speed

	if(ledcycle_speed>0x10)CR50=ledcycle_speed;	// if speed > 10H put speed in timer compare register
	ADIF=0;							// clear adc interrupt flag
	SRIF6=0;		// clear Rx interrupt request
	SRMK6=0;		// enable Rx interrupts

	srand(adc_result);		// seed random number with adc result

	EI();				// enable interrupts

	while(1){
		WDTE=0xac;			// clear watchdog timer
		loop_count++;	// increment loop count
		{rand2=rand(); gp_int2=rand2;}		// get random number 2
		{rand3=rand(); gp_int3=rand3;}		// get random number 3

		NOP();				// no operation

		if(doublesix && (!speedreported))  // if its a double six
		{								// and speed has not been
reported
			DI();						// disable interrupts
			if(adc_msbyte>highrollspeed)	// if current speed > high speed
			{
				highrollspeed=adc_msbyte;		// set high speed to current speed
			}

			Tx_String("Current Roll speed for double six = ");	// transmit roll speed
			Tx_Byte(adc_msbyte);			// value of ms byte of adc result
			Tx_CRLF();					// go to new line
			Tx_String("Highest Roll speed for double six = ");	// transmit roll speed
			Tx_Byte(highrollspeed);			// value of highest roll speed
			Tx_CRLF();					// go to new line
			Tx_CRLF();					// go to new line

			loop_count=0;				// clear loop count
			speedreported=1;			// set speed reported flag to true
			EI();						// enable interrupts
		}

		if(intRx)						// if character received
		{
			DI();							// disable interrupts

			Tx_String("\n\rStart bootloader Y/N?\n\r");  // ask for confirmation

			for(tcnt=0;tcnt<40;tcnt++)
			{
				WDTE=0xac;			// clear watchdog timer

				timedowncount=0xffff;	// set timeout count

				while(timedowncount>0)	// while not timed out
				{
```

```
                    if(SRIF6)                           // if received character
                    {
                        received_char=Rx_Character();   // get character
                        Tx_Character(received_char);    // echo character

                        if(received_char=='y'  || received_char=='Y')
                        {
                            bootRequest=1;   // if yes set boot request flag
                            tcnt=40;         // and set tcnt for exit
                        }
                        else                            // if not yes
                        {
                            if(received_char=='n'  || received_char=='N')
                            {
                                bootRequest=0;   // if no clear boot request flag
                                tcnt=40;         // and set tcnt for exit
                            }
                            else
                            {
                                bootRequest=0;   // clear boot request flag
                            }
                        }
                    }

                    timedowncount--;        // decrement timout count
                }
            }

            if(bootRequest)         // if its a boot request
            {                                       // set led1 and led2 displays to rE
                P4=0x0f;                // "r"
                P5=0x0a;
                P7=0x06;                // "E"

                while(1);               // loop to allow watchdog timer to timeout
            }

            else    // if not a boot request transmit returning to application message
            {
                Tx_String("\n\rReturned to Dice Rolling Demo on M-Station\n\r");
                Tx_String("(Hit any key to get bootloader prompt)\n\r\n\r");
            }

            intRx=0;                                            // clear interrupt flag
            EI();                                                       // enable interrupts
        }

        if(bootRequest)         // if its a boot request
        {                                       // set led1 and led2 displays to rE
            P4=0x0f;                // "r"
            P5=0x0a;
            P7=0x06;                // "E"

            while(1);               // loop to allow watchdog timer to timeout
        }
```

```
                WDTE=0xac;              // clear watchdog timer
        }
}

/*****************************************************************************
*
* Function:       UartRx_Interrupt
* Parameters:     none
* Returns:        nothing
* Date:           January 21, 2005
* Description:    Interrupt service routine for UART6 reception
*
* Notes:                  Used for bootloader request
*
*****************************************************************************/

__interrupt void UartRx_Interrupt(void)
{

        received_char=RXB6;     // get interrupting character
        intRx=1;                // set interrupt flag
}

/*****************************************************************************
*
* Function:       TM50_Interrupt
* Parameters:     none
* Returns:        nothing
* Date:           January 21, 2005
* Description:    Interrupt service routine for timer 50
*
* Notes:                  Used for switch debouncing
*
*****************************************************************************/

__interrupt void TM50_Interrupt(void)
{
        int_count++;            // increment interrupt count

        // if not double six set double six and speed reported flags to 0
        if((led1_count!=6) || (led2_count!=6)){doublesix=0;speedreported=0;}

        // if double six and neither switch 2 or switch 3 are pressed
        if((led1_count==6)&&(led2_count==6)&&(SWITCH_2==1)&&(SWITCH_3==1))
        {
                doublesix=1;                    // set double six flag
                if((int_count&0x03)==0)         // for every 4th interrupt
                {
                        P4=0xff;                // turn off seven seg led1
                        P5=0xff;
                        P7=0xff;                // turn off seven seg led2
                }
                else                                    // else for all other interrupts
                {
```

```
                P4=(led_values[6]) & 0x0f;                      // display 6 on seven seg led1
                P5=(led_values[6]>>4) & 0x0f;
                P5.3 = 1;

                P7=(led_values[6])|dp_LED2;     // display 6 on seven seg led2


            }
        }
        if((int_count&0x07)==0)                 // for every 8th interrupt
        {
            if(SWITCH_2==0)                             // if switch 2 is pressed
            {

                led1_count=(unsigned char)gp_int2;          // get led1 count
                led1_count&=0x07;                               // use 3 ls bits
                if(led1_count>6)led1_count=1;       // change 7 result to 1
                if(led1_count==0)led1_count=6;      // change 0 result to 6
                P4=(led_values[led1_count]) & 0x0f;             // display led1 count
                P5=(led_values[led1_count]>>4) & 0x0f;          // display led1 count
                P5.3 = 1;

            }
            if(SWITCH_3==0)                             // if switch 3 is pressed
            {
                led2_count=(unsigned char)gp_int3;          // get led2 count
                led2_count&=0x07;                               // use 3 ls bits
                if(led2_count>6)led2_count=1;       // change 7 result to 1
                if(led2_count==0)led2_count=6;      // change 0 result to 6
                P7=(led_values[led2_count])|dp_LED2;        // display led2 count

            }


            if(ADIF)                                    // if ADC is ready
            {
                adc_result=ADCR;                    // get ADC result
                adc_msbyte=(unsigned char)(ADCR>>8);    // get ms byte of result
                ledcycle_speed=0xff-adc_msbyte;             // calculate cycle speed
                if(ledcycle_speed>0x10)CR50=ledcycle_speed;     // if speed > 10H put speed in timer
compare register
                ADIF=0;                                         // clear ADC interrupt flag

            }
        }

}
/*****************************************************************************
*
* Function:       InitTimer50
* Parameters:     none
* Returns:        nothing
* Date:           January 21, 2005
* Description:    Initialises timer/event counter 50
*
```

```
* Notes:
*
*****************************************************************************/

void InitTimer50(void)
{
        TMC50=0x00;             // stop timer, clear and start by match with CR50
        TCL50=0x07;             // set slowest clock frequency
        CR50=0xff;              // set compare count
        PR1L |= 0x02;           // set low priority interrupt
        TMMK50=1;               // mask interrupt
        TMIF50=0;               // clear interrupt request


}
/*****************************************************************************
*
* Function:       StartTimer50
* Parameters:     none
* Returns:        nothing
* Date:           January 21, 2005
* Description:    Starts timer/event counter 50 and enables interrupt
*
* Notes:
*
*****************************************************************************/

void StartTimer50(void)
{
        TCE50=1;                // start timer 50
        TMIF50=0;               // clear interrupt request
        TMMK50=0;               // unmask interrupt

}
```

7.7 UART6.h

```
/*****************************************************************************
*
* FILE                    : UART6.h
* DATE                    : February 2006
* DESCRIPTION             : Uart header file for M-78F0537
* CPU TYPE                : 78K0/KE2 - 78F0537D
*
* Notes:          :
*
*****************************************************************************/
#ifndef _UART6_H
#define _UART6_H


#ifdef    UART6_C
#define UART6_GLOBAL
#else
#define UART6_GLOBAL extern
#endif

UART6_GLOBAL char rx_char;
#define BAUD_115200_20 87       // value for 115200 baud at 20.00MHz clock
                                                    // with clock set to fx

// character definitions
#define NULL_CHAR 0x00                      // NULL character
#define TAB_CHAR 0x09                       // Tab character
#define LF_CHAR 0x0a                  // Line Feed character
#define CR_CHAR 0x0d                  // Carriage Return character
#define ESC_CHAR 0x1B                       // Escape character
#define SP_CHAR 0x20                  // Space character
#define PROMPT_CHAR '>'                     // terminal prompt character
#define X_ON 0x11                           // XON control
#define X_OFF 0x13                          // XOFF control


// function prototypes
void InitUART6_8N1(unsigned char baudRate);
void Tx_String(const char *puc);
void Tx_Word(unsigned int data_16_bit);
void Tx_Byte(unsigned char data_8_bit);
void Tx_CRLF(void);
void Tx_Character(char ascii_character);
char Rx_Character(void);


#endif  /* _UART6_H */
```

7.8 epvdata.h

```
/*****************************************************************************
*
* FILE                  : epvdata.h
* DATE                  : February 2006
* DESCRIPTION           : bootload headerfile for M-78F0537
* CPU TYPE              : 78K0/KE2 - uPD78F0537D
*
* Notes:        :
*
*****************************************************************************/
#ifndef _EPVDATA_H
#define _EPVDATA_H


#ifdef   EPVDATA_C
#define EPVDATA_GLOBAL
#else
#define EPVDATA_GLOBAL extern
#endif

#define MDB_SIZE 136    // data buffer size
EPVDATA_GLOBAL unsigned char myDataBuffer[MDB_SIZE];// data buffer for programming
EPVDATA_GLOBAL unsigned char db_index;                  // data buffer index
EPVDATA_GLOBAL unsigned char numberOfWords;         // number of words to be programmed
EPVDATA_GLOBAL unsigned short programmingAddress;// start address for programming data
EPVDATA_GLOBAL unsigned char firstProgram;          // flag for first programming
EPVDATA_GLOBAL unsigned char oddBytes;                  // count of bytes needed to make up word
EPVDATA_GLOBAL unsigned long *validApp_ptr;         // pointer to valid application word
EPVDATA_GLOBAL unsigned long validAppWord; // checksum of application flash area
EPVDATA_GLOBAL unsigned int timeOut;                    // time out counter
EPVDATA_GLOBAL unsigned char i;                         // byte i
EPVDATA_GLOBAL unsigned int int_i;                      // integer i
EPVDATA_GLOBAL unsigned char bootSwapFlag;// boot swap flag
EPVDATA_GLOBAL unsigned char bootDownLoad;          // boot download flag

// general definitions
#define FALSE 0
#define TRUE 1
#define OUTPUT 0
#define INPUT 1

typedef  unsigned short     USHORT;
typedef  unsigned char      UCHAR;

// constant definitions
#define MAX_COUNT 0xffff

#define ENTRY_RAM_SIZE 100

// application definitions
#define FLMD0_CONTROL_PIN P3.0              // port used to set level on FLMD0 pin
#define FLMD0_CONTROL_IO PM3.0              // port input output configuration
```

```
// self-flash programming definitions
#define FIRST_BOOT_BLOCK 4                  // first boot block must be 4 if bootswapping
#define LAST_BOOT_BLOCK 7                   // last boot block must be 3 if bootswapping
#define FIRST_FLASH_BLOCK_C 8                       // first application block in Common Area
#define LAST_FLASH_BLOCK_C 31                       // last application block in Common Area
#define FIRST_FLASH_BLOCK_B 32                      // first application block in Bank Area
#define LAST_FLASH_BLOCK_B 47                       // last application block in Bank Area

#define CS_FLASH_BLOCK        47                    // Bank and Block where checksum is stored
#define CS_FLASH_BANK         5

#define CS_ADDRESS     0xBFFC          // address where checksum is stored in BANK 5

#define NOT_BLANK                      0x1b

// function prototypes
void programDataBuffer(void);
void Enable_SFP(void);
void Disable_SFP(void);
void EraseFlashBlocks(unsigned char firstBank, unsigned char lastBank, unsigned char firstBlock, unsigned char
lastBlock);
void WriteDataBufferToFlash(unsigned char Banknumber, unsigned short writeAddress,unsigned char wordCount);
void VerifyFlashBlocks(unsigned char firstBank, unsigned char lastBank, unsigned char firstBlock, unsigned char
lastBlock);
unsigned long CalculateAppCheckSum(void);
void WriteVerifyAppCheckSum(void);
unsigned char ConfirmOnPrompt(void);
unsigned char CheckBootSwapFlag(void);
void ToggleBootSwapFlag(void);
void performBootSwap(void);

#endif  /* _EPVDATA_H */
```

7.9 getdata.h

```
/******************************************************************************
*
* FILE                    : getdata.h
* DATE                    : January 17, 2005
* DESCRIPTION             : bootload headerfile for M-78F0148H
* CPU TYPE                : 78K0/KF1+ - uPD78F0148H
*
* Notes:          :
*
******************************************************************************/
#ifndef _GETDATA_H
#define _GETDATA_H


#ifdef    GETDATA_C
#define GETDATA_GLOBAL
#else
#define GETDATA_GLOBAL extern
#endif


#define RB_DATA_SIZE 354              // receive buffer data size
#define RB_OVER_SIZE 32               // receive buffer overflow size
#define RB_SIZE (RB_DATA_SIZE+RB_OVER_SIZE)       // total receive buffer size

#define SWITCH_2 P3.1
#define SWITCH_3 P3.2




// function prototypes
void GetFile(void);
unsigned char GetHexByte(void);


#endif  /* _GETDATA_H */
```

```
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  System       : Self programming library(Normal model)
;  File name    : SelfLibrary_normal.asm
;  Version      : 2.00
;  Target CPU   : 78K0/Kx2
;  Last updated : 2005/07/08
;++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

PUBLIC_FlashStart
PUBLIC_FlashEnd
PUBLIC_FlashEnv
PUBLIC_FlashBlockErase
PUBLIC_FlashWordWrite
PUBLIC_FlashBlockVerify
PUBLIC_FlashBlockBlankCheck
PUBLIC_FlashGetInfo
PUBLIC_FlashSetInfo
PUBLIC_CheckFLMD
;PUBLIC      _EEPROMWrite            ;not needed for Self-programming example (removed from code)

;-------------------------------------------------------------------
;      EQU settings
;-------------------------------------------------------------------
FLASH_ENV                  EQU    00H     ;Initialization
FLASH_BLOCK_ERASE          EQU    03H     ;Block erace
FLASH_WORD_WRITE           EQU    04H     ;Word write
FLASH_BLOCK_VERIFY                EQU     06H    ;Block verify
FLASH_BLOCK_BLANKCHECK                    EQU    08H    ;Block blank check
FLASH_GET_INF                     EQU     09H    ;Flash memory information read
FLASH_SET_INF                     EQU     0AH    ;Flash memory information setting
FLASH_CHECK_FLMD           EQU    0EH     ;Mode check
FLASH_EEPROM_WRITE                EQU     17H    ;EEPROM write

FLASHFIRM_PARAMETER_ERROR         EQU     05H    ;Parameter error

BANK_BLC_ERROR                    EQU     0FFH   ;Bank number error(BLOCK)
BANK_ADDR_ERROR                   EQU     0FFFFH ;Bank number error(ADDRESS)


SELF_PROG     CSEG

;-----------------------------------------------------------------
; Function name  : _FlashStart
; Input                : None
; Output         : None
; Destroyed register   : None
; Summary              : Self programming start processing.
;-----------------------------------------------------------------
_FlashStart:
      MOV    PFCMD,#0A5H          ;PFCMD register control
      MOV    FLPMC,#001H          ;FLPMC register control (set value)
      MOV    FLPMC,#0FEH          ;FLPMC register control (inverted set value)
      MOV    FLPMC,#001H          ;FLPMC register control (set value)
      RET
```

```
;-----------------------------------------------------------------
; Function name   : _FlashEnd
; Input                     : None
; Output          : None
; Destroyed register        : None
; Summary                   : Self programming end processing.
;-----------------------------------------------------------------
_FlashEnd:
        MOV     PFCMD,#0A5H         ;PFCMD register control
        MOV     FLPMC,#000H         ;FLPMC register control (set value)
        MOV     FLPMC,#0FFH         ;FLPMC register control (inverted set value)
        MOV     FLPMC,#000H         ;FLPMC register control (set value)
        RET

;-----------------------------------------------------------------
; Function name   : _FlashEnv
; Input                     : AX=Entry RAM address
; Output          : None
; Destroyed register        : None
; Summary                   : Initialization processing of self programming.
;-----------------------------------------------------------------
_FlashEnv:
;Initialization processing
        PUSH    PSW                 ;Save register bank in STACK.
        PUSH    AX
        SEL     RB3                 ;Sets to register bank 3.
        POP     HL                  ;Sets Entry RAM address to HL register
        MOV     C,#FLASH_ENV        ;Sets function number to C register
        CALL    !8100H              ;Calls flash firmware

        MOV     A, #09H
        MOV     [HL+13H], A         ;Set Block Erase Retry Number
        MOV     [HL+14H], A         ;Set Chip Erase Retry Number

        POP     PSW                 ;Restores register bank from STACK.
        RET

;-----------------------------------------------------------------
; Function name   : _FlashBlockErase
; Input                     : AX=Erase bank
;                             STACK=Erase block number
; Output          : BC=Status
; Destroyed register        : AX,BC
; Summary                   : Erases of specified block (1K byte).
;-----------------------------------------------------------------
_FlashBlockErase:
        PUSH    HL

;Calculate Erase block number from block number and bank.
        MOVW    BC,AX
        MOVW    AX,SP
        MOVW    HL,AX
        MOV     A,[HL+4]            ;Read STACK data(=Erase block number)
        MOV     B,A
        MOV     A,C                 ;A...Erase bank, B...Erase block number
```

```
        CALL    !ExchangeBlockNum       ;Block number is calculated from block number and bank.
                                        ;(Return A=Erase block number after it calculates)
        BZ      $FBE_PErr               ;It is error if the bank number is outside the range.

;Block erase processing
        PUSH    PSW                     ;Save register bank in STACK.
        PUSH    AX
        SEL     RB3                     ;Sets to register bank 3.
        POP     AX
        MOV     [HL+3],A                ;Sets entry RAM+3 to Erase block number after it calculates
        MOV     C,#FLASH_BLOCK_ERASE    ;Sets function number to C register
        CALL    !8100H                  ;Calls flash firmware
        POP     PSW                     ;Restores register bank from STACK.

;Get flash firmware error information
        MOV     A,0FEE3H                ;Sets flash firmware error information to return value
                                        ;(0FEE3H = B register of Bank 3)
        BR      FlashBlockErase00

;Parameter error
FBE_PErr:
        MOV     A,#FLASHFIRM_PARAMETER_ERROR  ;Sets parameter error to return value

FlashBlockErase00:
        MOV     C,A
        MOV     B,#00H
        POP     HL
        RET

;-----------------------------------------------------------------
; Function name   : _FlashWordWrite
; Input                 : AX=Address of writing beginning address structure
;                         (Member of structure...Writing starting address
;                                                Bank of writing starting address)
;                         STACK1=Number of writing data
;                         STACK2=Address in writing data buffer
; Output          : BC=Status
; Destroyed register    : AX,BC,DE
; Summary               : Data on RAM is written in the flash memory.
;                         256 bytes or less (Every 4 bytes) are written at a time.
;-----------------------------------------------------------------
_FlashWordWrite:
        PUSH    HL

;Calculate Writing address from writing address and bank.
        MOVW    DE,AX
        MOVW    AX,SP
        MOVW    HL,AX
        MOV     A,[HL+4]                ;Read STACK data(=Number of writing data)
        MOV     B,A
        MOV     A,[HL+6]                ;Read STACK data(=Address in writing data buffer)
        XCH     A,X
        MOV     A,[HL+7]
        MOVW    HL,AX
        MOVW    AX,DE                   ;AX...Address of writing beginning address structure address,
```

```
                                        ;B...Number of writing data,HL...Address in writing data buffer
        CALL    !ExchangeAddress        ;Writing address is calculated from structure member's writing address and
bank
                                        ;(Return AX=Writing address)
        BZ      $FWW_PErr               ;It is error if the bank number is outside the range.

;Word write processing
        PUSH    PSW                     ;Save register bank in STACK.
        PUSH    AX
        PUSH    BC
        PUSH    HL
        SEL     RB3                     ;Sets to register bank 3.
        POP     AX
        MOV     [HL+5],A                ;Sets entry RAM+5 to higher address in writing data buffer
        MOV     A,X
        MOV     [HL+4],A                ;Sets entry RAM+4 to lower address in writing data buffer
        POP     AX
        MOV     [HL+3],A                ;Sets entry RAM+3 to Number of writing data
        MOV     A,X
        MOV     [HL+0],A                ;Sets entry RAM+0 to Writing address lower bytes
        POP     AX
        MOV     [HL+2],A                ;Sets entry RAM+2 to Writing address most higher bytes
        MOV     A,X
        MOV     [HL+1],A                ;Sets entry RAM+1 to Writing address higher bytes
        MOV     C,#FLASH_WORD_WRITE     ;Sets function number to C register
        CALL    !8100H                  ;Calls flash firmware
        POP     PSW                     ;Restores register bank from STACK.

;Get flash firmware error information
        MOV     A,0FEE3H                ;Sets flash firmware error information to return value
                                        ;(0FEE3H = B register of Bank 3)
        BR      FlashWordWrite00

;Parameter error
FWW_PErr:
        MOV     A,#FLASHFIRM_PARAMETER_ERROR  ;Sets parameter error to return value

FlashWordWrite00:
        MOV     C,A
        MOV     B,#00H
        POP     HL
        RET

;-----------------------------------------------------------------
; Function name  : _FlashBlockVerify
; Input                 : AX=Verify bank
;                         STACK=Verify block number
; Output         : BC=Status
; Destroyed register    : AX,BC
; Summary               : Internal verify of specified block (1K byte).
;-----------------------------------------------------------------
_FlashBlockVerify:
        PUSH    HL

;Calculate Verify block number from block number and bank.
```

```
        MOVW BC,AX
        MOVW AX,SP
        MOVW HL,AX
        MOV     A,[HL+4]                ;Read STACK data(=Verify block number)
        MOV     B,A
        MOV     A,C                     ;A...Verify bank, B...Verify block number
        CALL    !ExchangeBlockNum       ;Block number is calculated from block number and bank.
                                        ;(Return A=Verify block number after it calculates)
        BZ      $FBV_PErr               ;It is error if the bank number is outside the range.

;Block verify processing
        PUSH    PSW                     ;Save register bank in STACK.
        PUSH    AX
        SEL     RB3                     ;Sets to register bank 3.
        POP     AX
        MOV     [HL+3],A                ;Sets entry RAM+3 to Verify block number after it calculates
        MOV     C,#FLASH_BLOCK_VERIFY       ;Sets function number to C register
        CALL    !8100H                  ;Calls flash firmware
        POP     PSW                     ;Restores register bank from STACK.

;Get flash firmware error information
        MOV     A,0FEE3H                ;Sets flash firmware error information to return value
                                        ;(0FEE3H = B register of Bank 3)
        BR      FlashBlockVerify00

;Parameter error
FBV_PErr:
        MOV     A,#FLASHFIRM_PARAMETER_ERROR  ;Sets parameter error to return value

FlashBlockVerify00:
        MOV     C,A
        MOV     B,#00H
        POP     HL
        RET

;-----------------------------------------------------------------
; Function name   : _FlashBlockBlankCheck
; Input                   : AX=Blank check bank
;                           STACK=Blank check block number
; Output          : BC=Status
; Destroyed register      : AX,BC
; Summary                 : Blank check of specified block (1K byte).
;-----------------------------------------------------------------
_FlashBlockBlankCheck:
        PUSH    HL

;Calculate Blank check block number from block number and bank.
        MOVW BC,AX
        MOVW AX,SP
        MOVW HL,AX
        MOV     A,[HL+4]                        ;Read STACK data(=Blank check block number)
        MOV     B,A
        MOV     A,C                             ;A...Blank check bank, B...Blank check block number
        CALL    !ExchangeBlockNum               ;Block number is calculated from block number and bank.
                                                ;(Return A=Blank check block number after it calculates)
```

```
        BZ      $FBBC_PErr                      ;It is error if the bank number is outside the range.

;Block blank check processing
        PUSH    PSW                             ;Save register bank in STACK.
        PUSH    AX
        SEL     RB3                             ;Sets to register bank 3.
        POP     AX
        MOV     [HL+3],A                        ;Sets entry RAM+3 to Blank check block number after it calculates
        MOV     C,#FLASH_BLOCK_BLANKCHECK       ;Sets function number to C register
        CALL    !8100H                          ;Calls flash firmware
        POP     PSW                             ;Restores register bank from STACK.

;Get flash firmware error information
        MOV     A,0FEE3H                        ;Sets flash firmware error information to return value
                                                ;(0FEE3H = B register of Bank 3)
        BR      FlashBlockBlankCheck00

;Parameter error
FBBC_PErr:
        MOV     A,#FLASHFIRM_PARAMETER_ERROR    ;Sets parameter error to return value

FlashBlockBlankCheck00:
        MOV     C,A
        MOV     B,#00H
        POP     HL
        RET

;-----------------------------------------------------------------
; Function name         : _FlashGetInfo
; Input                 : AX=Address of flash information acquisition structure
;                         (Member of structure...Option number
;                                                Bank
;                                                Block number)
;                         STACK=The first address in buffer where get data is stored
; Output                : BC=Status
; Destroyed register    : AX,BC,DE
; Summary               : The set up information of the flash memory is read.
;-----------------------------------------------------------------
_FlashGetInfo:
        PUSH    HL

;Check of Option number
        MOVW    BC,AX
        MOVW    AX,SP
        MOVW    HL,AX
        MOV     A,[HL+4]                ;Read STACK data(=The first address in buffer where get data is stored)
        XCH     A,X
        MOV     A,[HL+5]
        XCHW    AX,BC                   ;AX...Address of flash information acquisition structure
                                        ;BC...The first address in buffer where get data is stored
        MOVW    HL,AX
        MOVW    AX,BC
        MOVW    DE,AX
        MOV     A,[HL+0]                ;Read data from flash information acquisition structure(=Option number)
        CMP     A,#05H                  ;Option number = 5 ?
```

```
        BNZ     $FlashGetInfo10         ;NO

;Calculate Block number from block number and bank.
        MOV     X,A
        MOV     A,[HL+2]                ;Read data from flash information acquisition structure(=Block number)
        MOV     B,A
        MOV     A,[HL+1]                ;Read data from flash information acquisition structure(=Bank)
                                        ;A...Bank, B...Block number
        CALL    !ExchangeBlockNum       ;Block number is calculated from block number and bank.
                                        ;(Return A=Block number after it calculates)
        BZ      $FlashGetInfo20         ;It is error if the bank number is outside the range.
        XCH     A,X                     ;A...Option number, X...Block number

;Get info processing(When Option number = 5)
        PUSH    PSW                     ;Save register bank in STACK.
        PUSH    DE
        PUSH    AX
        SEL     RB3                     ;Sets to register bank 3.
        POP     AX
        XCH     A,X
        MOV     [HL+0],A                ;Sets entry RAM+0 to Block number
        MOV     A,X                     ;A...Option number
        BR      FlashGetInfo40

;Check of Option number error
FlashGetInfo10:
        CMP     A,#03H                  ;Option number = 3 ?
        BZ      $FlashGetInfo30         ;YES
        CMP     A,#04H                  ;Option number = 4 ?
        BZ      $FlashGetInfo30         ;YES
FlashGetInfo20:
        MOV     A,#FLASHFIRM_PARAMETER_ERROR  ;The parameter error is returned, except when option NO is
3, 4, and 5.
        BR      FlashGetInfo50

;Get info processing(When Option number = 3,4)
FlashGetInfo30:
        PUSH    PSW                     ;Save register bank in STACK.
        PUSH    DE
        PUSH    AX
        SEL     RB3                     ;Sets to register bank 3.
        POP     AX
FlashGetInfo40:
        MOV     [HL+3],A                ;Sets entry RAM+3 to Option number
        POP     AX
        MOV     [HL+5],A                ;Sets entry RAM+5 to Storage buffer higher address
        MOV     A,X
        MOV     [HL+4],A                ;Sets entry RAM+4 to Storage buffer lower address
        MOV     C,#FLASH_GET_INF        ;Sets function number to C register
        CALL    !8100H                  ;Calls flash firmware
        POP     PSW                     ;Restores register bank from STACK.

;Calculate Address from Storage buffer and bank.Nothing to do when Option number = 3or4 or Bank = 0
;or Block number(Previous) < 32 or Block number(Previous) >= 48.
;A=Option number, B=Bank, C...Block number(Previous), DE=Storage buffer first address of get data
```

```
        CMP     A,#05H                  ;Option number = 5 ?
        BNZ     $ReturnAddress_end      ;NO
        MOV     A,B
        CMP     A,#0                    ;Bank = 0 ?
        BZ      $ReturnAddress_end      ;YES
        XCH     A,C
        CMP     A,#32                   ;Block number(Previous) < 32 ?
        BC      $ReturnAddress_end      ;YES
        CMP     A,#48                   ;Block number(Previous) >= 48 ?
        BNC     $ReturnAddress_end      ;YES
        MOV     A,C

;Calculation of address(40H*Bank is pulled from address in two high rank bytes.Lower address is the state as it is.)
        XCHW    AX,DE
        MOVW    HL,AX
        MOV     A,[HL+1]
        MOV     X,A
        MOV     A,[HL+2]                ;A...Most higher address, X...Higher address
        XCHW    AX,DE                   ;A...Bank, D...Most higher address, E...Higher address
        MOV     [HL+2],A                ;Sets Storage buffer+2 to Bank.
        MOV     X,#0
        ROL     A,1
        ROL     A,1
        ROL     A,1
        ROL     A,1
        ROL     A,1
        ROLC    A,1
        XCH     A,X
        ROLC    A,1                     ;AX=40H*Bank
        XCHW    AX,DE
        XCH     A,X
        SUB     A,E
        XCH     A,X
        SUBC    A,D
        MOV     A,X
        MOV     [HL+1],A                ;Sets Storage buffer+1 to Calculated address(higher).
ReturnAddress_end:

;Get flash firmware error information
        MOV     A,0FEE3H                ;Sets flash firmware error information to return value
                                        ;(0FEE3H = B register of Bank 3)
FlashGetInfo50:
        MOV     C,A
        MOV     B,#00H
        POP     HL
        RET

;-----------------------------------------------------------------
; Function name     : _FlashSetInfo
; Input             : AX=Flash information data
; Output            : BC=Status
; Destroyed register    : A,BC
; Summary           : Setting of flash information.
;-----------------------------------------------------------------
_FlashSetInfo:
```

```
;Set infomation processing
        MOV     A,X
        PUSH    AX                          ;Save Flash information data in STACK.
        PUSH    PSW                         ;Save register bank in STACK.
        SEL     RB3                         ;Sets to register bank 3.
        MOVW    AX,SP
        ADDW    AX,#2
        MOV     [HL+5],A                    ;Sets entry RAM+5 to higher address of flash information data secured for
stack
        MOV     A,X
        MOV     [HL+4],A                    ;Sets entry RAM+4 to lower address of flash information data secured for
stack
        MOV     C,#FLASH_SET_INF            ;Sets function number to C register
        CALL    !8100H                      ;Calls flash firmware
        POP     PSW                         ;Restores register bank from STACK.
        POP     AX

;Get flash firmware error information
        MOV     A,0FEE3H                    ;Sets flash firmware error information to return value
                                            ;(0FEE3H = B register of Bank 3)
        MOV     C,A
        MOV     B,#00H
        RET

;-----------------------------------------------------------------
; Function name   : _CheckFLMD
; Input                 : None
; Output          : BC=Status
; Destroyed register    : A,BC
; Summary               : Checks voltage level of FLMD pin.
;-----------------------------------------------------------------
_CheckFLMD:
;Set infomation processing
        PUSH    PSW                         ;Save register bank in STACK.
        SEL     RB3                         ;Sets to register bank 3.
        MOV     C,#FLASH_CHECK_FLMD         ;Sets function number to C register
        CALL    !8100H                      ;Calls flash firmware
        POP     PSW                         ;Restores register bank from STACK.

;Get flash firmware error information
        MOV     A,0FEE3H                    ;Sets flash firmware error information to return value
                                            ;(0FEE3H = B register of Bank 3)
        MOV     C,A
        MOV     B,#00H
        RET

;-----------------------------------------------------------------
; Function name : ExchangeBlockNum
; Input           : A=Bank
;                   B=Block number
; Output : A=Block number(New)
;                   B=Bank
;                   C=Block number(Previous)
; Summary         : Block number is converted into the real address
;                   from bank information.
```

```
;-----------------------------------------------------------------
ExchangeBlockNum:
;It calculates from 32 to 47 block number.
        XCH     A,B
        CMP     A,#32
        BC      $EBN_end
        CMP     A,#48
        BNC     $EBN_end

;Calculation of block number(Bank*16 is added to block number.)
        XCH     A,B
        MOV     C,A                     ;C...Bank
        CMP     A,#6
        BNC     $EBN_error_end
        ROL     A,1
        ROL     A,1
        ROL     A,1
        ROL     A,1                     ;A=16*Bank
        ADD     A,B
        XCH     A,C
        XCH     A,B
        XCH     A,C                     ;A=Block number after it calculates, B=Bank,
                                        ;C=Block number before it calculates
        BR      EBN_end

;Bank error
EBN_error_end:
        MOV     A,#BANK_BLC_ERROR ;Return error number

EBN_end:
        CMP     A,#BANK_BLC_ERROR ;Bank error ?
        RET

;-----------------------------------------------------------------
; Function name : ExchangeAddress
; Input         : AX=Address of writing beginning address structure
;                       (Member of structure...Writing starting address
;                                                Bank of writing starting address)
; Output : AX=Writing starting address(Address in two high rank bytes)
;                 C=Writing starting address(Lower address)
; Summary       : Writing starting address of structure is converted
;                 into the real address from bank information.
;-----------------------------------------------------------------
ExchangeAddress:
        PUSH    HL

;It calculates from 8000H to BFFFH address.
        MOVW    HL,AX
        MOV     A,[HL+0]                ;Read data from writing beginning address structure(=Write address)
        MOV     X,A
        MOV     A,[HL+1]
        CMPW    AX,#8000H
        BC      $EA_end
        CMPW    AX,#0C000H
        BNC     $EA_end
```

```
;Calculation of address(Bank*40H is added to address in two high rank bytes.Lower address is the state as it is.)
        MOV     D,A
        XCH     A,X
        MOV     C,A
        MOV     X,#0
        MOV     A,[HL+2]                ;Read data from writing beginning address structure
                                        ;(=Bank of writing starting address)
        CMP     A,#6
        BNC     $EA_error_end
        ROL     A,1
        ROL     A,1
        ROL     A,1
        ROL     A,1
        ROL     A,1
        ROLC    A,1
        XCH     A,X
        ROLC    A,1                     ;AX=40H*Bank
        XCH     A,X
        ADD     A,D                     ;Addition of Higher address
        XCH     A,X
        ADDC    A,#0                    ;Addition of Most higher address
                                        ;A....Most higher address after it calculates
                                        ;X....higher address after it calculates, C...Lower address
        BR      EA_normal_end

;Bank error
EA_error_end:
        MOVW    AX,#BANK_ADDR_ERROR
        BR      EA_normal_end

EA_end:
        XCH     A,X
        MOV     C,A
        MOV     A,#0                    ;A....Most higher address after it calculates
                                        ;X....higher address after it calculates, C...Lower address
EA_normal_end:
        POP     HL
        CMPW    AX,#BANK_ADDR_ERROR         ;Bank error ?
        RET


        END
```

7.10 option.asm

```
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
;  System        : Option byte setting
;  File name     : option.asm
;  Version       :  0.00
;  Target CPU    : 78K0/Kx2
;+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++

; specify option byte for 78F0537 family
; option byte is at 0080H or 1080H (when boot swap used)
;  OB.7 = 0              bit 7 set to zero
;  OB.6 = WINDOW1        WINDOW1/2
;  OB.5 = WINDOW2        00 = 25%  WDT window open period
;                        01 = 50%  WDT window open period
;                        10 = 75%  WDT window open period
;                        11 = 100%  WDT window open period (default)
;  OB.4 = WDTON          0=WDT count disabled, 1=WDT count enabled
;  OB.3 = WDCS2          WDCS[2:0] = WDT overflow time
;  OB.2 = WDCS1
;  OB.1 = WDCS0
;  OB.0 = INTERNALOSC    0=can be stopped by sw, 1=cannot be stopped

OPT     CSEG AT 0080H

OPTION:         DB 01111110B    ; bit 7   = 0
                                ; bit 65  = 11  = 100%  window open period
                                ; bit 4   = 1   = WDT enabled
                                ; bit 321 = 111 = 496 ms.
                                ; bit 0   = 0   = internal-osc can be stopped by software
```