To our customers,

Old Company Name in Catalogs and Other Documents

On April 1st, 2010, NEC Electronics Corporation merged with Renesas Technology Corporation, and Renesas Electronics Corporation took over all the business of both companies. Therefore, although the old company name remains in this document, it is a valid Renesas Electronics document. We appreciate your understanding.

Renesas Electronics website: http://www.renesas.com

April 1st, 2010
Renesas Electronics Corporation

Issued by: Renesas Electronics Corporation (http://www.renesas.com)

Send any inquiries to http://www.renesas.com/inquiry.

Notice

1. All information included in this document is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas Electronics products listed herein, please confirm the latest product information with a Renesas Electronics sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas Electronics such as that disclosed through our website.
2. Renesas 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 Renesas Electronics products or technical information described in this document. No license, express, implied or otherwise, is granted hereby under any patents, copyrights or other intellectual property rights of Renesas Electronics or others.
3. You should not alter, modify, copy, or otherwise misappropriate any Renesas Electronics product, whether in whole or in part.
4. Descriptions of circuits, software and other related information in this document are provided only to illustrate the operation of semiconductor products and application examples. You are fully responsible for the incorporation of these circuits, software, and information in the design of your equipment. Renesas Electronics assumes no responsibility for any losses incurred by you or third parties arising from the use of these circuits, software, or information.
5. When exporting the products or technology described in this document, you should comply with the applicable export control laws and regulations and follow the procedures required by such laws and regulations. You should not use Renesas Electronics products or the technology described in this document for any purpose relating to military applications or use by the military, including but not limited to the development of weapons of mass destruction. Renesas Electronics products and technology may not be used for or incorporated into any products or systems whose manufacture, use, or sale is prohibited under any applicable domestic or foreign laws or regulations.
6. Renesas Electronics has used reasonable care in preparing the information included in this document, but Renesas Electronics does not warrant that such information is error free. Renesas Electronics assumes no liability whatsoever for any damages incurred by you resulting from errors in or omissions from the information included herein.
7. Renesas Electronics products are classified according to the following three quality grades: "Standard", "High Quality", and "Specific". The recommended applications for each Renesas Electronics product depends on the product's quality grade, as indicated below. You must check the quality grade of each Renesas Electronics product before using it in a particular application. You may not use any Renesas Electronics product for any application categorized as "Specific" without the prior written consent of Renesas Electronics. Further, you may not use any Renesas Electronics product for any application for which it is not intended without the prior written consent of Renesas Electronics. Renesas Electronics shall not be in any way liable for any damages or losses incurred by you or third parties arising from the use of any Renesas Electronics product for an application categorized as "Specific" or for which the product is not intended where you have failed to obtain the prior written consent of Renesas Electronics. The quality grade of each Renesas Electronics product is "Standard" unless otherwise expressly specified in a Renesas Electronics data sheets or data books, etc.

   "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.

   "High Quality": 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; medical equipment or systems for life support (e.g. artificial life support devices or systems), surgical implantations, or healthcare intervention (e.g. excision, etc.), and any other applications or purposes that pose a direct threat to human life.
8. You should use the Renesas Electronics products described in this document within the range specified by Renesas Electronics, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas Electronics shall have no liability for malfunctions or damages arising out of the use of Renesas Electronics products beyond such specified ranges.
9. Although Renesas Electronics endeavors to improve the quality and reliability of its products, semiconductor products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Further, Renesas Electronics products are not subject to radiation resistance design. Please be sure to implement safety measures to guard them against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas Electronics product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other appropriate measures. Because the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
10. Please contact a Renesas Electronics sales office for details as to environmental matters such as the environmental compatibility of each Renesas Electronics product. Please use Renesas Electronics products in compliance with all applicable laws and regulations that regulate the inclusion or use of controlled substances, including without limitation, the EU RoHS Directive. Renesas Electronics assumes no liability for damages or losses occurring as a result of your noncompliance with applicable laws and regulations.
11. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written consent of Renesas Electronics.
12. Please contact a Renesas Electronics sales office if you have any questions regarding the information contained in this document or Renesas Electronics products, or if you have any other inquiries.

(Note 1) "Renesas Electronics" as used in this document means Renesas Electronics Corporation and also includes its majority-owned subsidiaries.

(Note 2) "Renesas Electronics product(s)" means any product developed or manufactured by or for Renesas Electronics.

M16C/62P

Example CPU Rewrite Mode 0 FLASH Programming Using IAR C-Compiler

Introduction

One of the most useful features of microcontrollers which incorporate FLASH memory is their ability to 'self program' their FLASH memory.

With Renesas M16C microcontrollers when this FLASH programming is occurring in CPU rewrite mode 0, the application has to be executing from a memory source other than FLASH. Typically this is the internal RAM of the device. With devices such as the M16C/62P series, it is typically internal RAM. The processes of relocating and executing code from RAM can pose several problems, more of which will be discussed later.

Aspects associated with FLASH programming are discussed in several application notes. Examples of these are: Apps Notes REG05B0021-0100 and REG05B0025-0100.

It is recommended that these three application notes be read either in conjunction or prior to this application note.

It is the aim of this Application Note to bring together all of the concepts discussed in the earlier Application Notes into one simple example.

The Application Note will show how a M16C/62P can reprogram an ADC value into a user defined FLASH block in response to external interrupt.

Contents

An Overview of FLASH Programming

It was mentioned in the introduction of the application note that when the FLASH memory of an M16C device is being erased or programmed in CPU rewrite mode 0, then the application code has to be executing from a memory other than FLASH, typically RAM.

At first the solution to this 'problem' seems straightforward enough. At runtime copy the program or erase routine from FLASH into RAM and call it via a function pointer. In many cases this method will work but cannot be guaranteed - the reason being that any jumps within the code or to subroutines may refer to absolute addresses. Therefore, the code may be executing correctly in RAM and then jump back into the FLASH unexpectedly. This can be avoided by using only branch statements that use offsets relative to the program counter but unfortunately with the current M16C tools there is no way to force the output of position independent code exclusively utilising branches.

The solution to this problem is to link the code that must run from RAM to the actual RAM addresses at build time. This can introduce further problems. The first is that of library routines. If a RAM based function is part of a larger project then it may happily run from RAM but may feature calls to library routines that are linked to FLASH addresses causing accesses to FLASH memory at undesirable moments during execution. Even something as innocuous as the C statement below can result in a library call.

```
i = 1 << some_variable;
```

Simply looking through the C source and avoiding calls to functions such as 'printf' is not enough to guarantee that there are no library calls to FLASH based routines.

The second issue concerning copying functions from FLASH to RAM is that of constant data. If the RAM routine makes reference to constant data, including items such as string literals, this can cause the FLASH memory to be accessed.

A third consideration is how to get code that is linked to RAM into FLASH for storage at build time and then back into RAM at runtime for execution.

A solution to these problems is to place the entire RAM based routines into completely separate projects with all the code, variable and constant data linked to the RAM addresses. This eliminates the problems of jumps back into FLASH for code, libraries and constant data. Getting this code from the RAM addresses into the FLASH for storage at build time can be achieved by using the 'motice_cl' utility and method described in Application Note REG05B0021-0100.

This utility converts an S-record file into a constant 'C' array. For example, a FLASH erasing function is built as a separate project and linked to RAM. The linker is configured so that it outputs an S-record file for this project. This S-record is processed by 'motice_cl' which converts it into a constant 'C' array which can be included into the Application project. As the array is constant data it resides in the FLASH. When the erase routine is to be called by the Application the constant array data is copied to the correct place in RAM and called by a function pointer. While the erase routine is executing only RAM is accessed for program code and data as this is all the routine knows about as it has been linked to RAM addresses in a separate project.

The above method relies on 3 things being known at runtime. These are:

1. The start address that the RAM code should be copied to from FLASH. This is achieved by storing the constant data as part of a structure which contains the start address (put there by 'motice_cl' from the s-record) and the length of the data.
2. The size of the data to be copied to RAM so the copying routine knows how much data to move. See the explanation above for how this is known.
3. If the RAM based code contains multiple functions, e.g. erase and delay routines, the start addresses for these functions must be known so they can be correctly called via function pointers. This can be achieved by loading these addresses into a 'vector' table starting at the beginning of the RAM code area. Although the addresses of the functions may change, the location of where the value and order of these are stored does not and is known by the Application. All the Application must do is read the correct address and call the function via a pointer.

Software Overview

All of the software for 'FlashTraining' was written using IAR Workbench with compiler version V2.12A. The source code and IAR Workbench projects are available for download with this apps note.

As detailed in the previous section, one of the ways to avoid the problems associated with FLASH programming is to place all of the RAM based routines into separate projects. This is the technique that was used to develop this example.

Table 1 details the projects described by this apps note.

Name	Comment
Flash	Project that contains all code required to implement FLASH erasing and programming. The S-Record generated by this file is converted using motice_cl[1] to a constant 'C' structure called flash_converted.c This file is added to the application project FlashTraining
FlashTraining	Project that contains the files required for the application. 2 addition files are added to the project: ferase_converted.c & fprogram_converted.c These 2 files contain the erasing and programming execution code pre linked to the RAM address from which they will execute. The application copies this code from FLASH to RAM when CPU rewrite mode 0 Mode FLASH programming is required.

[1]Refer to Application Note REG05B0021-0100

Table 1. Projects

For the 'FlashTraining' project the 'Debug' build can be used with the KD30 ROM monitor debugger. The 'Release' build is designed to be programmed directly into the flash of the microcontroller. For the 'Flash' project only the 'Release' build is valid.

Screen shots taken from IAR Workbench are shown in figure 1 showing the two projects.

It is mentioned in table 1 that pre linked execution codes are copied from the FLASH memory to internal RAM as and when required. To ensure that no data is corrupted when the code is copied up to RAM, it is necessary to reserve an area of RAM. This is done via the linker settings.

Figures 2 and 3 show the 'FlashTraining' memory map and linker settings.

‘FlashTraining’ IAR Workbench Projects

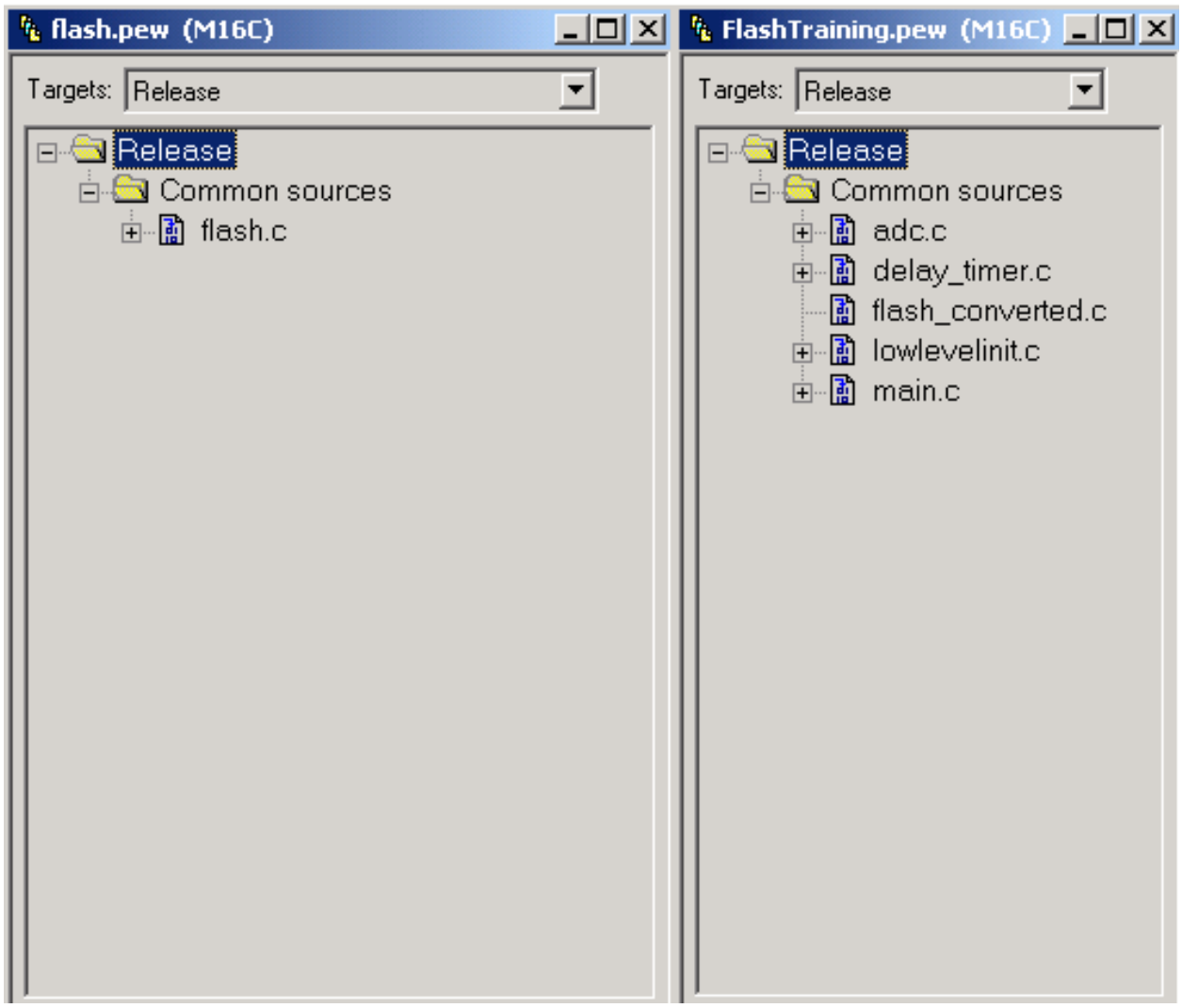

Figure 1. IAR Workbench Projects

‘FlashTraining’ Memory Map

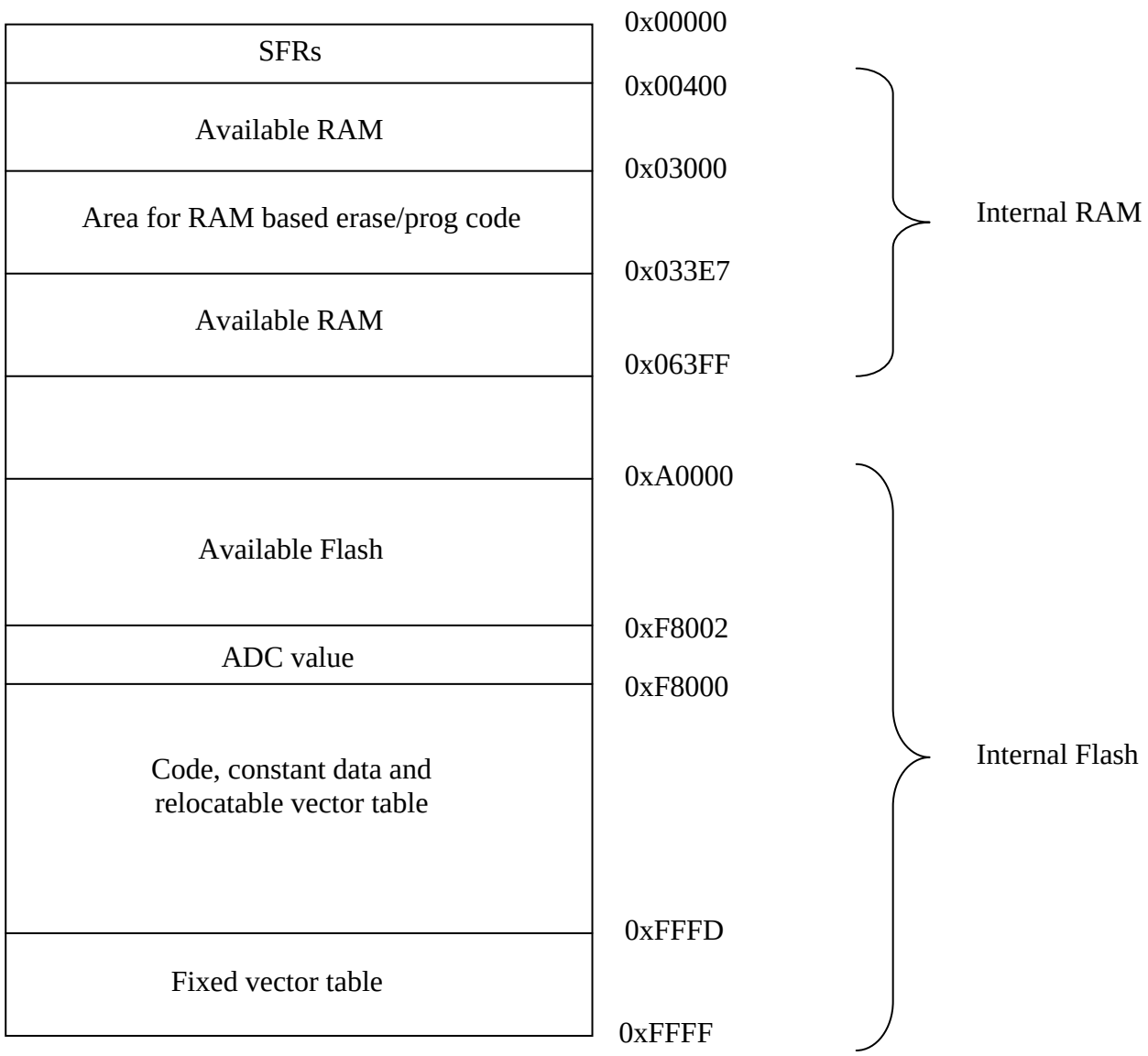

Figure 2. FlashTraining Project Memory Map

‘FlashTraining’ Linker Command File

```
//====================================================================
//  IAR XLINK command file for the M16C/R8C IAR C/EC++ Compiler
//
//  This is an example XLINK command file for use with the
//  IAR C libraries and M16C derivatives.
//
//  Copyright 1995 - 2003 IAR Systems. All rights reserved.
//
//  Usage:  xlink  your_file(s)  -f this_file  clm16cxxx
//
//  $Revision: 1.6 $
//====================================================================

//====================================================================
// The M16C IAR C/EC++ Compiler places code and data into named
// segments which are referred to by the IAR XLINK Linker. The
// table below shows the available segments.
//
// SEGMENT REFERENCE
// =================
//
// Segment   Description
// -------   -----------
// BITVARS   Bit variables.
// CODE      The program code.
// CSTACK    The stack used by C or Embedded C++ programs.
// CSTART    The startup code.
//
// x_AC      Non-initialized located const objects.
// x_AN      Non-initialized located non-const objects.
// x_C       Constant data, including string literals.
// x_I       Initialized data.
// x_ID      Data that is copied to x_I by cstartup.
// x_N       Uninitialized data.
// x_Z       zero initialized data.
//
// Where x can be one of:
//    DATA13  (Range: 0-0x1FFF)
//    DATA16  (Range: 0-0xFFFF)
//    DATA20  (Range: 0-0xFFFFF)
//    FAR     (Range: 0-0xFFFFF)
//
// DIFUNCT   Pointers to code, typically EC++ constructors
// FLIST     Jump table for __tiny_func functions.
// HEAP      The heap data used by malloc and free.
// INTVEC    Contains reset and interrupt vectors.
// INTVEC1   Contains the fixed reset and interrupt vectors.
// ISTACK    The stack used by interrupts and exceptions.
//====================================================================

// Define CPU
-cm16c

//====================================================================
// USER DEFINITIONS
// Please customize according to your specific derivative!
//====================================================================

// Size of the user stack
-D_CSTACK_SIZE=400

// Size of the interrupt stack
-D_ISTACK_SIZE=40
```

```
// Memory areas available for the application
-D_USER_RAM_BEGIN=400
-D_USER_RAM_END=63FF
-D_USER_ROM_BEGIN=FE000
-D_USER_ROM_END=FFFFE
-D_CSTART_ROM_BEGIN=FEA00

// Relocatable "bit" variables, range
-D_BITVAR_BEGIN=2000 // 8*400 hex
-D_BITVAR_END=FFFF

// ID code written to ROM memory for the ID Code Check Function
-D_ID_CODE_1=0
-D_ID_CODE_2=0
-D_ID_CODE_3=0
-D_ID_CODE_4=0
-D_ID_CODE_5=0
-D_ID_CODE_6=0
-D_ID_CODE_7=0

// ========================
//      DATA13 RAM
// ========================

-Z(NEAR)DATA13_AN=0-1FFF
-Z(NEAR)DATA13_I=_USER_RAM_BEGIN-1FFF
-Z(NEAR)DATA13_Z,DATA13_N

// Relocatable "bit" segment. As BITVARS contains bit addresses,
// the desired (byte) address has to be multiplied by 8.
-Z(BIT)BITVARS=_BITVAR_BEGIN-_BITVAR_END

// ========================
//       DATA16 RAM
// ========================

// Set up user stack
-Z(NEAR)CSTACK+_CSTACK_SIZE=_USER_RAM_BEGIN-_USER_RAM_END

// Set up interrupt stack
-Z(NEAR)ISTACK+_ISTACK_SIZE

// Near variables
-Z(NEAR)DATA16_I,DATA16_Z,DATA16_N,DATA16_AN

// User defined near DATA segments

// ========================
//       DATA16 ROM
// ========================

// Constant segments (in ROM), reachable for near pointers
-Z(NEARCONST)DATA16_C=8000-FFFF

// User defined near CONST segments

// ========================
//     FAR/DATA20 RAM
// ========================

// Far and huge data segments
-Z(FAR)FAR_I,FAR_Z,FAR_N,FAR_AN=_USER_RAM_BEGIN-_USER_RAM_END
-Z(HUGE)DATA20_I,DATA20_Z,DATA20_N,DATA20_AN
// Reserved space for RAM based flash erasing and programming code
-Z(HUGE)RESERVED=3000-33E7

// User defined far & huge DATA segments
```

```
// ======================
//     FAR/DATA20 ROM
// ======================

// Constant and initializer segments (in ROM)
-Z(FARCONST)FAR_ID=_USER_ROM_BEGIN-_USER_ROM_END
-Z(FARCONST)FAR_C
-Z(HUGECONST)DATA20_C,DATA20_ID,CHECKSUM
-Z(FARCONST)DATA16_ID,DATA13_ID,DIFUNCT

// CODE segments
-Z(HUGECODE)CODE=_USER_ROM_BEGIN-_USER_ROM_END

// TINYFUNC code must be located above 0xF000
-Z(HUGECODE)TINYFUNC=F0000-_USER_ROM_END

// Startup code
-Z(HUGECODE)CSTART=_CSTART_ROM_BEGIN-_USER_ROM_END

// User defined CODE segments

// User defined far & huge CONST segments

// Variable vector table
-Z(CONST)INTVEC=_CSTART_ROM_BEGIN-_USER_ROM_END

// Special page table
-Z(CONST)FLIST=FFE00-FFFDB

// Fixed interrupt vector table
-Z(CONST)INTVEC1=FFFDC-FFFFF

// ========================
// IAR C library formatting
// ========================
-e_small_write=_formatted_write
-e_medium_read=_formatted_read

// ========================
// Output files
// ========================
// Use the -O option to create one or more output files
// at the same link session. Formats flags, file name and
// extension is optional. Please un-comment the wanted
// output formats below.
//
// CAUTION: Do not combine ohter output formats with -rt (special
// UBROF for Terminal I/O in C-SPY). Output files are valid but
// contain code that expects to be run under C-SPY.

// Motorola output
-Omotorola=.mot

// IEEE-695 output with format flags for the Renesas debugger
//-Oieee695,lbm=.x30

// ELF/DWARF output with format flags for the Renesas debugger
//-Oelf,spc=.elf
```

Figure 4. Linker Command File

‘FlashTraining’ Hardware Overview

The ‘FlashTraining’ application was developed using a M16C/62P. No additional hardware is required as the application makes use of the on-board pot connected to the ADC input, the INT switch and the 8 LEDs.

‘FlashTraining’ Application Overview

Timer A is configured generate a compare match interrupt at 1ms intervals.

The ADC is configured to convert the analog voltage supplied by the pot ‘VR1’ in repeat mode.

External interrupt INT0 is configured to be generated when INT0 is activated.

When the M16C/62P is powered and comes out of reset it initialises and starts the ADC and timer A and also configures port 2 as an output as this is the port to which the 8 LEDs are connected. The RAM based code is copied from flash into RAM so it can be called upon when flash erasing and programming is required.

The software contains a global variable called ‘UpdateFlashFlag’ which is set in the INT0 ISR (interrupt service routine) when the INT0 switch is activated. When ‘UpdateFlashFlag’ is set flash block 4 is erased, ADC channel AN0 is read and the 10-bit value is programmed into flash memory address 0xF8000. A function pointer is used to call the RAM based erase and program code.

The code reads the value stored at 0xF8000 and delays by this number of milliseconds using timer A as a delay timer. After each delay the pattern displayed on the LEDs is changed so that a chaser type effect is generated. Execution then loops reading the stored value, delaying and updating the LED pattern. Whenever the INT0 interrupt occurs the ISR sets the ‘UpdateFlashFlag’ and a new ADC (delay) value is programmed into the flash memory. Therefore by altering the position of the potentiometer and pressing the INT0 switch the speed of the LED chaser sequence can be changed.

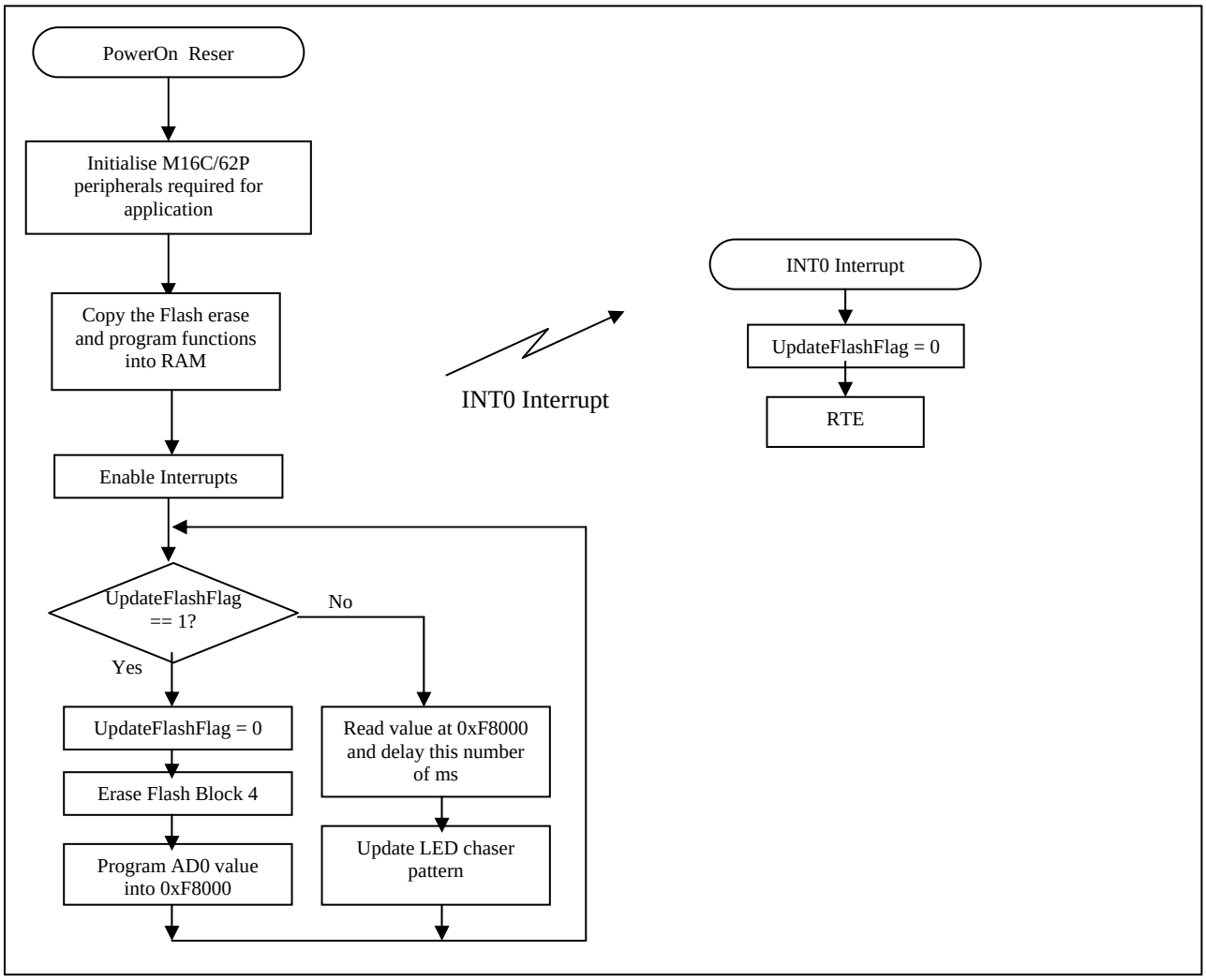

Figure 4.Basic Program Flow.

‘FlashTraining’ Code

When the M16C/62P is powered and comes out of reset the reset vector (address 0xFFFFC) is read. Code execution begins at this address pointed to by the reset vector.

The library version of the CSTARTUP code is contained at this address and starts executing. The CSTARTUP code calls the user generated ‘__low_level_init’ function contained in ‘lowlevelinit.c’.

lowlevelinit.c:

```
#include <intrinsics.h>
#include "iom16c62p.h"

unsigned char __low_level_init(void)
{
//     low_level_init is called as part of CSTARTUP

//  NOTE: PLL register bits PLC02 -> PLC00 can only be set once after reset.
//     This is set up for 24MHz operation running off a 6MHz crystal.
//  Note if you are building for use with the ROM monitor and KD30 the below
//  intialisation should remain commented out so that the PLL isn't set twice.

#ifdef KD30
       return(1);
#else
       // In the event of 'program runaway', the PRCR (Protect Register) will try and restrict
access
       // to the following registers:
       // CM0 - System Clock Control Register 0
       // CM1 - System Clock Control Register 1
       // CM2 - Oscillation Stop Detection Register
       // PLC0 - PLL Control Register 0
       // PCLKR - Peripheral Clock Select Register
       // PM0 - Processor Mode Register 0
       // PM1 - Processor Mode Register 1
       // PM2 - Processor Mode Register 2
       // TB2SC - Timer B2 Special Mode Register
       // INVC0 - Three Phase PWM Control Register 0
       // INVC1 - Three Phase PWM Control Register 1
       // PD9 - Port 9 Direction Register
       // S3C - Serial I/O Control Register 3
       // S4C - Serial I/O Control Register 4
       // VCR2 - Voltage Detection Register 2
       // D4INT - Voltage Down Detection Interrupt Register

       // PRCR - Protect Register
       //------------------------------------------------------------------------
       PRCR_bit.PRC3 = 1;       // 0 - Write Protected  : VCR2, D4INT
                                // 1 - Write Enabled    : VCR2, D4INT

       PRCR_bit.PRC2 = 1;       // 0 - Write Protected  : PD9, S3C, S4C
                                // 1 - Write Enabled    : PD9, S3C, S4C

       PRCR_bit.PRC1 = 1;       // 0 - Write Protected  : PM0, PM1, PM2, TB2SC, INV0, INVC1
                                // 1 - Write Enabled    : PM0, PM1, PM2, TB2SC, INV0, INVC1

       PRCR_bit.PRC0 = 1;       // 0 - Write Protected  : CM0, CM1, CM2, PLC0, PCLKR
                                // 1 - Write Enabled    : CM0, CM1, CM2, PLC0, PCLKR

       // PM0 - Processor Mode Register 0
       //------------------------------------------------------------------------
       PM0_bit.PM07 = 1;        // 0 - BCLK is output
                                // 1 - BCLK is not output (Pin is high impdance)
```

```
        PM0_bit.PM06 = 1;       // 0 - Port 4.0 -> Port 4.3 are Address Pins
                                // 1 - Port 4.0 -> Port 4.3 are Port Pins

        PM0_bit.PM05 = 0;       //     PM05    PM04
        PM0_bit.PM04 = 0;       //     0       0               - Multiplexed bus is unused
                                //     0       1               - Allocated to /CS2 space
                                //     1       0               - Allocated to /CS1 space
                                //     1       1               - Allocated to entire /CS space

        PM0_bit.PM03 = 0;       // Software Reset Bit
                                // Setting this bit to '1' resets the micro

        PM0_bit.PM02 = 0;       // Only valid in Memory Expansion or Microprocessor
                                // 0 - /RD, /BHE, /WR
                                // 1 - /RD, /WRH, /WRL

        PM0_bit.PM01 = 0;       //     PM01    PM00
        PM0_bit.PM00 = 0;       //     0       0               - Single Chip Mode
                                //     0       1               - Memory Expansion Mode
                                //     1       0               - MUST NOT BE SET
                                //     1       1               - Microprocessor mode

        // PM1 - Processor Mode Register 1
        //-----------------------------------------------------------------------
        PM1_bit.PM17 = 0;       // 0 - No wait state
                                // 1 - 1 wait state

        PM1_bit.PM15 = 0;       //     PM15    PM14
        PM1_bit.PM14 = 0;       //     0       0               - 1 Mbyte Mode
                                //     0       1               - MUST NOT BE SET
                                //     1       0               - MUST NOT BE SET
                                //     1       1               - 4 Mbyte Mode

        PM1_bit.PM13 = 1;       // Internal Addresses
                                // 0 -  RAM:   H'400 -> H'3FFF accessable
                                //             ROM:    H'D000 -> H'FFFF accessable
                                // 1 -  RAM:   Entire area is accessable
                                //             ROM:    Entire area is accessable
                                // External Addresses
                                // 0 -  RAM:   H'4000 -> H'7FFF are accessable
                                //             ROM:    H'80000 -> H'CFFFF are accessable
                                // 1 -  RAM:   H'4000 -> H'7FFF are reserved
                                //             ROM:    H'80000 -> H'CFFFF are reserved

        PM1_bit.PM12 = 0;       // 0 - Watchdog Generates Interrupt
                                // 1 - Watchdog Resets Device

        PM1_bit.PM11 = 1;       // 0 - Port 3.4 -> Port 3.7 are Address Pins
                                // 1 - Port 3.4 -> Port 3.7 are Port Pins

        PM1_bit.PM10 = 1;       // 0 - H'8000 -> H'26FFF (Block A) Disabled
                                // 1 - H'8000 -> H'26FFF (Block A) Enabled

        // CM1 - System Clock Control Register 1
        //-----------------------------------------------------------------------
        CM1_bit.CM17 = 0;       // Main Clock Divide Ratio
        CM1_bit.CM16 = 0;       //     CM17    CM16
                                //     0       0               - No Division
                                //     0       1               - Divide by 2
                                //     1       0               - Divide by 4
                                //     1       1               - Divide by 16

        CM1_bit.CM15 = 1;       // 0 - Xin - Xout Drive Capacity LOW
                                // 1 - Xin - Xout Drive Capacity HIGH

        CM1_bit.CM11 = 0;       // 0 - System Clock is Main Clock
                                // 1 - System Clock is PLL Clock
```

```
        CM1_bit.CM10 = 0;        // 0 - Clock ON
                                 // 1 - ALL Clocks OFF

        // CM0 - System Clock Control Register 0
        //-------------------------------------------------------------------------
        CM0_bit.CM07 = 0;        // 0 - System Clock is Main Clock, PLL Clock or Ring Oscillator
Clock
                                 // 1 - System Clock is Sub Clock

        CM0_bit.CM06 = 0;        // 0 - Main Clock is set by bit CM17 & CM16 in Register, CM1
                                 // 1 - Main Clock is divided by 8

        CM0_bit.CM05 = 0;        // 0 - Main Clock is ON
                                 // 1 - Main Clock is OFF

        CM0_bit.CM04 = 0;        // 0 - Pins P8.6 & P8.7 are I/O pins
                                 // 1 - Pins P8.6 & P8.7 are Sub Clock Oscillator pins

        CM0_bit.CM03 = 1;        // 0 - XCin - XCout Drive Capacity LOW
                                 // 1 - XCin - XCout Drive Capacity HIGH

        CM0_bit.CM02 = 0;        // 0 - DO NOT stop peripheral function clock in wait mode
                                 // 1 - Stop peripheral function clock in wait mode

        CM0_bit.CM01 = 0;        // Clock Output Selection
        CM0_bit.CM00 = 0;        //      CM01    CM00
                                 //      0       0               - P5.7 is I/O Port
                                 //      0       1               - fc is Output: Main Clock, PLL
Clock or Ring Oscillator Clock
                                 //      1       0               - f8 is Output: fc / 8
                                 //      1       1               - f32 is Output: Sub Clock / 32

        // CM2 - Oscillation Stop Detection Register
        //-------------------------------------------------------------------------
        CM2_bit.CM27 = 0;        // After Oscillation Stop & re-oscillation detected
                                 // 0 - Oscillation stop detection RESET
                                 // 1 - Oscillation stop detection INTERRUPT

        // PLC0 - PLL Control Register 0
        //-------------------------------------------------------------------------
        PLC0_bit.PLC07 = 0;      // 0 - PLL OFF
                                 // 1 - PLL ON

        PLC0_bit.PLC02 = 0;      // PLL Multiplying Factor
        PLC0_bit.PLC01 = 1;      //      PLC02   PLC01    PLC00
        PLC0_bit.PLC00 = 0;      //      0       0        0       - DO NOT SET
                                 //      0       0        1       - Multiplier x 2
                                 //      0       1        0       - Multiplier x 4
                                 //      0       1        1       - Multiplier x 6
                                 //      1       0        0       - Multiplier x 8
                                 //      1       0        1       - DO NOT SET
                                 //      1       1        0       - DO NOT SET
                                 //      1       1        1       - DO NOT SET

        // PM2 - Processor Mode Register 2
        //-------------------------------------------------------------------------
        PM2_bit.PM22 = 0;        // 0 - WDT Clock Source is CPU clock
                                 // 1 - WDT Clock Source is RING Oscillator

        PM2_bit.PM21 = 0;        // 0 - Clock settings are protected by PRCR register
                                 // 1 - Clock modification is disabled
                                 // NOTE:       Once this bit is set to '1', it can not be cleared to
'0' by software.
                                 // Once this bit is set to '1', writing to the following has no
effect
                                 // CM02 bit of CM0 register
                                 // CM05 bit of CM0 register (main clock does not stop)
```

```
                                // CM07 bit of CM0 register (clock source for the CPU clock does not
change)
                                // CM10 bit of CM1 register (stop mode is not entered)
                                // CM11 bit of CM1 register (clock source for the CPU clock does not
change)
                                //  CM20  bit  of  CM2  register  (oscillation  stop,  re-oscillation
detection function settings do not change)
                                // All bits of PLC0 register (PLL frequency synthesizer settings do
not change)

        PM2_bit.PM20 = 0;       // 0 - '2' waits required when accessing PLL SFR
                                // 1 - '1' waits required when accessing PLL SFR

        // Turn PLL ON
        PLC0_bit.PLC07 = 1;     // 0 - PLL OFF
                                // 1 - PLL ON

        // Wait until the PLL clock becomes stable (tsu(PLL))
        __no_operation();

        // Set the PLL clock as the CPU clock source
        CM1_bit.CM11 = 1;       // 0 - System Clock is Main Clock
                                // 1 - System Clock is PLL Clock

        return (1);
#endif
}
```

If the definition ‘KD30’ has not been made this code sets up the clocks, CPU mode, external bus access etc. If the definition is present then ROM monitor performs this intialisation.

The function main( ) performs function calls that enable and initialise the 3 peripherals that are required in this application. These are the:

- ADC
- Timer A
- INT0

Main:

```
void main( void )
{
        unsigned short us;
        unsigned short *ptrs;
        unsigned char Dir = 0;
        unsigned long *ptr;
        pt2FunctionErase fp;                    // function pointer for calling RAM based function
        unsigned char Status;
        unsigned short AD0_Val;

        InitAdc();
        StartAdc();
        Init_Int0();
        InitDelayTimer();

        // initialise the LED IO port
        PD2 = 0xFF;
        P2 = 0xFF;

        // copy all RAM based code from Flash to RAM
        memcpy( (void *) flashcode.start_address, &flashcode.data[0], flashcode.data_length );
```

```
        __enable_interrupt();

        // wait until INT0 has been activated once to prevent a very large delay occuring
        // from reading 0xffff from the flash
        while( !UpdateFlashFlag );

        while( 1 )
        {
                if( UpdateFlashFlag )
                {
                        UpdateFlashFlag = 0;

                        // erase flash block 4
                        // load pointer to erase block function from function pointer table
                        ptr = (unsigned long *) ( ProgEraseArray + ( ERASEBLOCK_OFFSET * 4 ) );
                        fp = (pt2FunctionErase) *ptr;

                        // call the erase function with the block to be erased and a dummy value (0)
                        Status = fp ( 4, 0 );
                        if ( Status == ERASE_PASS )
                        {
                                  // load the address of the programming function from the function
pointer table
                                  ptr = (unsigned long *) ( ProgEraseArray + ( PROGRAM_OFFSET_WORD *
4 ) );
                                  fp = (pt2FunctionErase) *ptr;

                                  AD0_Val = AD0;

                                  // program the flash with the ADC value
                                  Status = fp( 0xf8000, &AD0_Val );
                        }
                }

                // read the ADC value from the flash location
                // pause for ADC value number of ms
                ptrs = (unsigned short *)0xf8000;
                us = *ptrs;
                while( us )
                {
                        StartDelayTimer();
                        // wait for the 1ms delay to expire
                        while( !GetDelayTimerStatus() );
                        StopDelayTimer();
                        us--;
                }

                if( Dir == 0 )
                {
                        P2 = P2 << 1;
                        if( P2 == 0x80 )
                        {
                                  Dir = 1;
                        }
                }
                else
                {
                        P2 = P2 >> 1;
                        if( P2 == 0x01 )
                        {
                                  Dir = 0;
                        }
                }
        }
}
```

Once the 3 peripherals are enabled and initialised the application sits in a while(1) loop. Here the LED chaser sequence is output to the LED port with the delay between updates being read from the flash memory location 0xF8000. As described previously this delay value is originated from the ADC converter. This value is updated in the flash whenever the INT0 interrupt is activated by pressing the INT0 switch.

Global declarations:

```
// external reference to the
// structure contained in 'flash_converted.c' which contains
// the code which is loaded into RAM and executed from RAM
extern const struct rom_data flashcode;

// storage space for the RAM based functions
#pragma dataseg=RESERVED
__no_init unsigned char ProgEraseArray[1000];
#pragma dataseg=default

static unsigned char UpdateFlashFlag = 0;

typedef unsigned char (*pt2FunctionErase)(unsigned long, unsigned short * );
```

INT0 Code:

```
#pragma vector = 29
__interrupt void int0(void)
{
        // INT0 ISR

        UpdateFlashFlag = 1;
}
```

ADC Code:

```
void InitAdc( void )
{
        // pins used as ADC inputs the port direction bit for the pin must be input (0)
        // AN0 is on P10.0
        PD10_bit.PD10_0 = 0;      //      0       -        input
                                  //      1       -        output

        // ADCON0
        // the CH2-0 bits must be written after any changes to MD1-0 bits
                                  //      MD1     MD0
        ADCON0_bit.MD1 = 0;       //      0       0        -        one shot mode
        ADCON0_bit.MD0 = 1;       //      0       1        -        repeat mode
                                  //      1       0        -        single sweep mode
                                  //      1       1        -        repeat sweep mode

                                  // CH2  CH1     CH0
        ADCON0_bit.CH2 = 0;       //      0       0        0        - AN0 selected
        ADCON0_bit.CH1 = 0;       //      0       0        1        - AN1 selected
        ADCON0_bit.CH0 = 0;       //      0       1        0        - AN2 selected
                                  //      0       1        1        - AN3 selected
                                  //      1       0        0        - AN4 selected
                                  //      1       0        1        - AN5 selected
                                  //      1       1        0        - AN6 selected
                                  //      1       1        1        - AN7 selected

        ADCON0_bit.TRG = 0;       //      0       -        software trigger
```

```
                                //      1       -       /ADTRG

        ADCON0_bit.ADST = 0;    //      0       -       conversion disabled
                                //      1       -       conversion started


        // ADC clock must be <10MHz
        // fAD ~ system clock (24MHz)
                                //      CKS2 CKS1 CKS0
        ADCON2_bit.CKS2 = 0;    //      0    0    0             fAD / 4
        ADCON1_bit.CKS1 = 0;    //      0    0    1             fAD / 2
        ADCON0_bit.CKS0 = 0;    //      0    1    0             fAD
                                //      0    1    1             fAD
                                //      1    0    0             fAD / 12
                                //      1    0    1             fAD / 6
                                //      1    1    0             fAD / 3
                                //      1    1    1             fAD / 3

        // ADCON1
                                // SCAN1 SCAN0
        ADCON1_bit.SCAN1 = 0;   // 0     0             - invalid in repeat mode
        ADCON1_bit.SCAN0 = 0;   // 0     1             - invalid in repeat mode
                                // 1     0             - invalid in repeat mode
                                // 1     1             - invalid in repeat mode

        ADCON1_bit.MD2 = 0;     //      0              - any mode other than repeat sweep
                                //      1              - repeat sweep mode

        ADCON1_bit.BITS = 1;    //      0              -       8-bit mode
                                //      1              -       10-bit mode

        ADCON1_bit.VCUT = 1;    //      0              -       Vref uncocnnected
                                //      1              -       Vref connected

                                //      OPA1 OPA0      -       external op-amp connection
        ADCON1_bit.OPA1 = 0;    //      0      0       -       ANEX1 and ANEX0 not used
        ADCON1_bit.OPA0 = 0;    //      0      1       -       ANEX0 input is converted
                                //      1      0       -       ANEX1 input is converted
                                //      1      1       -       external op-amp connection mode

        // ADCON2
        ADCON2_bit.SMP = 1;     //      0              -       without sample and hold
                                //      1              -       with sample and hold

                                //      ADGSEL1 ADGSEL0
        ADCON2_bit.ADGSEL1 = 0; //      0       0              P10 group selected
        ADCON2_bit.ADGSEL0 = 0; //      0       1              must not be set
                                //      1       0              P0 group selected
                                //      1       1              P2 group selected
}

void StartAdc( void )
{
        ADCON0_bit.ADST = 1;    //      0              -       conversion disabled
                                //      1              -       conversion started
}

void StopAdc( void )
{
        ADCON0_bit.ADST = 0;    //      0              -       conversion disabled
                                //      1              -       conversion started
}
```

Flash Project

The code for FLASH erasing and programming is developed as a separate project. The code and linker settings are shown over the next couple of pages.

Flash.c

```
// flash.c

#include <intrinsics.h>
#include "flash_header.h"
#include "..\FlashTraining\iom16c62p.h"

//
//      __root directive required to ensure that the function and variable/constant
//      data is included in the object code even if the linker thinks it is not used
//

typedef unsigned char (*pt2Function)( unsigned long, unsigned short * );

#pragma location="CPROGERA"
__root const pt2Function fp[] = {
        BlockErase,
        Program_128_Bytes,
        Program_Word
};

__root const unsigned long Block_Address[][2] = {
        { 0xFF000, 0xFFFFE },
        { 0xFE000, 0xFEFFE },
        { 0xFC000, 0xFDFFE },
        { 0xFA000, 0xFBFFE },
        { 0xF8000, 0xF9FFE },
        { 0xF0000, 0xF7FFE },
        { 0xE0000, 0xEFFFE },
        { 0xD0000, 0xDFFFE },
        { 0xC0000, 0xCFFFE },
        { 0xB0000, 0xBFFFE },
        { 0xA0000, 0xAFFFE },
        { 0x0F000, 0x0FFFE }
};

__root void EnterEW0Mode( void )
{
        // during flash memory programming, erasing etc
        // the CPU clock frequency should be < 10MHz (with 1 wait state)
        // or < 6.5MHz (no wait state)
        // 3D board has a 6MHz xtal which is internal multiplied by 4 and divided by 0
        // therefore this clock should be divided by 4 during flash memory operations

        __disable_interrupt();

        // enable access to the clock mode register 1 (CM0)
        PRCR_bit.PRC0 = 1;

        // CM1 - System Clock Control Register 1
        //-------------------------------------------------------------------------
        CM1_bit.CM17 = 1;        // Main Clock Divide Ratio
        CM1_bit.CM16 = 0;        //      CM17    CM16
                                 //      0       0               - No Division
                                 //      0       1               - Divide by 2
                                 //      1       0               - Divide by 4
                                 //      1       1               - Divide by 16

        // disable access to the clock mode register 1 (CM0)
```

```
        PRCR_bit.PRC0 = 0;

        // CPU now operating at 6MHz
        //
        // enter CPU re-write mode
        FMR0_bit.FMR01 = 0;             // first write 0 then write 1
        FMR0_bit.FMR01 = 1;

        // set EW0 mode
        FMR1_bit.FMR11 = 0;

        // disable all lock bit protection to allow programming and erase operations
        FMR0_bit.FMR02 = 0;             // first write 0 then write 1
        FMR0_bit.FMR02 = 1;

        // all flash blocks now unlocked
        //
        // wait while flash is busy
        while( FMR0_bit.FMR00 == 0 );
}

__root void ExitEW0Mode( void )
{
        // when this function exits the CPU clock will be 24MHz

        // enable the lock-bit protection
        FMR0_bit.FMR02 = 0;

        // exit CPU re-write mode
        FMR0_bit.FMR01 = 0;

        // enable access to the clock mode register 1 (CM0)
        PRCR_bit.PRC0 = 1;

        // CM1 - System Clock Control Register 1
        //-------------------------------------------------------------------------
        CM1_bit.CM17 = 0;        // Main Clock Divide Ratio
        CM1_bit.CM16 = 0;        // CM17 CM16
                                 // 0          0               - No Division
                                 // 0          1               - Divide by 2
                                 // 1          0               - Divide by 4
                                 // 1          1               - Divide by 16

        // disable access to the clock mode register 1 (CM0)
        PRCR_bit.PRC0 = 0;

        __enable_interrupt();
}

__root unsigned char BlockErase( unsigned long BlockNum, unsigned short *Dummy )
{
        // attempts to erase the block specified by BlockNum
        //
        // returns:
        //
        // ERASE_PASS
        // or
        // ERASE_FAIL

        volatile unsigned short __data20 *p;
        volatile unsigned char Status;
        unsigned long ul;
        volatile unsigned short Tmp;

        // check if block needs erasing
        Status = 0;
        p = (volatile unsigned short __data20 *)Block_Address[ BlockNum ][ 0 ];
        for( ul = Block_Address[ BlockNum ][ 0 ]; ul<Block_Address[ BlockNum ][ 1 ] + 1; ul++ )
```

```
        {
                if( *p != 0xffff )
                {
                        Status = 1;
                        break;
                }
                else
                {
                        p++;
                }
        }

        if( Status == 0 ) return ERASE_PASS;

        EnterEW0Mode();

        // write erase block command to even flash address
        p = (volatile unsigned short __data20 *)Block_Address[ BlockNum ][ 0 ];
        *p = BLOCK_ERASE;
        // write H'xxD0 to upper most address in block to be erased
        p = (volatile unsigned short __data20 *)Block_Address[ BlockNum ][ 1 ];
        *p = 0x00D0;

        // wait while flash is busy
        while( FMR0_bit.FMR00 == 0 );

        // read flash to allow access to the status register
        Tmp = *p;

        // read the erase status flag in the status register
        Status = FMR0_bit.FMR07;

        ExitEW0Mode();

        if( Status == 0 )
        {
                // flash block erase successful
                return ERASE_PASS;
        }
        else
        {
                // flash block erase failed
                return ERASE_FAIL;
        }
}

__root unsigned char Program_128_Bytes( unsigned long Address, unsigned short *Data )
{
        // attempts to program 128 bytes (in word units)
        // at address the specified address
        //
        // returns:
        //
        // PROG_PASS
        // or
        // PROG_FAIL

        unsigned char Count, Status;
        unsigned long Addr;
        volatile unsigned short __data20 *p;
        volatile unsigned short Tmp;

        Count = 0;
        Addr = Address;

        EnterEW0Mode();

        Status = PROG_PASS;
```

```
        while( ( Count < (128 / 2) ) && ( Status == PROG_PASS ) )
        {
                p = (volatile unsigned short __data20 *)Addr;
                *p = PROGRAM_CMD;
                *p = Data[ Count ];

                // wait while flash is busy
                while( FMR0_bit.FMR00 == 0 );

                // read flash to allow access to the status register
                Tmp = *p;

                // read the erase status flag in the status register
                Status = FMR0_bit.FMR06;

                if( Status == 0 )
                {
                        Status = PROG_PASS;
                }
                else
                {
                        Status = PROG_FAIL;
                }

                Count++;
                Addr += 2;
        }

        ExitEW0Mode();

        return Status;
}

__root unsigned char Program_Word( unsigned long Address, unsigned short *Data )
{
        // attempts to program 128 bytes (in word units)
        // at address the specified address
        //
        // returns:
        //
        // PROG_PASS
        // or
        // PROG_FAIL

        unsigned char Status;
        volatile unsigned short __data20 *p;
        volatile unsigned short Tmp;

        EnterEW0Mode();

        p = (volatile unsigned short __data20 *)Address;
        *p = PROGRAM_CMD;
        *p = Data[ 0 ];

        // wait while flash is busy
        while( FMR0_bit.FMR00 == 0 );

        // read flash to allow access to the status register
        Tmp = *p;

        // read the erase status flag in the status register
        Status = FMR0_bit.FMR06;

        if( Status == 0 )
        {
                Status = PROG_PASS;
        }
        else
```

```
        {
                Status = PROG_FAIL;
        }

        ExitEW0Mode();

        return Status;
}
```

Flash Linker Command File

```
//====================================================================
//  IAR XLINK command file for the M16C/R8C IAR C/EC++ Compiler
//
//  This is an example XLINK command file for use with the
//  IAR C libraries and M16C derivatives.
//
//  Copyright 1995 - 2003 IAR Systems. All rights reserved.
//
//  Usage:  xlink  your_file(s)  -f this_file  clm16cxxx
//
//  $Revision: 1.6 $
//====================================================================

//====================================================================
// The M16C IAR C/EC++ Compiler places code and data into named
// segments which are referred to by the IAR XLINK Linker. The
// table below shows the available segments.
//
// SEGMENT REFERENCE
// =================
//
// Segment   Description
// -------   -----------
// BITVARS   Bit variables.
// CODE      The program code.
// CSTACK    The stack used by C or Embedded C++ programs.
// CSTART    The startup code.
//
// x_AC      Non-initialized located const objects.
// x_AN      Non-initialized located non-const objects.
// x_C       Constant data, including string literals.
// x_I       Initialized data.
// x_ID      Data that is copied to x_I by cstartup.
// x_N       Uninitialized data.
// x_Z       zero initialized data.
//
// Where x can be one of:
//    DATA13  (Range: 0-0x1FFF)
//    DATA16  (Range: 0-0xFFFF)
//    DATA20  (Range: 0-0xFFFFF)
//    FAR     (Range: 0-0xFFFFF)
//
// DIFUNCT   Pointers to code, typically EC++ constructors
// FLIST     Jump table for __tiny_func functions.
// HEAP      The heap data used by malloc and free.
// INTVEC    Contains reset and interrupt vectors.
// INTVEC1   Contains the fixed reset and interrupt vectors.
// ISTACK    The stack used by interrupts and exceptions.
//====================================================================

// Define CPU
-cm16c

//====================================================================
```

```
// USER DEFINITIONS
// Please customize according to your specific derivative!
//===================================================================

// User defined far & huge CONST segments
-Z(FARCONST)CPROGERA=0x3000
-Z(FARCONST)FAR_C
-Z(FARCODE)CODE


// ========================
// IAR C library formatting
// ========================
-e_small_write=_formatted_write
-e_medium_read=_formatted_read

// ========================
// Output files
// ========================
// Use the -O option to create one or more output files
// at the same link session. Formats flags, file name and
// extension is optional. Please un-comment the wanted
// output formats below.
//
// CAUTION: Do not combine ohter output formats with -rt (special
// UBROF for Terminal I/O in C-SPY). Output files are valid but
// contain code that expects to be run under C-SPY.

// Motorola output
-Omotorola=.mot

// IEEE-695 output with format flags for the Renesas debugger
//-Oieee695,lbm=.x30

// ELF/DWARF output with format flags for the Renesas debugger
//-Oelf,spc=.elf
```

The s-record file generated by the linker is converted into a C structure using the 'Motice_cl' utility as described in apps note REG05B0021-0100. The command line used to perform the conversion is shown below.

```
flash.mot flash_converted.c flashcode
```

For convenience the 'Motice_cl' tool and command line can be added to the IAR Workbench 'Tools' menu. The conversion can then be initiated directly from the menu. The screenshot shown in figure 5 gives an example of how this menu option can be configured.

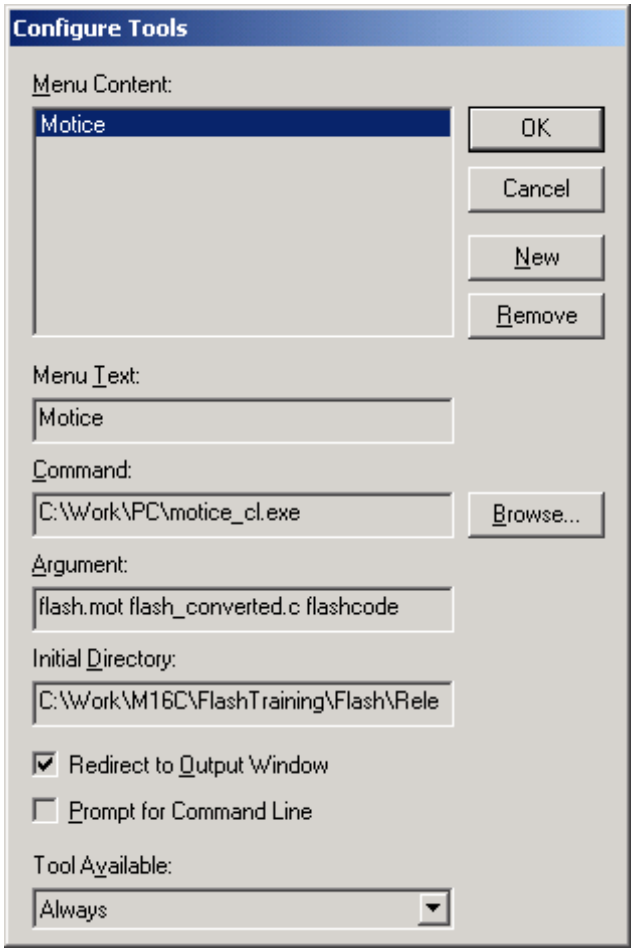

Figure 5: Configuring IAR Workbench Tools Menu Command

Summary

It has been the aim of this application note to demonstrate via a simple example how it is possible to implement User Mode Flash Programming on a Renesas M16C microcontroller.

Even though the example was written for the M16C/62P, all of the code and concepts can be easily ported to other members of the M16C family.

Accompanying this application note there are 2 file downloads, each containing an IAR Workbench project.

Website and Support

Renesas Technology Website
http://www.renesas.com/

Inquiries
http://www.renesas.com/inquiry
csc@renesas.com

Notes regarding these materials

1. This document is provided for reference purposes only so that Renesas customers may select the appropriate Renesas products for their use. Renesas neither makes warranties or representations with respect to the accuracy or completeness of the information contained in this document nor grants any license to any intellectual property rights or any other rights of Renesas or any third party with respect to the information in this document.
2. Renesas shall have no liability for damages or infringement of any intellectual property or other rights arising out of the use of any information in this document, including, but not limited to, product data, diagrams, charts, programs, algorithms, and application circuit examples.
3. You should not use the products or the technology described in this document for the purpose of military applications such as the development of weapons of mass destruction or for the purpose of any other military use. When exporting the products or technology described herein, you should follow the applicable export control laws and regulations, and procedures required by such laws and regulations.
4. All information included in this document such as product data, diagrams, charts, programs, algorithms, and application circuit examples, is current as of the date this document is issued. Such information, however, is subject to change without any prior notice. Before purchasing or using any Renesas products listed in this document, please confirm the latest product information with a Renesas sales office. Also, please pay regular and careful attention to additional and different information to be disclosed by Renesas such as that disclosed through our website. (http://www.renesas.com)
5. Renesas has used reasonable care in compiling the information included in this document, but Renesas assumes no liability whatsoever for any damages incurred as a result of errors or omissions in the information included in this document.
6. When using or otherwise relying on the information in this document, you should evaluate the information in light of the total system before deciding about the applicability of such information to the intended application. Renesas makes no representations, warranties or guaranties regarding the suitability of its products for any particular application and specifically disclaims any liability arising out of the application and use of the information in this document or Renesas products.
7. With the exception of products specified by Renesas as suitable for automobile applications, Renesas products are not designed, manufactured or tested for applications or otherwise in systems the failure or malfunction of which may cause a direct threat to human life or create a risk of human injury or which require especially high quality and reliability such as safety systems, or equipment or systems for transportation and traffic, healthcare, combustion control, aerospace and aeronautics, nuclear power, or undersea communication transmission. If you are considering the use of our products for such purposes, please contact a Renesas sales office beforehand. Renesas shall have no liability for damages arising out of the uses set forth above.
8. Notwithstanding the preceding paragraph, you should not use Renesas products for the purposes listed below:
   (1) artificial life support devices or systems
   (2) surgical implantations
   (3) healthcare intervention (e.g., excision, administration of medication, etc.)
   (4) any other purposes that pose a direct threat to human life

   Renesas shall have no liability for damages arising out of the uses set forth in the above and purchasers who elect to use Renesas products in any of the foregoing applications shall indemnify and hold harmless Renesas Technology Corp., its affiliated companies and their officers, directors, and employees against any and all damages arising out of such applications.
9. You should use the products described herein within the range specified by Renesas, especially with respect to the maximum rating, operating supply voltage range, movement power voltage range, heat radiation characteristics, installation and other product characteristics. Renesas shall have no liability for malfunctions or damages arising out of the use of Renesas products beyond such specified ranges.
10. Although Renesas endeavors to improve the quality and reliability of its products, IC products have specific characteristics such as the occurrence of failure at a certain rate and malfunctions under certain use conditions. Please be sure to implement safety measures to guard against the possibility of physical injury, and injury or damage caused by fire in the event of the failure of a Renesas product, such as safety design for hardware and software including but not limited to redundancy, fire control and malfunction prevention, appropriate treatment for aging degradation or any other applicable measures. Among others, since the evaluation of microcomputer software alone is very difficult, please evaluate the safety of the final products or system manufactured by you.
11. In case Renesas products listed in this document are detached from the products to which the Renesas products are attached or affixed, the risk of accident such as swallowing by infants and small children is very high. You should implement safety measures so that Renesas products may not be easily detached from your products. Renesas shall have no liability for damages arising out of such detachment.
12. This document may not be reproduced or duplicated, in any form, in whole or in part, without prior written approval from Renesas.
13. Please contact a Renesas sales office if you have any questions regarding the information contained in this document, Renesas semiconductor products, or if you have any other inquiries.