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/62

Programming the M16C/62 Flash in CPU Rewrite Mode

1.0 Abstract

The following article describes CPU Rewrite Mode on the M16C/62 (M30624FG), which allows erasing and programming the on-chip flash memory under control of a user's program. It is assumed that the microcontroller is operating in "single chip" mode. A short program, targeted for the MSV1632 Starter Kit, illustrates how to apply CPU Rewrite Mode.

2.0 Introduction

The Renesas M16C/62 is a 16-bit MCU, based on the M16C CPU core, with 256k bytes of user flash. The device can erase and program the on-chip flash memory under control of a user's program with no external programming devices required. This feature is called "CPU Rewrite Mode".

The premise for CPU Rewrite Mode is that applications may require nonvolatile storage of data. In general, this could be data acquisition, configuration parameters, hours in service, and so on.

3.0 Background

The M16C/62 has two other flash programming modes: Parallel I/O Mode, and Standard Serial I/O Mode. Because these modes are mainly for programming the application code into the flash, details are not discussed in this article.

To use CPU Rewrite Mode, the memory structure and the control registers need to be identified. The memory map of the M16C/62 is shown in Figure 1. Note that the flash is divided into blocks such that certain erase/programming functions are done on a block basis. The boot flash area is used for serial I/O mode and is not available for CPU Rewrite mode programming.

The "Flash Memory Control Register" (FMR0) is shown in Figure 2. Normally, only the first three LSBs are used for CPU rewrite mode.

Beyond CPU registers, the flash memory has its own logic to handle erase and programming procedures. This is the flash's "Write State Machine" (WSM). The WSM commands are given in Table 1.

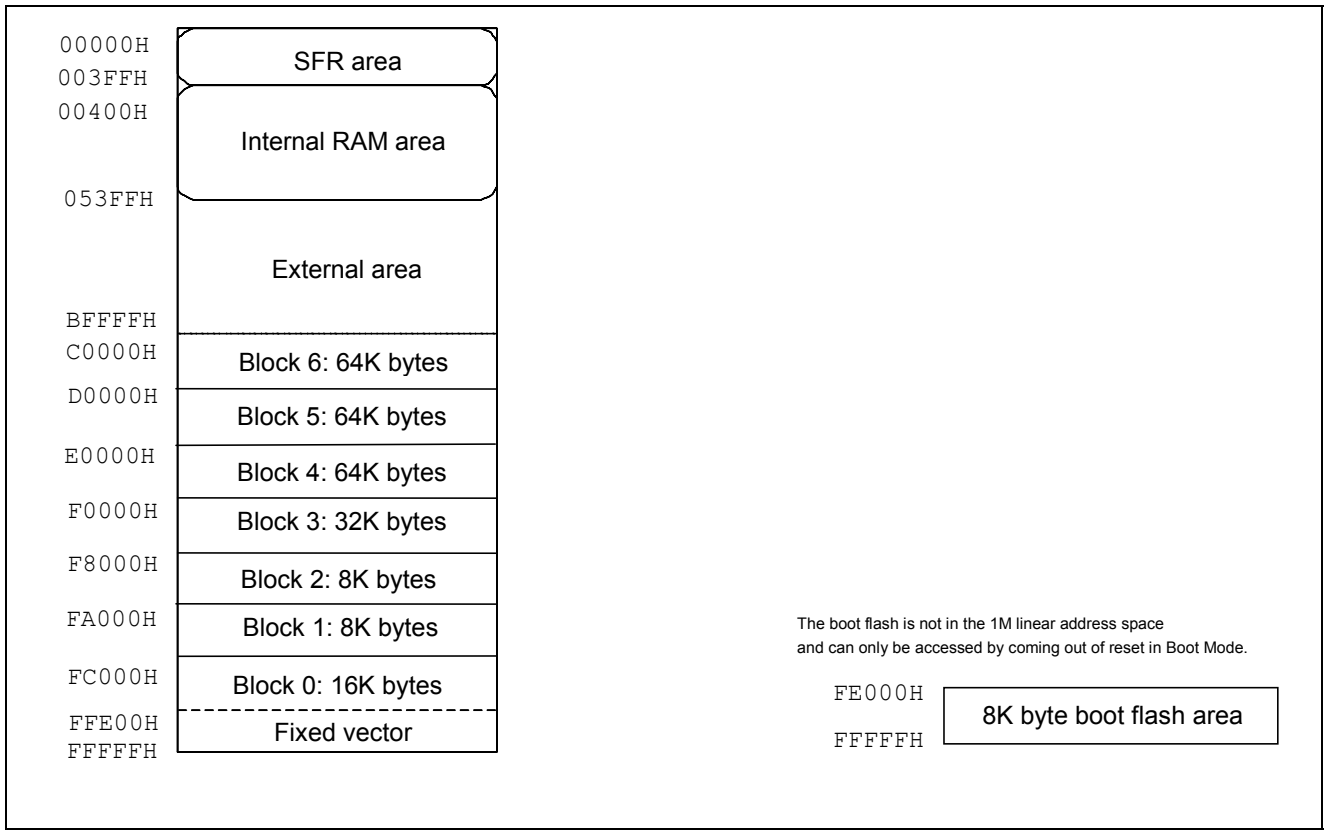

Figure 1 M16C/62 Memory Map

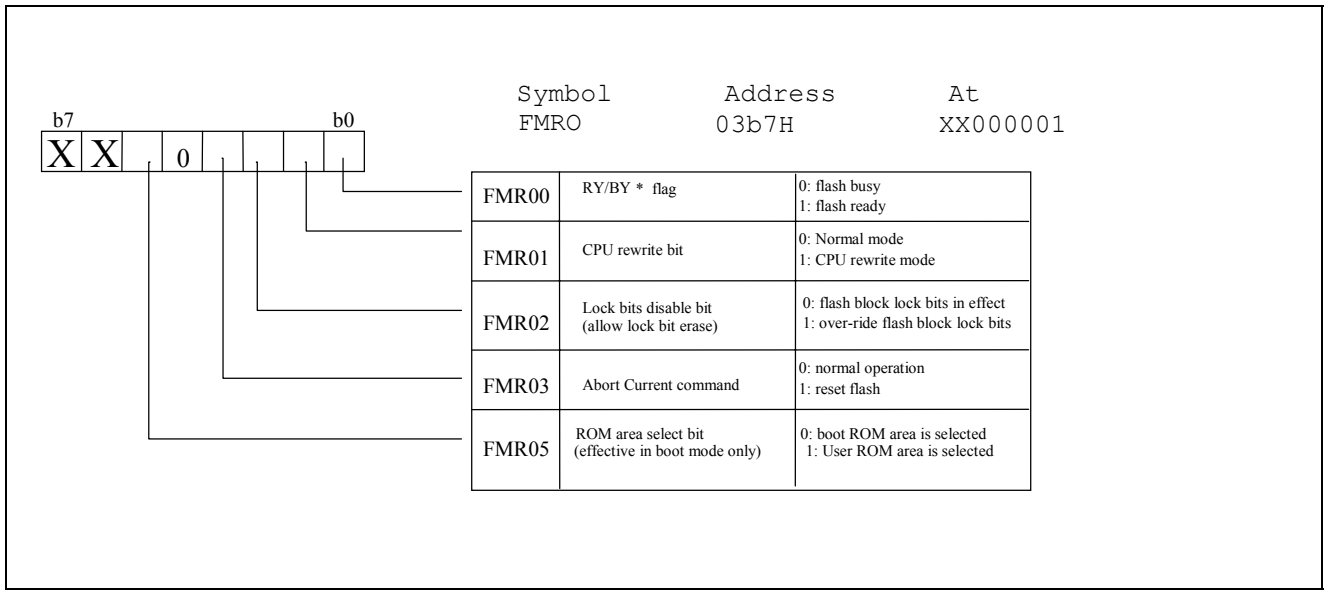

Figure 2 Flash Memory Control Register

Table 1 List of Software Commands (CPU Rewrite Mode)

Command	First bus cycle			Second bus cycle			Third bus cycle		
	Mode	Address	Data (D_0 to D_7)	Mode	Address	Data (D_0 to D_7)	Mode	Address	Data (D_0 to D_7)
Read array	Write	X (Note 6)	FF_{16}						
Read status register	Write	X	70_{16}	Read	X	SRD (Note 2)			
Clear status register	Write	X	50_{16}						
Page program (Note 3)	Write	X	41_{16}	Write	WA0 (Note 3)	WD0 (Note 3)	Write	WA1	WD1
Block erase	Write	X	20_{16}	Write	BA (Note 4)	$D0_{16}$			
Erase all unlock block	Write	X	$A7_{16}$	Write	X	$D0_{16}$			
Lock bit program	Write	X	77_{16}	Write	BA	$D0_{16}$			
Read lock bit status	Write	X	71_{16}	Read	BA	D_6 (Note 5)			

Note 1: When a software command is input, the high-order byte of data (D_8 to D_{15}) is ignored.
Note 2: SRD = Status Register Data
Note 3: WA = Write Address, WD = Write Data
WA and WD must be set sequentially from 00_{16} to FE_{16} (byte address; however, an even address). The page size is 256 bytes.
Note 4: BA = Block Address (Enter the maximum address of each block that is an even address.)
Note 5: D_6 corresponds to the block lock status. Block not locked when D_6 = 1, block locked when D_6 = 0.
Note 6: X denotes a given address in the user ROM area (that is an even address).

4.0 Flash Programming Basics

The flash must be programmed in 256-byte pages on page boundaries (A0 –A7 = 00 – FEh), 16 bits at a time. Attempts to program 8-bit data are ignored, and even commands must be set as 16-bit words. The flash can be “bulk” erased (‘erase all unlocked blocks’ command) or erased one block at a time (see memory map). Bit erase state = 1. Once a block is erased, individual pages can be programmed at any time. The CPU rewrite program can be stored in the flash, but because the WSM is common to all flash (all blocks), the CPU rewrite code cannot execute out of flash. The rewrite code must be transferred to RAM before it can be executed.

A generalized CPU Rewrite mode flowchart is shown in Figure 3. Note that before and after executing any program or erase command, the “Read Array” command is issued. This has the effect of resetting the flash control logic.

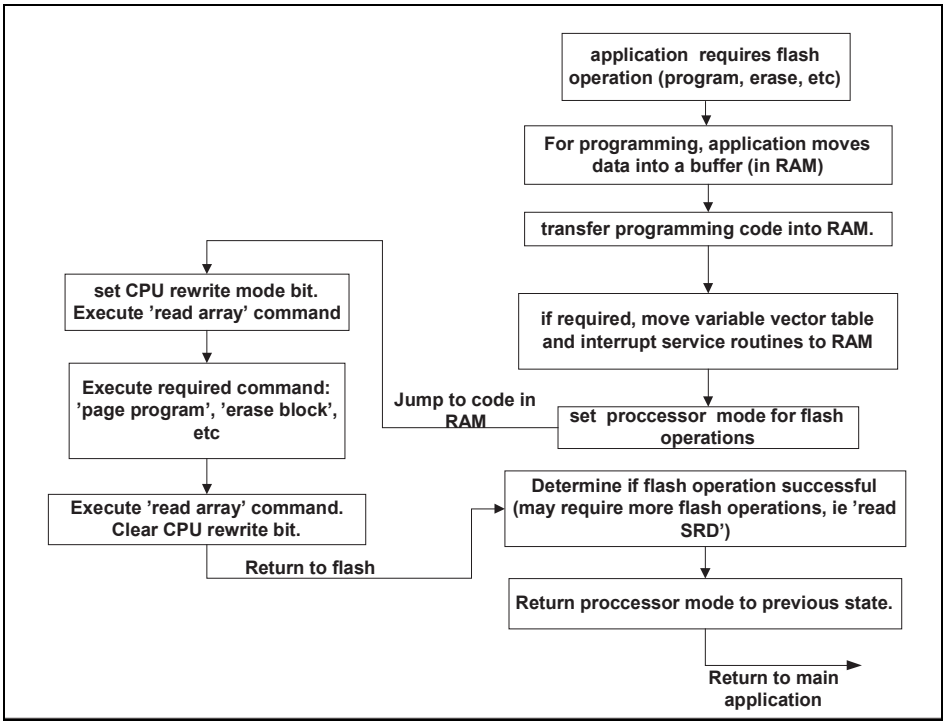

Figure 3 CPU Rewrite Mode Flowchart

5.0 Example Program

The example program was written to run on the MSV1632 Starter Kit but could be modified to implement in a user application. The program is written in C (Renesas' NC30 Compiler), with assembler used for the code executing out of RAM. This is a "no frills" program, but its main feature is that the code in RAM uses only about 160 bytes.

As illustrated in Figure 4, the program erases a block, programs a page, locks a block, and then unlocks the block. The block is unlocked because locked blocks are not compatible with the Starter Kit's debugger, KD30.

Note in the assembler code that after the CPU Rewrite bit is cleared, the Reserve bit is set. This is required because clearing the CPU Rewrite bit automatically clears the Reserve bit. The Reserve bit is used for compatibility between other M16C derivatives. For consistent operation of the M16C/62 in single chip mode, this bit is set before returning to the flash area. See the M16C/62 data sheets for more information.

5.1 Compatibility

This program is compatible with M16C/6x microcontrollers with page write (256 bytes) flash memory. It is NOT compatible with word write MCUs such as the M16C/62P series. The driver is compatible with the above noted MCUs on any Starter Kit/evaluation system running under the KD30 debugger. It CANNOT be evaluated or demonstrated on any emulator using RAM to emulate flash (i.e., Renesas' PC4701, Nohau, or Ashling emulator systems).

5.2 Demonstrating the Program

Under KD30, the RAM buffer (at 2000h) can be edited, the program run, and the flash viewed. Note that viewing the C Watch -> Globals window will produce the status of the flash (SRD) and the lock bit status.

6.0 Alternate Implementation

To program the flash, 256 bytes must be sent to the flash at a time. However, not all 256 bytes need to be programmed at the same time. If only part of a page is to be programmed, the remaining bytes are sent FFh (erase state). At a later time, more bytes can be programmed using the following steps:

1. Copy the partly programmed flash page to a 256-byte RAM buffer.
2. Insert the bytes to be programmed into the RAM buffer.
3. Write the RAM buffer to flash.

It must be noted that each time a page is programmed, it should be considered as another erase/program cycle.

7.0 Reference

Renesas Technology Corporation Semiconductor Home Page

http://www.renesas.com

E-mail Support

support_apl@renesas.com

Data Sheets

M16C/62 datasheets, 62aeds.pdf

User's Manual

- 6020esm.pdf (Software Manual)
- 6020ec.pdf (C Manual)
- 6020easm.pdf (Assembler Manual)
- NC30ue.pdf (Compiler Manual)

8.0 Appendix

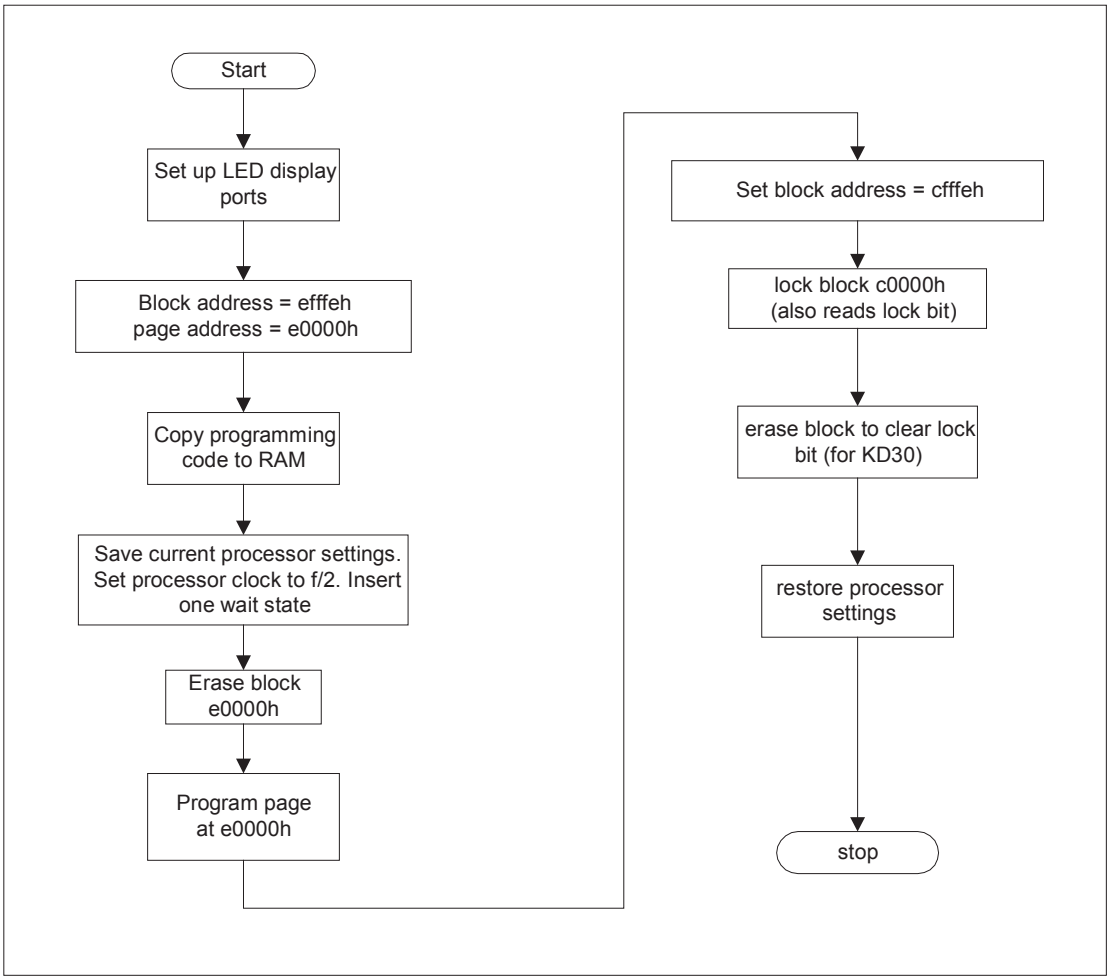

Figure 4 Example Program Flowchart: Main C Routine

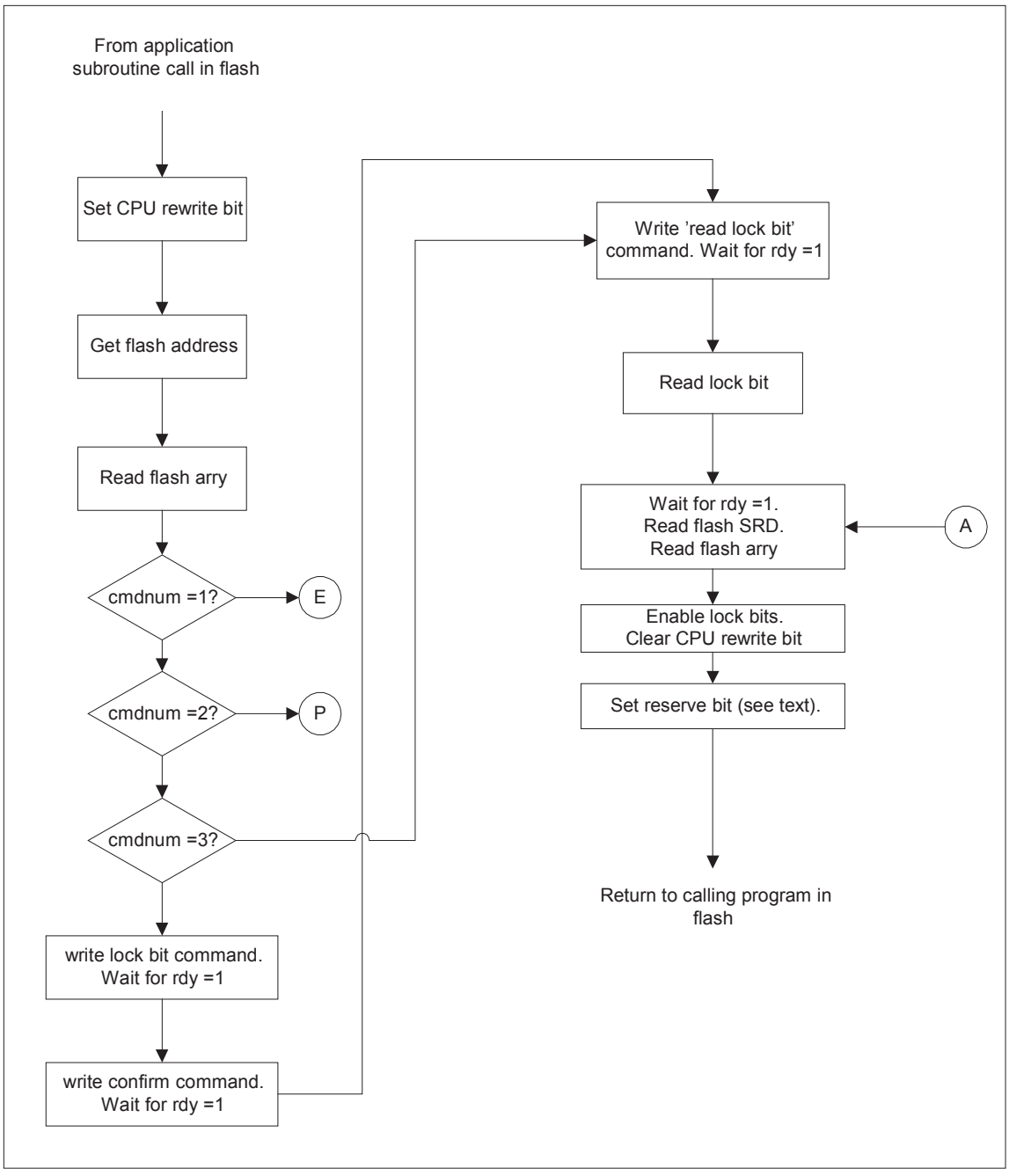

Figure 5 Example Program Flowchart: Assembler Code in RAM. Page 1

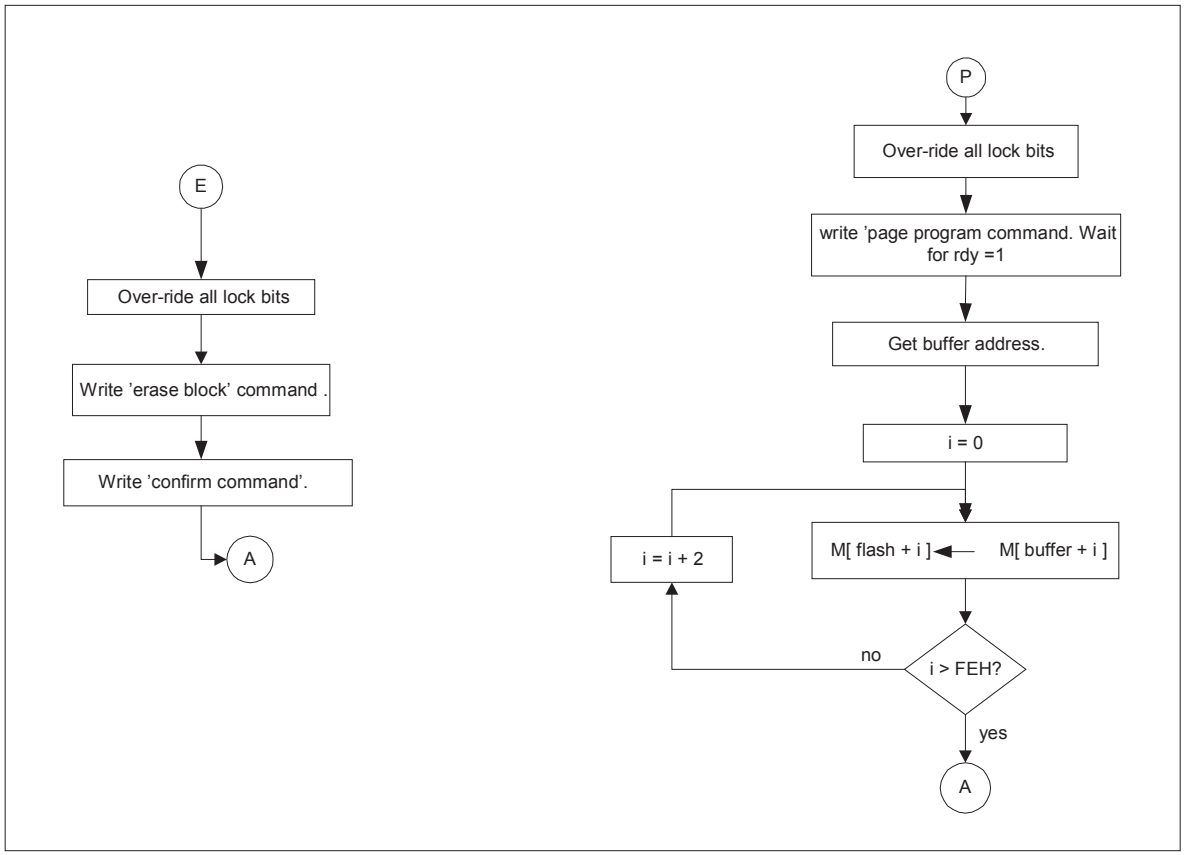

Figure 6 Example Program Flowchart: Assembler Code in RAM. Page 2

9.0 Software Code

```
/***********************************************************************************
*
*
*       File Name: cpu_rw62.c
*
*       Content: CPU REWRITE PROGRAMMER version 2.1
*        Compiled with KNC30 ver. 3.20.00.
*
*        Demonstrates CPU rewrite mode on the M16C/62
*        microcontroller. This program was designed to
*        operate on any M16C/62 starter kit, using the KD30 debugger.
*        Note it is NOT compatible with emulators such as
*        Renesas's PC4701 series that use RAM to emulate flash ROM.
*
*        It is intended that the program executes out of the D0000h block
*        as blocks C0000h and E0000h are erased and programmed.
*
*        The program also demonstrates how to mix assembler
*        and C code using Renesas's NC30 compiler.
*
*
```

```
*   Copyright,2003 RENESAS TECHNOLOGY CORPORATION
*
*
*  note:
*      Renesas Technology Corporation does not guarantee the performance or
*      use of this source-code.  The intended use of provided source-code is the
*      sole responsible of the user.  The files have been successfully compiled
*      using Renesas's NC30 compiler.  Before using this software review the
*      source and make any necessary changes to support your hardware and
*      application.
*
*=============================================================================
*      $Log:$
*============================================================================*/
#define aerase   1      //set cmndnum to one of these 'a' values
#define aprogram 2
#define areadloc 3
#define alocblock 4

#define wait1 0x8e00    // processor mode value for 1 wait state

#define interrupton    _asm ("   fset i")   // "macro" assembler code
#define interruptoff   _asm ("   fclr i")

#define RAMBUFFLOC  0x2000  //location of ram buffer (for programming flash)


#pragma ADDRESS ramcode 1000h           // address in RAM for CPU rewrite code
#pragma ADDRESS prcr 00ah               // SFR's
#pragma ADDRESS cm0 006h
#pragma ADDRESS cm1 007h
#pragma ADDRESS pm01 004h

unsigned int ram_buff;              // assembler "pointer" to RAM buffer
unsigned long int flsh_addr;        // assembler "pointer" to flash page/block
extern far int copy_strt,copy_end;
far volatile int *ramxfer,*romxfer;
volatile int pm01;
int    pm01sav;

char   cm0sav,cm1sav;
char flsh_stat,cmndnum,lokbitstat;
volatile char cm0,cm1,prcr,port0,port1,dir_p0,dir_p1;

// function prototypes
void asm_block(void);
void ramcode(void);            // From the 'pragma' above, C considers "ramcode"
                               //just as any other label in memory.
void transfer(void);
void cpu_ini(void);
void eraseblock(unsigned long int);
void writepage(unsigned long int);
void lockblock(unsigned long int);
void cpu_rstor(void);
void lockbitstat(unsigned long int);
```

```
 /***********************************************************************
Name:    asm_block()

Parameters:
  inputs: globals only: cmndnum - 1 = erase block
                                  2 = program page
                                  3 = read lock bit status (default command)
                                  4 = lock block, includes read lock bit status


                        flsh_addr - the block or page to be operated on
                        ram_buff  - points to the 256 bytes of data to be
                                    programmed
                                    (required only for program page command)

Returns: nothing
  modifies: global flsh_stat  - the flash SRD status byte after performing command
            global lokbitstat - if bit6 (D6) = 1, block unlocked,
                                if bit6 = 0, block locked.


Description: CPU rewrite assembler code. The following 'block' of assembler
             code is moved to RAM then executed out of RAM. The assembler
             code is relocatable and command functions are generic enough
             to be implemented in a user application. Although not required
             for the starter kit, the erase and program routines disable
             the lock block bits.
             Ram usage approx. 160 bytes.
*********************************************************************** */



void asm_block(void)    // clean place for the assembler code, but function
                        // not required.
{

#pragma       ASM                          // 'ASM' must be in upper case


;flash memory commands
rd_fstat             .equ   0070h
wrt_cmd              .equ   0041h
erase_cmd            .equ   0020h
cfm_cmd              .equ   00D0h
lok_cmd              .equ   0077h
rlok_cmd             .equ   0071h
rda_cmd              .equ   00ffh


fmr0                 .equ          03b7h   ;M16C\62 flash control register
bsy_rdy              .btequ        0,fmr0
cpu_rwrt             .btequ        1,fmr0
lock_dis             .btequ        2,fmr0
```

```
flsh_rst            .btequ      3,fmr0
resbit              .equ        3       ; reserve area bit

_copy_strt:
; first do instructions common to all commands (save RAM space)
                    bclr        cpu_rwrt
                    bset        cpu_rwrt            ;set CPU rewrite mode
                    mov.w       _flsh_addr+2,a1
                    mov.w       _flsh_addr,a0       ;get block address
; sequence to reset flash
                    mov.b       #rda_cmd,r0l        ; Read array resets flash
                    jsr         Command_write

;  decode command
                    cmp.b       #aerase,_cmndnum
                    jeq         eraseflsh           ;erase block?
                    cmp.b       #aprogram,_cmndnum
                    jeq         progflsh            ;write page?
                    cmp.b       #alocblock,_cmndnum
                    jeq         lockflshblk         ;lock block?
                    jmp         readlokstat         ;default:read lock bit

lockflshblk:
                    mov.b       #lok_cmd,r0l
                    jsr         Command_write       ;send lock command
                    mov.b       #cfm_cmd,r0l
                    jsr         Command_write       ;send confirm byte
                                                    ;fall into read lock bit
readlokstat:
                    mov.b       #rlok_cmd,r0l
                    jsr         Command_write       ;read lock bit
                    lde.w       [a1a0],r0
                    mov.b       r0l,_lokbitstat

                    jmp         flsh_bsy      ;lock/read  command sent,
                                              ;now wait for flash ready then exit

eraseflsh:
                    bclr        lock_dis
                    bset        lock_dis      ;unlock all blocks

                    mov.b       #erase_cmd,r0l
                    ste.w       r0,[a1a0]     ;send erase command
                    mov.b       #cfm_cmd,r0l
                    ste.w       r0,[a1a0]     ;send confirm byte
;erase command sent, now wait for
;flash ready then exit (fall into flsh_bsy)

flsh_bsy:
                    btst        bsy_rdy
                    jeq         flsh_bsy

                    mov.w       #rd_fstat,r1
                    ste.w       r1,[a1a0]
                    lde.w       [a1a0],r1
```

```
                    mov.b       r1l,_flsh_stat ; get status (SRD) of
                                               ; last operation (error checking)

                    mov.b       #rda_cmd,r0l   ; Read array resets flash
                    jsr         Command_write
                    bclr        lock_dis
                    bclr        cpu_rwrt
                    bset        resbit,_pm01+1      ;above instruction clears
                                                    ;this bit!!!

                    rts                             ;return to ROM area

progflsh:
                    bclr        lock_dis
                    bset        lock_dis            ;unlock all blocks
                    mov.b       #wrt_cmd,r0l        ;Page program command
                    jsr         Command_write
                    mov.w       _ram_buff,r1        ;get ram buffer address
prog_loop:
                    xchg.w      r1,a0               ;save flash address
                    mov.w       [a0],r3             ;get data
                    xchg.w r1,a0                    ;get write to address
                    ste.w       r3,[a1a0]           ;and write
                    add.w       #2,r1
                    add.w       #2,a0
                    tst.b       #0ffh,a0
                    jnz         prog_loop
                    jmp         flsh_bsy            ;write page command
                                                    :complete, now wait
                                                    ;for flash ready then exit
Command_write:
                    btst        bsy_rdy             ; wait for flash ready
                    jz          Command_write
                    ste.w       r0,[a1a0]           ; write command to flash WSM
                    rts


_copy_end:
#pragma ENDASM

}


/***************************************************************************
Name:         Main()

Parameters:   none
Returns:      nothing

Description: Main program. For demonstration purposes, the main program erases
    the E0000H - EFFFFH block then writes the contents of ram_buff
    to the first (256 byte) page in that block. Ram_buff was given an absolute
    address (2000H) such that under KD30, the buffer could be edited,
    program run, then the flash viewed to see that it programmed. To
    use this program with KD30 on a Starter Kit, run the program in "free run"
```

```
    mode and do a "go free". Hit the reset icon on KD30. At this
    point the flash can be viewed. Note that when you reload a program into
    flash, KD30 erases the entire user flash.

    Notes on lock bit. WARNING!! locking a block IS NOT compatible with KD30
    and the ROM monitor. If you lock any block and attempt to
    download your program to the flash, it will mess up the ROM monitor and you
    will need to reprogram the ROM monitor back into the M16C/62 (i.e.
    using flashwriter). The lock disable bit in the 'flash memory
    control register' only over-rides the lock bits, not clear them. The only
    way to clear a lock bit is to erase the associated block. This example
    operates on a 'dummy' block for demonstrating lock bit programming.

    It is up to the main program (and user) to ensure that valid
    addresses are passed to the commands (no error checking). The assembler
    code supports reading the flash status (SRD) but main() doesn't use it.

***************************************************************************/


void main(void)
{
  long maxb_addr,writ_addr;

       maxb_addr = 0xefffe;         // for erasing
       writ_addr = 0xe0000;         // page to program
       transfer();                  // transfer assembler program to ram
       cpu_ini();                   // set processor for programming
       eraseblock(maxb_addr);
       writepage(writ_addr);        // assumes data in buffer, fill via KD30.

 // Now demonstrate locking a block, under KD30, view global 'lokbitstat'
       maxb_addr = 0xdfffe;
       lockblock(maxb_addr);        // also reads lock bit into 'lokbitstat'
       eraseblock(maxb_addr);       // clears lock bit..VERY Important for KD30! see
                                    //above text on lock bit.
       cpu_rstor();                 // restore processor modes

       while(1)
              {   //good ending
              }
 }


/***************************************************************************
Name:         transfer()
Parameters:   none
Returns:      nothing
Description:  Copies assembler code to program the flash into RAM

*************************************************************************** */


void transfer(void)
{
```

```
        romxfer = &copy_strt;               // point to assembler code in flash area
        ramxfer = (far int *)&ramcode;      // point to RAM where code is to be executed
                                            // from (get compiler warning if don't
                                            // typecast)

        while (romxfer < &copy_end)
        {
          *ramxfer = *romxfer;
          ramxfer++;
          romxfer++;
        }
}


/***********************************************************************
Name:         void cpu_ini(void)
Parameters:   none
Returns:      nothing
Description:  Sets the processor mode for programming flash. A wait state is
              added and the clock is set to the input frequency/2 (Xin/2).
              Original configuration saved in globals: cm0sav, cm1sav,pm01sav.

*********************************************************************** */

 void cpu_ini(void)
 {
    cm0sav = cm0;    // save current CPU modes and clock setting
    cm1sav = cm1;
    pm01sav = pm01;
    prcr = 3;        // protect off
    pm01 = wait1;           // insert wait state
    cm1 = 0x60;             // divide CPU clock by 2
    cm0 = 0x08;             // ensure high drive on clock
    prcr = 0;               // protection back on
 }


/***********************************************************************
Name:         void eraseblock(unsigned long int)
Parameters:   maxb_addr
Returns:      nothing, modifies global 'flsh_stat'
Description:  Erases the flash block at maxb_addr. maxb_addr is the last
              (even) address in the block to be erased
*********************************************************************** */

void eraseblock(unsigned long int maxb_addr)
  {
       flsh_addr = maxb_addr;
       cmndnum = aerase;
       interruptoff;
      ramcode();
       interrupton;
  }
```

```
/*
/*************************************************************************
Name:         void writepage(unsigned long int )
Parameters:   writ_addr
Returns:      nothing, modifies global 'flsh_stat'
Description:  writes a 256 byte page in flash, starting at address writ_addr.
              Function has fixed the ram buffer at RAMBUFFLOC for demonstration
              purposes.
************************************************************************* */


 void writepage(unsigned long int writ_addr)
  {
       flsh_addr = writ_addr;
       ram_buff = RAMBUFFLOC;
       cmndnum = aprogram;
       interruptoff;
       ramcode();
       interrupton;
  }


/*************************************************************************
Name:         lockblock(unsigned long int )
Parameters:   maxb_addr
Returns:      nothing, modifies globals 'lokbitstat','flsh_stat'
Description:  Programs the lock bit for the block at maxb_addr.  maxb_addr is
              the last (even) address in the block.
************************************************************************* */

 void lockblock(unsigned long int maxb_addr)
 {
       flsh_addr = maxb_addr;
       cmndnum = alocblock;
       interruptoff;
       ramcode();
       interrupton;
 }


/*************************************************************************
Name:       cpu_rstor(void)
Parameters: none
Returns:  nothing
Description:  Restores the processor mode back to original speed.
************************************************************************* */
```

```
 void cpu_rstor(void)
  {
 // restore CPU & clock settings
    prcr = 3;               // protect off
    pm01 = pm01sav;
    cm1 = cm1sav;
    cm0 = cm0sav;
    prcr = 0;               // protection back on
  }


/***********************************************************************
Name:         lockbitstat(unsigned long int)
Parameters:   maxb_addr
Returns:      nothing, modifies globals 'lokbitstat','flsh_stat'
Description:  Reads lock bit status determined by maxb_addr into global
              'lokbitstat'. maxb_addr is the last (even) address in the block.

*********************************************************************** */

  void lockbitstat(unsigned long int maxb_addr)
      {
           flsh_addr = maxb_addr;
           cmndnum = areadloc;
           interruptoff;
          ramcode();
           interrupton;
         }
```

Keep safety first in your circuit designs!

- Renesas Technology Corporation puts the maximum effort into making semiconductor products better and more reliable, but there is always the possibility that trouble may occur with them. Trouble with semiconductors may lead to personal injury, fire or property damage.
  Remember to give due consideration to safety when making your circuit designs, with appropriate measures such as (i) placement of substitutive, auxiliary circuits, (ii) use of nonflammable material or (iii) prevention against any malfunction or mishap.

Notes regarding these materials

- These materials are intended as a reference to assist our customers in the selection of the Renesas Technology Corporation product best suited to the customer's application; they do not convey any license under any intellectual property rights, or any other rights, belonging to Renesas Technology Corporation or a third party.
- Renesas Technology Corporation assumes no responsibility for any damage, or infringement of any third-party's rights, originating in the use of any product data, diagrams, charts, programs, algorithms, or circuit application examples contained in these materials.
- All information contained in these materials, including product data, diagrams, charts, programs and algorithms represents information on products at the time of publication of these materials, and are subject to change by Renesas Technology Corporation without notice due to product improvements or other reasons. It is therefore recommended that customers contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor for the latest product information before purchasing a product listed herein.
  The information described here may contain technical inaccuracies or typographical errors.
  Renesas Technology Corporation assumes no responsibility for any damage, liability, or other loss rising from these inaccuracies or errors.
  Please also pay attention to information published by Renesas Technology Corporation by various means, including the Renesas Technology Corporation Semiconductor home page (http://www.renesas.com).
- When using any or all of the information contained in these materials, including product data, diagrams, charts, programs, and algorithms, please be sure to evaluate all information as a total system before making a final decision on the applicability of the information and products. Renesas Technology Corporation assumes no responsibility for any damage, liability or other loss resulting from the information contained herein.
- Renesas Technology Corporation semiconductors are not designed or manufactured for use in a device or system that is used under circumstances in which human life is potentially at stake. Please contact Renesas Technology Corporation or an authorized Renesas Technology Corporation product distributor when considering the use of a product contained herein for any specific purposes, such as apparatus or systems for transportation, vehicular, medical, aerospace, nuclear, or undersea repeater use.
- The prior written approval of Renesas Technology Corporation is necessary to reprint or reproduce in whole or in part these materials.
- If these products or technologies are subject to the Japanese export control restrictions, they must be exported under a license from the Japanese government and cannot be imported into a country other than the approved destination.
  Any diversion or reexport contrary to the export control laws and regulations of Japan and/or the country of destination is prohibited.
- Please contact Renesas Technology Corporation for further details on these materials or the products contained therein.