AN2346/D
Rev. 0, 9/2002

EEPROM Emulation Using FLASH in MC68HC908QY/QT MCUs

by **Peter Topping**
Applications Engineering
Freescale, East Kilbride

Introduction

As versatile, low pin-count and low cost variants of the MC68HC(9)08 range of MCUs, the MC68HC908QY1, QY4, QT1 and QT4 have many potential applications. They incorporate easily and quickly programmable FLASH memory for their program code. Their cost is minimised by not adding any byte programmable EEPROM (Electrically Erasable Programmable Read Only Memory) as this functionality can be facilitated using a small portion of the FLASH memory.

There are many types of application which are enhanced by the inclusion of non-volatile data storage. External serial EEPROMs are thus sometimes added to systems using low-cost MCUs with no on-chip EEPROM. If, however, the chip has FLASH memory for its application software then a portion of it can be used to emulate EEPROM thus obviating the additional cost and complexity incurred by an extra chip. Using less than a hundred bytes of code, this application note presents a method of doing this. It allows the FLASH to behave like EEPROM and, at the same time, enhances its endurance in terms of the available number of write cycles.

The FLASH memory in the MC68HC908QY/QT family is organised in pages of 64 bytes. Although individual bytes can be programmed, an erase operation necessarily applies to a whole page. By using different bytes within a page each time a block of data is saved, the number of write cycles can be extended beyond its specification of 10,000. The improvement factor is the number of times the data block fits into a page. A 6-byte data block, for example, would fit in 10 times and thus guarantee 100,000 writes. Only once a page is full, and another data save is required, is the page erased and the cycle started again.

To use the EEPROM emulation code, the data to be saved is put into a RAM buffer of user-defined length. This unit will be referred to as a data “block” and can be any size from 1 byte to a full page (64 bytes) according to the

requirements of the application. Any number of block types consistent with the number of FLASH memory pages available can be used. Each block can be the most appropriate size for its particular data.

The method described here is just one of many different strategies which can be adopted to facilitate non-volatile data storage using FLASH rather than EEPROM. These vary greatly from using the FLASH to “shadow” a block of data held in RAM to setting up a file system which allows different types of tagged data to be saved cumulatively in an arbitrary order. In the latter method the tagging would allow a request for any data type to return its most recent value. Once the FLASH block was full, all the most recent data would be transferred to a second block and the process continued using the two blocks alternately. The method described here provides a combination of versatility and simplicity that is particularly appropriate for the MC68HC08 family and other low-end and mid-range MCU/MPU applications.

Emulated EPROM interface

EEPROM can be used for data-logging or the storing of equipment status during power-down, node addresses, calibration data, cryptographic keys, radio or television frequencies or channel numbers and for numerous other types of data. For this reason a versatile interface is required. The method adopted here is similar to that presented for the MC68HC908GP32 in Application Note AN2183. This allows a page of FLASH to be used for each data type using the block size most appropriate for that data. **Figure 1** shows a typical example of the use of the write routine presented here. It is assumed that the data to be saved is in RAM starting at address $8C (see below).

```
ldhx    #$F040      ;FLASH page address used for this data
lda     #7          ;data block size
jsr     WrtBlock    ;write the block of data from RAM to FLASH
```

Figure 1. Example use of the write routine

The block size can be as small as a single byte and as large as a full page of 64 bytes. However, as the FLASH programming routine attempts to save the data in a different place in the page each time it is called, there is no benefit if the block size is over 32 bytes. This is because there would be room for only a single block in the page which would be erased, and the same FLASH bytes written to, every time data was saved.

Often the different bytes of data will serve different purposes when a particular block is read so the reading routine does not actually read the whole block. Instead it returns the start address of the most recently saved data block in the 16-bit index register (H:X) and the first byte of the data in the accumulator.

Figure 2 shows the code required. Once the address is available the user can use indexed addressing to access the required bytes. This method has the advantage of not forcing the use of a RAM buffer while still allowing the retrieved data to be used as required for the particular application.

```
   ldhx    #$F040       ;FLASH page for this data
   lda     #7           ;data block size
   jsr     RdBlock      ;get pointer to latest data block (1st byte in A)
```

Figure 2. Basic use of the read routine

In a radio, for example, each station could have two bytes of frequency information and eight bytes of ASCII data for the station name. The frequency bytes would need to be latched into a PLL and the station name data sent serially to a display module. There could be a further byte to specify waveband etc., which should be put onto an I/O port. In this type of application, the designer may wish this data to be saved as a single 11-byte data block or split up into 2 or 3 separate blocks. The method presented here allows either strategy.

Sometimes, for example when fetching a cryptographic key, there may be a requirement to transfer some or all of the data block into a contiguous area of RAM. The example code shown below illustrates the use of the read routine to perform this function. It transfers a complete block of data into RAM starting at the address defined by DATA. If DATA is $8C then this code exactly complements the write routine which transfers a block in the other direction. Although the write routine must use $8C, this data reading software could use any available RAM locations.

```
         ldhx    #$F040       ;example FLASH page for data
         lda     #7           ;example data block size
         psha                 ;save initial byte count (block size)
         jsr     RdBlock      ;get pointer to latest data block
         txa                  ;get address LS byte
         add     1,sp         ;add block size
         deca                 ;and decrement
         tax                  ;H:X now point to last byte in block

again:   pshx
         pshh                 ;save H:X
         lda     ,x           ;get a byte of data
         clrh
         ldx     3,sp         ;byte count now in H:X
         sta     DATA-1,x     ;put byte into RAM
         pulh
         pulx                 ;retrieve FLASH data pointer
         decx                 ;and point to next (previous) byte
         dec     1,sp         ;decrement byte count
         bne     again        ;finished ?

         pula                 ;fix stack
```

Figure 3. Example use of the read routine to retrieve a complete block of data

The data saving method presented here necessarily includes some history of the saved data. In some applications this may be of value. The number of old blocks available will of course depend on the current position in the page and there will sometimes be none. If historical data is always required then it is possible to use two pages for the same block of data thus guaranteeing that at least one page of historical data will always be available. This capability is not included in this application note but it could be added to the application software prior to calling the read and write routines. Clearly the most important aspect would be to keep track of which of the two pages holds the most recent data. This approach would also increase further the number of write cycles available for this block of data.

One disadvantage of the simple method presented here is that it necessarily assumes that the block writing process will not be interrupted by a power-fail or any other unexpected event that stops the application. If this occurs during the data saving procedure the page being written to (or erased) may be left in an intermediate state from which valid data is not available. It is up to the system designer to minimise the possibility of this happening and/or facilitate acceptable recovery or default behaviour if the data is corrupted. In this respect, however, the emulated EEPROM is superior to most serial EEPROM implementations due to the much shorter writing time.

FLASH memory

The FLASH memory used in the MC68HC908 family of devices allows very fast programming. Including software overhead, programming can be carried out at over 10 bytes per millisecond which is a factor of a hundred faster than most EEPROMs. An additional consideration is the page erase time of 4ms but this doesn't occur prior to every write. Careful management in the application software can thus avoid always having to allow for the possibility of this happening prior to saving data if this potential delay is unacceptable.

In the case of the MC68HC908QY/QT devices, a page of FLASH consists of two rows of 32 bytes each for a total of 64 bytes. FLASH memory is programmed a row (or part of a row) at a time but erased in pages. Although some data sheets discourage writing to a particular row more than once without erasing it in between, there is no technical reason why this should not be done. It is also allowable to write to only part of a row at a time, there being in practice no minimum number of bytes which must be programmed each time a row is written to.

The only restriction is that the total write time between erases should not exceed t_{HV} (4ms) per row. This is ensured by the software: each of the 32 bytes is only written once between erases so the maximum time is 32 x 35µs i.e. 1.12ms. In this application the number of bytes in a block is not restricted so

there will sometimes be some unused bytes at the end of the page. Clearly block sizes of 1, 2, 4 etc. (any power of 2) will use all 64 bytes in their page.

The code section of any FLASH based application should always be protected against accidental erasure using, in the case of MC68HC908 MCUs, the FLASH block protection register, FLBPR[1]. As it works by protecting all FLASH above a particular memory address, the area of FLASH used as EEPROM should be at the start (lowest address) of the FLASH memory. This allows it to be enabled for erasure and programming while the program code, starting at a higher address, is fully protected. (see references 1 and 4).

On-chip ROM routine

Like other small members of the MC68HC908 family, the MC68HC908QY and QT devices have FLASH program/erase software included in on-chip ROM code. This code is used during factory test and burn-in. On larger devices like the GP32 this testing is carried out using code downloaded into RAM but variants with 256 bytes or less of RAM (128 in the case of the MC68HC908QY/QT) cannot do this efficiently because of the limited space.

The code included in the MC68HC908GR, KX, JL/JK and JB devices is described in Application Note AN1831. The code in the MC68HC908QY and QT devices operates in the same way, the only significant difference being the entry addresses for the routines. The ROM routines PrgRnge and EraRnge are used in this application note. Their use requires the equates shown below.

```
EraRnge     equ      $2806      ;FLASH erase routine in internal ROM
PgrRnge     equ      $2809      ;FLASH program routine in internal ROM
```

Figure 4. Equates required to access the on-chip ROM subroutines

The read routine in ROM is not used in this application. It can read a whole data block and place it in RAM and is even capable of verifying the contents of FLASH against RAM. Usually neither of these functions will be required in an actual application and the much simpler read routine RdBlock is used here.

The programming routine in ROM, PrgRnge, can program over row boundaries. This greatly simplifies the block search code developed for this

1. The FLASH block protection register, FLBPR, is actually a page protection register as it can be specified to protect the FLASH in increments of a page. In this application note, the word “block” refers to the user-defined data block that can be any size from 1 byte to a full page of 64 bytes.

application note as there is no requirement to take the row boundary into account. The whole page is thus available to hold as many blocks as possible.

The programming routine uses RAM location CPUSpd ($89) to determine the bus speed, LstAddr ($8A and $8B) to save the end address in FLASH and a data buffer starting at BfrStrt ($8C). The data buffer is the length of a data block. Before writing a block it is thus necessary to put the data to be stored into RAM locations from BfrStrt to BfrStrt +blocksize-1 and to leave the 4 locations from $88 to $8B available for use by the ROM routines. The first FLASH address is held in the index register and thus does not require to be stored in RAM. The erase routine in ROM, EraRnge, uses CtrlByt ($88) for control information.

The use of the 4 bytes is shown in **Figure 5** and described in more detail in Application Note AN1831. It is important that CtrlByt and CPUSpd are initialised correctly. CtrlByt allows the on-chip erase routine to distinguish between a page erase and a mass erase (not used in this application). CPUSpd tells the erase and programming routines what the bus speed is so that the program and erase delays can be calculated correctly. Correct timing assumes that there are no interrupts during erasing or programming and they are automatically disabled during the execution of the ROM subroutines. The application code should re-enable interrupts if required.

Location	RAM address	Bytes	Use
CtrlByt	$88	1	Control bits
CPUSpd	$89	1	Bus speed in units of 0.25MHz
LstAddr	$8A – $8B	2	FLASH block end address
BfrStrt	$8C =>	block size	Data buffer

Figure 5. RAM locations used by the on-chip ROM subroutines

The software subroutines in ROM handle the 4 RAM locations and no intervention is required except to change the data written to CPUSpd in the “WrtBlock” routine. This is shown as 13 (decimal) assuming that the internal clock is being used to obtain a 3.2 MHz bus speed. It should be changed if required to the value of the actual bus speed being used. The number should be the bus speed in units of 0.25MHz.

The data buffer at BfrStrt is the size of a data block. As multiple data block sizes are possible, the simplest way to organise an application’s RAM would be to allocate a data buffer the same length as the largest block used. This is however not strictly necessary as only the RAM used for a particular block is

required and any unused RAM can be utilised for other purposes. Indeed all of the RAM used by the EEPROM emulation code can serve other purposes when it is not actually required for saving data to non-volatile memory. Clearly care would be required, perhaps by permanently allocating the required RAM, if saving data could be initiated by an interrupt.

Software

The key to this type of use of FLASH is knowing where the latest block of data is situated within its page. This is required so that the latest block can be read and so that, if new data has to be written, it is put into the next available block-sized space in the page. If there is no room then the whole page is erased and the data is written at the start of the page. The current location could be held in RAM but would need to be remembered for each data type. Even more troublesome would be the requirement to provide non-volatile storage of this information so the strategy adopted here avoids the need to remember the current position.

Instead, every time a read or write is requested, the page is scanned to find the location of the latest data or the first available erased block. This has the disadvantage that the signature used to signify an unused block ($FF in the first byte) has to be forbidden as valid data and it is up to the main application software to ensure that this doesn't occur. Clearly this signature could be made less restrictive by modifying the code to require that more bytes (perhaps the complete block) have to be erased ($FF) to signify an unused block. Alternatively, a dummy byte could be added at the start of the data block thus avoiding any restrictions on the data.

The search is performed by the subroutine "FindClear" which is used by both the read and write procedures to determine the status of the data in the page. The subroutine requires that the block size is pushed onto the stack before it is called. It subtracts this size from the page size to obtain the bytes remaining after the first block and then reads the first byte of the first block. If it is $FF, the subroutine exits with $FF in the accumulator to indicate that an erased block was found. The first block will in fact only be erased if data has never been stored to this page so this is a special case.

Usually the first read will not be $FF and the subroutine uses the number of bytes remaining after the first block to check if there is room for another. If not then the subroutine exits with the accumulator clear to indicate that no erased block is available. If there is room, the code checks the first byte of the next block for the signature of $FF. This process is repeated until the location of the first erased block (if there is one) is found. On exit from "FindClear", the index register contains the address of the next available block unless their isn't one in which case it points to the last complete block.

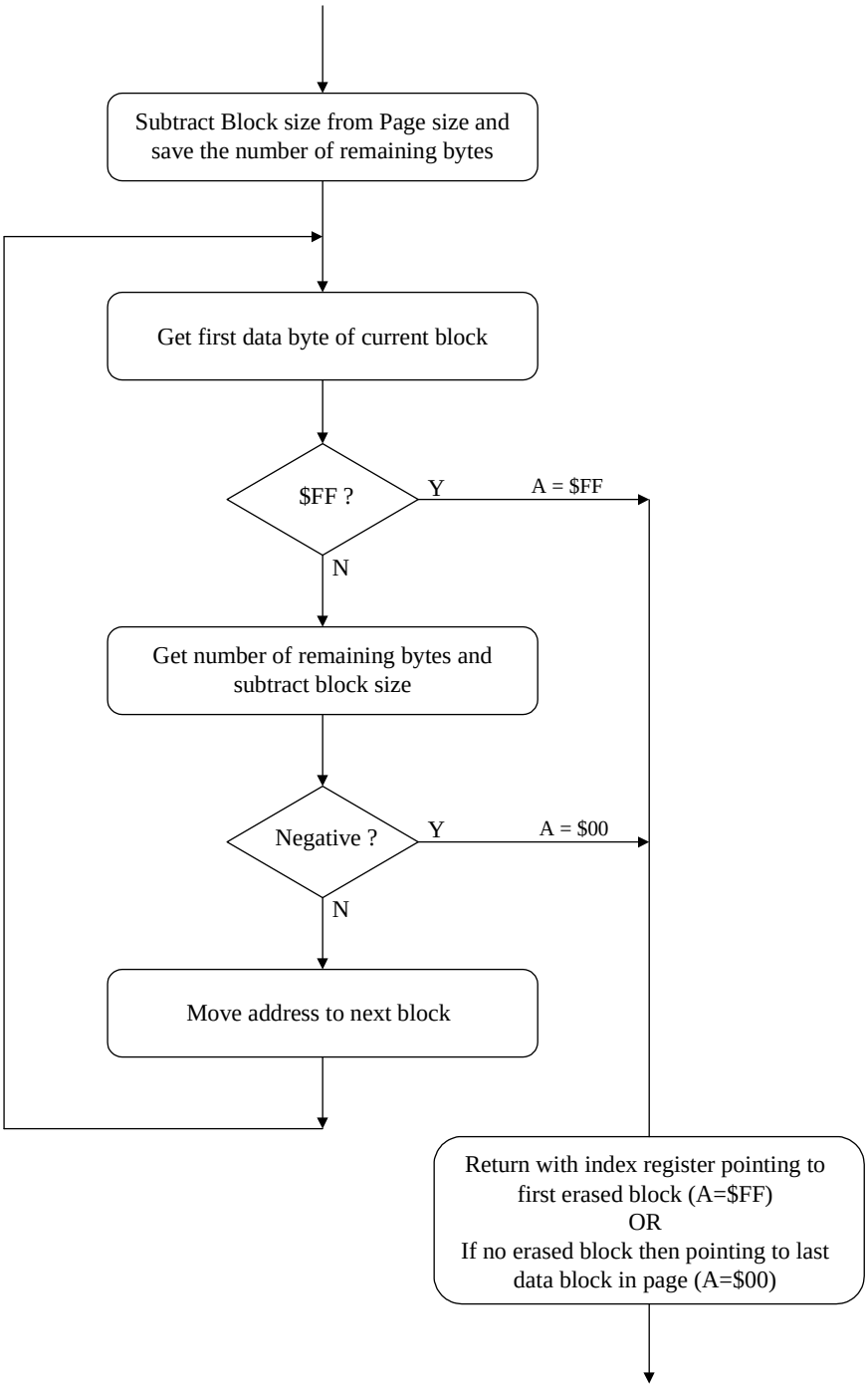

Figure 6. FindClear flow diagram

The Write routine “WrtBlock” initialises the RAM locations CtrlByt and CPUSpd and pushes the block size onto the stack before calling “FindClear”. It then checks the accumulator and, if it is $FF, goes ahead and writes the data block using the address left in the 16-bit index register (H:X) by “FindClear”. If it isn’t $FF, there is no room for another block and the page is erased and the address initialised to the start of the page. The data can then be written. This involves saving in RAM (at LstAddr) the address of the last byte to be written before calling the programming subroutine.

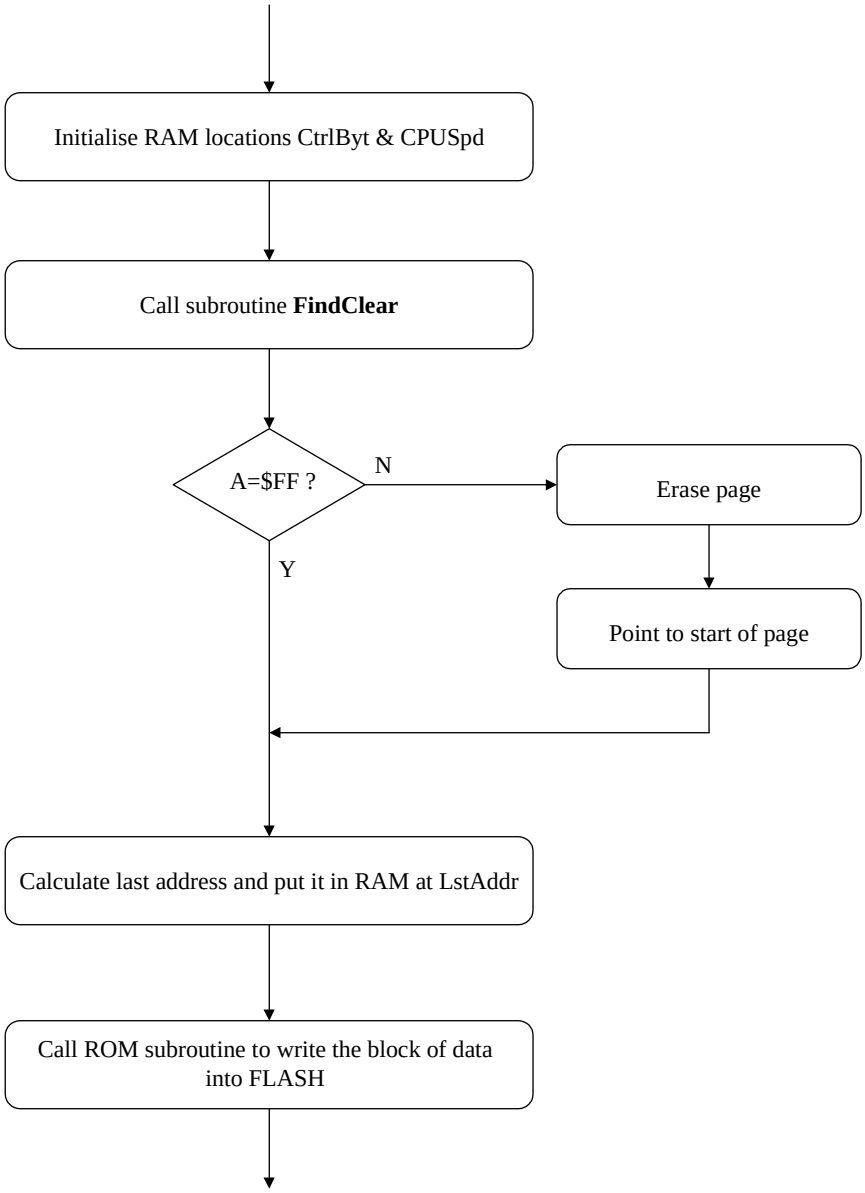

Figure 7. WrtBlock flow diagram

The read routine also uses “FindClear” to determine the status of the page but, as the address of the first erased block is returned, it has to go back a block to access the data. There are two exceptional cases when going back a block is not appropriate. If the page is full and no erased block is found, “FindClear” returns the address of the last complete block of data so the address is already correct for reading. Also, in the situation where there is no saved data, “FindClear” will return the address of the first block in the page. If this happens, going back a block would go into the previous page. The address is therefore not modified and the data, including the first byte returned in the accumulator, will be $FF.

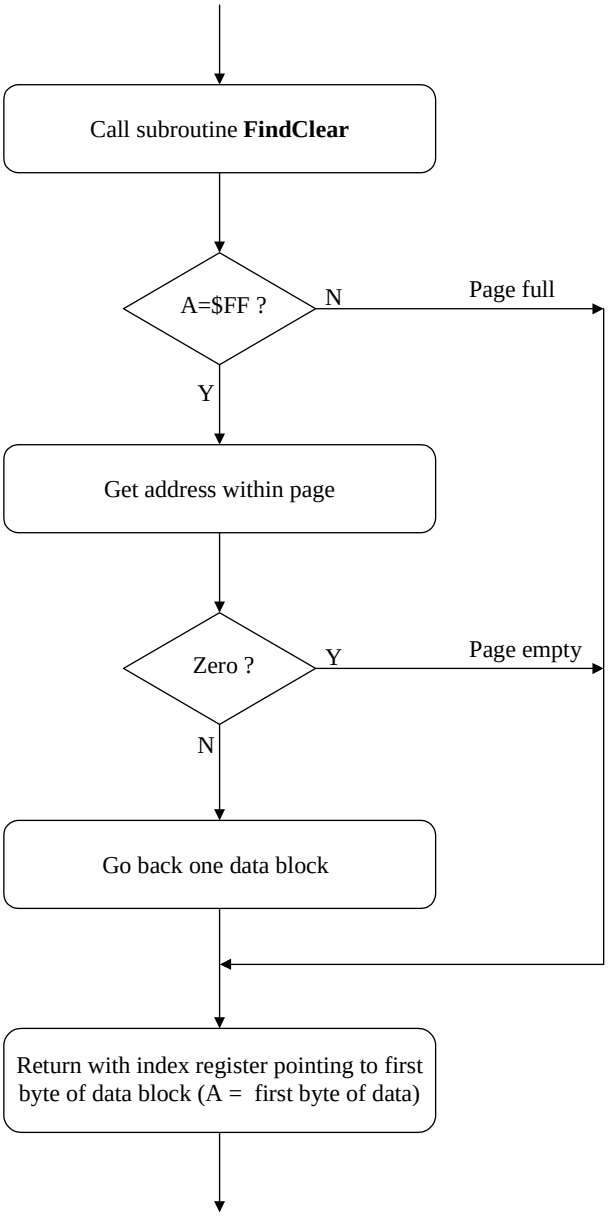

Figure 8. RdBlock flow diagram

References

1. MC68HC908QY4/D, technical data sheet
2. Application Note AN1831, “Using MC68HC908 On-chip FLASH Programming Routines”
3. Application Note AN2183, “Using FLASH as EEPROM on the MC68HC908GP32”
4. Engineering Bulletin EB398, “Techniques to Protect MCU Applications Against Malfunction due to Code Run-away”

Software listing

```
                        1
                        2  ;*********************************************************************
                        3  ;*                                                                   *
                        4  ;*        EEPROM Emulation using FLASH in MC68HC908QT/QY MCUs        *
                        5  ;*                                                                   *
                        6  ;*  QTEEApp.asm                          Copyright (c)  2002  *
                        7  ;*                                                                   *
                        8  ;*********************************************************************
                        9  ;*                                                                   *
                       10  ;*  Description: Read and Write subroutines which facilitate the     *
                       11  ;*  saving and retrieval of blocks of data of user-defined size      *
                       12  ;*  in non-volatile FLASH memory in such a way that the write-erase  *
                       13  ;*  cycling capability of the FLASH is extended up to 64 times its   *
                       14  ;*  specification of 10,000 cycles.                                  *
                       15  ;*                                                                   *
                       16  ;*  Include files: none                                              *
                       17  ;*                                                                   *
                       18  ;*  Documentation: MC68HC908QY4/D Technical Data Sheet.              *
                       19  ;*  Application Note AN2346 - "EEPROM Emulation using FLASH in       *
                       20  ;*  MC68HC908QT/QY MCUs".                                            *
                       21  ;*                                                                   *
                       22  ;*  This software is classified as Engineering Sample Software.     *
                       23  ;*                                                                   *
                       24  ;*********************************************************************
                       25  ;*                                                                   *
                       26  ;* Author:       Peter Topping - TSPG Applications - East Kilbride   *
                       27  ;*                                                                   *
                       28  ;* Update History:                                                   *
                       29  ;*                                                                   *
                       30  ;* Rev     Date        Author  Description of Change                 *
                       31  ;* ------  ---------   ------  ------------------------------------- *
                       32  ;* ES 0.1  22-Aug-02   PT      Initial release                       *
                       33  ;*                                                                   *
                       34  ;*                                                                   *
                       35  ;*                                                                   *
                       36  ;*                                                                   *
                       37  ;*********************************************************************
                       38  ;*                                                                   *
                       39  ;* Freescale reserves the right to make changes without further notice*
                       40  ;* to any product herein to improve reliability, function, or design.*
                       41  ;* Freescale does not assume any liability arising out of the        *
                       42  ;* application or use of any product, circuit, or software described *
                       43  ;* herein; neither does it convey any license under its patent rights*
                       44  ;* nor the rights of others. Freescale products are not designed,    *
                       45  ;* intended, or authorized for use as components in systems intended *
                       46  ;* for surgical implant into the body, or other applications intended*
                       47  ;* to support life, or for any other application in which the failure*
                       48  ;* of the Freescale product could create a situation where personal   *
                       49  ;* injury or death may occur.  Should Buyer purchase or use Freescale *
                       50  ;* products for any such intended or unauthorized application, Buyer *
                       51  ;* shall indemnify and hold Freescale and its officers, employees,    *
                       52  ;* subsidiaries, affiliates, and distributors harmless against all   *
                       53  ;* claims, costs, damages, and expenses, and reasonable attorney     *
                       54  ;* fees arising out of, directly or indirectly, any claim of personal*
                       55  ;* injury or death associated with such unintended or unauthorized   *
                       56  ;* use, even if such claim alleges that Freescale was negligent      *
                       57  ;* regarding the design or manufacture of the part.                  *
                       58  ;*                                                                   *
                       59  ;* Freescale is a registered trademark of Freescale Semiconductor, Inc. *
                       60  ;*                                                                   *
                       61  ;*********************************************************************
                       62
                       63  ;*  Equates for ROM Subroutines and start of RAM
                       64
0000                   65  EraRnge      equ    $2806         ;FLASH erase routine in ROM
0000                   66  PgrRnge      equ    $2809         ;FLASH programming routine in ROM
0000                   67  CtrlByt      equ    $88           ;control byte for ROM subroutines
0000                   68  CPUSpd       equ    $89           ;CPU speed in units of 0.25MHz
0000                   69  LstAddr      equ    $8A           ;last FLASH address to be programmed
                       70
                       71
FC00                   72               org    $FC00
                       73
                       74
```

```
                        75
                        76  ;*********************************************************************
                        77  ;*                                                                   *
                        78  ;*  RdBlock - Reads a block of data from FLASH and puts it in RAM    *
                        79  ;*                                                                   *
                        80  ;*  Calling convention:    ldhx   #Blk1page                        *
                        81  ;*                        lda    #Blk1Size                        *
                        82  ;*                        jsr    RdBlock                          *
                        83  ;*                                                                   *
                        84  ;*  Inputs:  H:X - pointing to start of FLASH page used for data     *
                        85  ;*           A   - block size                                        *
                        86  ;*                                                                   *
                        87  ;*  Returns: H:X - pointing to start of FLASH block containing data  *
                        88  ;*           A   - data from first byte of block                     *
                        89  ;*                                                                   *
                        90  ;*  Uses:    FindClear                                              *
                        91  ;*                                                                   *
                        92  ;*********************************************************************
                        93
FC00 [02] 87            94  RdBlock:     psha                ;save block size
FC01 [04] AD32          95              bsr    FindClear    ;find first erased block
                        96
FC03 [02] A1FF          97              cmp    #$FF         ;was an erased block found ?
FC05 [03] 260A          98              bne    skipdec      ;if not then don't go back a block
FC07 [01] 9F            99              txa                 ;get LS byte of address
FC08 [02] A43F         100              and    #$3F         ;only look at address within page
FC0A [03] 2705         101              beq    skipdec      ;if 0 then no data so don't go back
FC0C [01] 9F           102              txa                 ;if not get LS byte of address again
FC0D [04] 9EE001       103              sub    1,sp         ;and subtract block size to point
FC10 [01] 97           104              tax                 ;to start of valid data block
                       105
FC11 [02] F6           106  skipdec:     lda    ,x           ;get first byte of data
FC12 [02] A701         107              ais    #1           ;de-allocate stack
FC14 [04] 81           108              rts
                       109
                       110
                       111  ;*********************************************************************
                       112  ;*                                                                   *
                       113  ;*  WrtBlock - Writes a block of data into FLASH from RAM buffer     *
                       114  ;*                                                                   *
                       115  ;*  Calling convention:    ldhx   #Blk1page                        *
                       116  ;*                        lda    #Blk1Size                        *
                       117  ;*                        jsr    WrtBlock                         *
                       118  ;*                                                                   *
                       119  ;*  Inputs:  H:X - pointing to start of FLASH page used for data     *
                       120  ;*           A   - block size                                        *
                       121  ;*                                                                   *
                       122  ;*  Returns: nothing                                                 *
                       123  ;*                                                                   *
                       124  ;*  Uses:    FindClear, EraRnge (ROM), PgrRnge (ROM)                *
                       125  ;*                                                                   *
                       126  ;*********************************************************************
                       127
FC15 [04] 6E0D89       128  WrtBlock:    mov    #13,CPUSpd   ;3.2MHz/0.25MHz = 13
FC18 [03] 3F88         129              clr    CtrlByt      ;page (not mass) erase
FC1A [02] 87           130              psha                ;save block size
FC1B [04] AD18         131              bsr    FindClear    ;find first available erased block
FC1D [02] A1FF         132              cmp    #$FF         ;erased block found ?
FC1F [03] 2707         133              beq    blkfnd       ;if so write to it
FC21 [05] CD2806       134              jsr    EraRnge      ;if not then erase page
FC24 [01] 9F           135              txa                 ;get LS byte of FLASH address
FC25 [02] A4C0         136              and    #$C0         ;and reset it to start of page
FC27 [01] 97           137              tax                 ;H:X now pointing to first block
                       138
FC28 [02] 86           139  blkfnd:      pula                ;get block size
FC29 [02] 89           140              pshx                ;save start address LS byte
FC2A [04] 9EEB01       141              add    1,sp         ;add block size to LS byte
FC2D [01] 4A           142              deca                ;back to last address in block
FC2E [01] 97           143              tax                 ;last address now in H:X
FC2F [04] 358A         144              sthx   LstAddr      ;save in RAM for use by ROM routine
FC31 [02] 88           145              pulx                ;restore X (H hasn't changed)
FC32 [03] CC2809       146              jmp    PgrRnge      ;program block (includes RTS)
                       147
                       148
```

```
                        149  ;*********************************************************************
                        150  ;*                                                                   *
                        151  ;*  FindClear - Finds first erased block within page                  *
                        152  ;*                                                                   *
                        153  ;*  Inputs:  H:X - pointing to start of page used for required data  *
                        154  ;*           Stack - block size last thing on stack                  *
                        155  ;*                                                                   *
                        156  ;*  Returns if erased block found:                                   *
                        157  ;*           H:X - pointing to start of first erased block in page   *
                        158  ;*           A   - $FF                                               *
                        159  ;*  Returns if no erased block found (page full):                    *
                        160  ;*           H:X - pointing to start of last written block           *
                        161  ;*           A   - $00                                               *
                        162  ;*                                                                   *
                        163  ;*********************************************************************
                        164
FC35 [02] A640          165  FindClear:   lda    #$40          ;number of bytes in a page
FC37 [04] 9EE003        166               sub    3,sp          ;less number in first block
FC3A [02] 87            167               psha                 ;save bytes left
                        168
FC3B [02] F6            169  floop:       lda    ,x            ;get first data byte in block
FC3C [02] A1FF          170               cmp    #$FF          ;erased byte ?
FC3E [03] 270F          171               beq    finish1       ;if so then exit, otherwise try next
                        172
FC40 [02] 86            173               pula                 ;bytes left
FC41 [04] 9EE003        174               sub    3,sp          ;less number in next block
FC44 [02] 87            175               psha                 ;resave bytes left
FC45 [03] 2B07          176               bmi    finish2       ;enough for another block ?
                        177
FC47 [01] 9F            178               txa                  ;yes, get LS byte of address
FC48 [04] 9EEB04        179               add    4,sp          ;add block size
FC4B [01] 97            180               tax                  ;put it back (can't be a carry)
FC4C [03] 20ED          181               bra    floop         ;and try again
                        182
FC4E [01] 4F            183  finish2:     clra                 ;no room (A shouldn't be $FF)
FC4F [02] A701          184  finish1:     ais    #1            ;fix stack pointer
FC51 [04] 81            185               rts
                        186
                        187
                        188
```

Symbol Table

```
BLKFND          FC28
CPUSPD          0089
CTRLBYT         0088
ERARNGE         2806
FINDCLEAR       FC35
FINISH1         FC4F
FINISH2         FC4E
FLOOP           FC3B
LSTADDR         008A
PGRRNGE         2809
RDBLOCK         FC00
SKIPDEC         FC11
WRTBLOCK        FC15
```

Appendix

The code shown in the listing in this appendix is an alternative to that shown in the previous section. It uses a different WrtBlock routine that does not use the erase routine, EraRnge, which is included in the on-chip ROM.

This alternative code is appropriate for use with early mask sets of the MC68HC908QY/QT (1L69J, 2L69J and 3L69J) which have the FLASH control logic error described in errata 68HC908QY/QTMSE3.

There are two blocks of FLASH memory in the M68HC908QY/QT MCU which are selected internally by array select signals. Address values are protected against changes after a page erase sequence has started. Any attempt to write a new address after HVEN=1 is blocked. However, due to a logic error in these mask sets, the latching of the array select signals is not blocked so it is possible that one page in one array could be unintentionally erased when a page erase is performed on a page in the other array.

EraRnge refreshes the COP by periodically writing to address $FFFF. This is in the top FLASH array so a write to this location while erasing (using EraRnge) a FLASH page in the bottom array ($EE00-$FDFF) can result in the erroneous erasure of a page in the top array. This occurs regardless of the protection status of the page in the top array.

To avoid this problem it is thus necessary, on the mask sets with this problem, to avoid using the on-chip erase routine. The alternative code shown below replaces this routine with one downloaded into RAM.

Although functionally equivalent, the replacement software uses half of the available RAM (from $C0 to $FF) and is thus intended only as an interim solution until silicon without the logic fault is available.

Alternative software listing

```
                          1  ;*********************************************************************
                          2  ;*                                                                   *
                          3  ;*        EEPROM Emulation using FLASH in MC68HC908QT/QY MCUs        *
                          4  ;*                                                                   *
                          5  ;*  This listing includes an alternative Write subroutine to the     *
                          6  ;*  one presented in Application Note AN2346. It avoids using the    *
                          7  ;*  908QT/QY erase routine in ROM and thus the additional page erase *
                          8  ;*  described in Errata 68HC908QY/QTMSE3. It downloads code into     *
                          9  ;*  RAM and uses all of the top half of the RAM (from $C0 to $FF).   *
                         10  ;*                                                                   *
                         11  ;*  The main subroutine "WrtBlock" is the same as in the Application *
                         12  ;*  Note code except that it calls "EEEPage" instead of the ROM      *
                         13  ;*  subroutine "EraRnge". As long as this change is made to          *
                         14  ;*  WrtBlock, the only aditional code required is "EEEPage" and      *
                         15  ;*  "EEEinRAM". The FLASH reading routime "RdBlock" and the          *
                         16  ;*  subroutine "FindClear" are identical to those in the Application *
                         17  ;*  Note.                                                            *
                         18  ;*                                                                   *
                         19  ;*  Peter Topping                                  18th July 2002   *
                         20  ;*                                                                   *
                         21  ;*********************************************************************
                         22
0000                     23  PgrRnge      equ    $2809         ;FLASH programming routine in ROM
0000                     24  CtrlByt      equ    $88           ;control byte for ROM subroutines
0000                     25  CPUSpd       equ    $89           ;CPU speed in units of 0.25MHz
0000                     26  LstAddr      equ    $8A           ;last FLASH address to be programmed
                         27
                         28  ;*  Additional equates
0000                     29  ERASE        equ    %00000010     ;erase bit in FLCR
0000                     30  HVEN         equ    %00001000     ;high voltage bit in FLCR
0000                     31  ERAHVEN      equ    %00001010     ;erase and high voltage bits in FLCR
0000                     32  FLBPR        equ    $FFBE         ;flash block protect reg (flash)
0000                     33  FLCR         equ    $FE08         ;FLASH control register
                         34
FD00                     35                org    $FD00
                         36
                         37  ;*********************************************************************
                         38  ;*                                                                   *
                         39  ;*  RdBlock - Reads a block of data from FLASH and puts it in RAM    *
                         40  ;*                                                                   *
                         41  ;*  Calling convention:    ldhx   #Blk1page                         *
                         42  ;*                         lda    #Blk1Size                         *
                         43  ;*                         jsr    RdBlock                           *
                         44  ;*                                                                   *
                         45  ;*  Inputs:  H:X - pointing to start of FLASH page used for data     *
                         46  ;*           A   - block size                                        *
                         47  ;*                                                                   *
                         48  ;*  Returns: H:X - pointing to start of FLASH block containing data  *
                         49  ;*           A   - data from first byte of block                     *
                         50  ;*                                                                   *
                         51  ;*  Uses:    FindClear                                             *
                         52  ;*                                                                   *
                         53  ;*********************************************************************
                         54
FD00 [02] 87             55  RdBlock:     psha                 ;save block size
FD01 [04] AD32           56                bsr    FindClear     ;find first erased block
FD03 [02] A1FF           57                cmp    #$FF          ;was an erased block found ?
FD05 [03] 260A           58                bne    skipdec       ;if not then don't go back a block
FD07 [01] 9F             59                txa                  ;get LS byte of address
FD08 [02] A43F           60                and    #$3F          ;only look at address within page
FD0A [03] 2705           61                beq    skipdec       ;if 0 then no data so don't go back
FD0C [01] 9F             62                txa                  ;if not get LS byte of address again
FD0D [04] 9EE001         63                sub    1,sp          ;and subtract block size to point
FD10 [01] 97             64                tax                  ;to start of valid data block
                         65
FD11 [02] F6             66  skipdec:     lda    ,x            ;get first byte of data
FD12 [02] A701           67                ais    #1            ;de-allocate stack
FD14 [04] 81             68                rts
                         69
                         70
                         71
                         72
                         73
                         74
```

```
                        75  ;*********************************************************************
                        76  ;*                                                                   *
                        77  ;*  WrtBlock - Writes a block of data into FLASH from RAM buffer     *
                        78  ;*                                                                   *
                        79  ;*  Calling convention:    ldhx   #Blk1page                       *
                        80  ;*                        lda    #Blk1Size                       *
                        81  ;*                        jsr    WrtBlock                        *
                        82  ;*                                                                   *
                        83  ;*  Inputs:  H:X - pointing to start of FLASH page used for data     *
                        84  ;*           A   - block size                                        *
                        85  ;*                                                                   *
                        86  ;*  Returns: nothing                                                 *
                        87  ;*                                                                   *
                        88  ;*  Uses:    FindClear, EEEPage, EEEinRAM (RAM), PgrRnge (ROM)       *
                        89  ;*                                                                   *
                        90  ;*********************************************************************
                        91
FD15 [04] 6E0D89        92  WrtBlock:    mov    #13,CPUSpd    ;3.2MHz/0.25MHz = 13
FD18 [03] 3F88          93              clr    CtrlByt       ;page (not mass) erase
FD1A [02] 87            94              psha                 ;save block size
FD1B [04] AD18          95              bsr    FindClear     ;find first available erased block
FD1D [02] A1FF          96              cmp    #$FF          ;erased block found ?
FD1F [03] 2707          97              beq    blkfnd        ;if so write to it
FD21 [05] CDFD52        98              jsr    EEEPage       ;if not then erase page
FD24 [01] 9F            99              txa                  ;get LS byte of FLASH address
FD25 [02] A4C0         100              and    #$C0          ;and reset it to start of page
FD27 [01] 97           101              tax                  ;H:X now pointing to first block
                       102
FD28 [02] 86           103  blkfnd:      pula                 ;get block size
FD29 [02] 89           104              pshx                 ;save start address LS byte
FD2A [04] 9EEB01       105              add    1,sp          ;add block size to LS byte
FD2D [01] 4A           106              deca                 ;back to last address in block
FD2E [01] 97           107              tax                  ;last address now in H:X
FD2F [04] 358A         108              sthx   LstAddr       ;save in RAM for use by ROM routine
FD31 [02] 88           109              pulx                 ;restore X (H hasn't changed)
FD32 [03] CC2809       110              jmp    PgrRnge       ;program block (includes RTS)
                       111
                       112  ;*********************************************************************
                       113  ;*                                                                   *
                       114  ;*  FindClear - Finds first erased block within page                 *
                       115  ;*                                                                   *
                       116  ;*  Inputs:  H:X - pointing to start of page used for required data  *
                       117  ;*           block size last thing on stack                          *
                       118  ;*                                                                   *
                       119  ;*  Returns if erased block found:                                   *
                       120  ;*           H:X - pointing to start of first erased block in page   *
                       121  ;*           A   - $FF                                               *
                       122  ;*  Returns if no erased block found (page full):                    *
                       123  ;*           H:X - pointing to start of last written block           *
                       124  ;*           A   - $00                                               *
                       125  ;*                                                                   *
                       126  ;*********************************************************************
                       127
FD35 [02] A640         128  FindClear:   lda    #$40          ;number of bytes in a page
FD37 [04] 9EE003       129              sub    3,sp          ;less number in first block
FD3A [02] 87           130              psha                 ;save bytes left
                       131
FD3B [02] F6           132  floop:       lda    ,x            ;get first data byte in block
FD3C [02] A1FF         133              cmp    #$FF          ;erased byte ?
FD3E [03] 270F         134              beq    finish1       ;if so then exit, otherwise try next
                       135
FD40 [02] 86           136              pula                 ;bytes left
FD41 [04] 9EE003       137              sub    3,sp          ;less number in next block
FD44 [02] 87           138              psha                 ;resave bytes left
FD45 [03] 2B07         139              bmi    finish2       ;enough for another block ?
                       140
FD47 [01] 9F           141              txa                  ;yes, get LS byte of address
FD48 [04] 9EEB04       142              add    4,sp          ;add block size
FD4B [01] 97           143              tax                  ;put it back (can't be a carry)
FD4C [03] 20ED         144              bra    floop         ;and try again
                       145
FD4E [01] 4F           146  finish2:     clra                 ;no room but A can't be $FF
FD4F [02] A701         147  finish1:     ais    #1            ;fix stack pointer
FD51 [04] 81           148              rts
```

```
                        149
                        150 ;*********************************************************************
                        151 ;*                                                                   *
                        152 ;*  EEEPage - Erases a page of emulated EEPROM FLASH                 *
                        153 ;*                                                                   *
                        154 ;*  Calling convention:    ldhx   #EEPage                           *
                        155 ;*                         jsr    EEEpage                           *
                        156 ;*                                                                   *
                        157 ;*  Inputs:  H:X - pointing into FLASH page to be erased             *
                        158 ;*                                                                   *
                        159 ;*  Returns: H:X - unchanged                                         *
                        160 ;*                                                                   *
                        161 ;*********************************************************************
                        162
FD52 [02] 89            163 EEEPage:     pshx                ;save FLASH address in RAM for
FD53 [02] 8B            164              pshh                ;retrieval from within RAM routine
FD54 [03] 450034        165              ldhx   #RAMsize     ;get size of RAM resident routine
FD57 [04] D6FD66        166 loadloop:    lda    EEEinRAM-1,x ;get a byte of code
FD5A [02] 87            167              psha                ;and put it into RAM
FD5B [03] 5BFA          168              dbnzx  loadloop     ;finished ?
FD5D [01] 85            169              tpa                 ;get CCR
FD5E [02] 9B            170              sei                 ;disable interrupts
FD5F [02] 95            171              tsx                 ;pointer to RAM routine
FD60 [04] FD            172              jsr    ,x           ;execute RAM routine
FD61 [02] A734          173              ais    #RAMsize     ;de-allocate stack space
FD63 [02] 8A            174              pulh                ;restore FLASH address
FD64 [02] 88            175              pulx
FD65 [02] 84            176              tap                 ;restore CCR
FD66 [04] 81            177              rts
                        178
                        179 ;*********************************************************************
                        180 ;*                                                                   *
                        181 ;*  EEEinRAM - RAM resident part of EEEPage                          *
                        182 ;*                                                                   *
                        183 ;*  Calling convention:    ldhx   #{pointer to routine}             *
                        184 ;*                         jsr    ,x                                *
                        185 ;*                                                                   *
                        186 ;*  Delays calculated to give the required times assuming the bus    *
                        187 ;*  clock is 3.2MHz + 25% ie 4.0MHz.                                 *
                        188 ;*                                                                   *
                        189 ;*********************************************************************
                        190
FD67 [02] 87            191 EEEinRAM:    psha                ;save CCR
FD68 [04] D60034        192              lda    (RAMsize),x  ;retrieve FLASH address MSB from RAM
FD6B [04] DE0035        193              ldx    (RAMsize+1),x ;and LS byte
FD6E [02] 87            194              psha
FD6F [02] 8A            195              pulh                ;MSB into h (address is now in H:X)
FD70 [02] A602          196              lda    #ERASE
FD72 [04] C7FE08        197              sta    FLCR         ;set ERASE bit in control register
FD75 [04] C6FFBE        198              lda    FLBPR        ;read block protection register
FD78 [02] F7            199              sta    ,x           ;write to an address within page
FD79 [02] A60E          200              lda    #14          ;3 cycle loop so 14 times for delay
FD7B [03] 4BFE          201              dbnza  *            ;of 10us at 4 MHz (14*3/4MHz=10.5us)
                        202
FD7D [02] A60A          203              lda    #ERAHVEN     ;ERASE and HVEN bit
FD7F [04] C7FE08        204              sta    FLCR         ;set HVEN bit in control register
FD82 [02] AE28          205              ldx    #40          ;40 times
FD84 [02] A686          206 tloop:       lda    #134         ;100us delay
FD86 [03] 4BFE          207              dbnza  *            ;for 4ms of HVEN high
FD88 [03] 5BFA          208              dbnzx  tloop        ;40*(5+134*3)/4MHz=4070us
                        209
FD8A [02] A608          210              lda    #HVEN
FD8C [04] C7FE08        211              sta    FLCR         ;clear ERASE bit
FD8F [02] A607          212              lda    #7           ;3 cycle loop so 7 times for delay
FD91 [03] 4BFE          213              dbnza  *            ;of 10us at 4 MHz (7*3/4MHz=5.2us)
                        214
FD93 [01] 4F            215              clra
FD94 [04] C7FE08        216              sta    FLCR         ;clear HVEN bit
FD97 [02] 86            217              pula                ;restore CCR (2 cycles)
FD98 [03] 21FE          218              brn    *            ;3 more cycles ie >1us
FD9A [04] 81            219              rts
                        220
FD9B                    221 RAMsize      equ    (*-EEEinRAM)
                        222
```

```
 Symbol Table

BLKFND          FD28
CPUSPD          0089
CTRLBYT         0088
EEEINRAM        FD67
EEEPAGE         FD52
ERAHVEN         000A
ERASE           0002
FINDCLEAR       FD35
FINISH1         FD4F
FINISH2         FD4E
FLBPR           FFBE
FLCR            FE08
FLOOP           FD3B
HVEN            0008
LOADLOOP        FD57
LSTADDR         008A
PGRRNGE         2809
RAMSIZE         0034
RDBLOCK         FD00
SKIPDEC         FD11
TLOOP           FD84
WRTBLOCK        FD15
```

How to Reach Us:

Home Page:
www.freescale.com

E-mail:
support@freescale.com

USA/Europe or Locations Not Listed:
Freescale Semiconductor
Technical Information Center, CH370
1300 N. Alma School Road
Chandler, Arizona 85224
+1-800-521-6274 or +1-480-768-2130
support@freescale.com

Europe, Middle East, and Africa:
Freescale Halbleiter Deutschland GmbH
Technical Information Center
Schatzbogen 7
81829 Muenchen, Germany
+44 1296 380 456 (English)
+46 8 52200080 (English)
+49 89 92103 559 (German)
+33 1 69 35 48 48 (French)
support@freescale.com

Japan:
Freescale Semiconductor Japan Ltd.
Headquarters
ARCO Tower 15F
1-8-1, Shimo-Meguro, Meguro-ku,
Tokyo 153-0064
Japan
0120 191014 or +81 3 5437 9125
support.japan@freescale.com

Asia/Pacific:
Freescale Semiconductor Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
P.O. Box 5405
Denver, Colorado 80217
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

freescale™
semiconductor