Freescale Semiconductor
Application Note

AN2881
Rev. 0, 1/2005

Mapping Memory Resources on HCS12 Microcontrollers

by: Luis Reynoso Covarrubias
RTAC Americas Mexico 2004

Introduction

This document is intended to serve as a quick reference for embedded microcontroller engineers to correctly memory map registers, RAM, and EEPROM while running any HCS12 MCU. Basic knowledge about the functional description and configuration options will provide a better understanding of how the memory mapping works. This application note provides examples illustrating how to perform the memory mapping within the HCS12 Family of microcontrollers. The examples mentioned are intended to be modified to suit the specific needs of any application.

Description

The HCS12 Family of microcontrollers has the functionality to remap its internal registers, RAM memory, and EEPROM memory within some boundaries defined for each microcontroller. This functionality allows the capability to change the default address for each module. This allows a better optimization of the available resources with possibilities such as:

- Direct memory addressing to any memory location:

  The HCS12 CPU has a direct addressing mode using zero-page ($00XX) with only eight bits to address the memory. Using direct addressing mode, the instructions are faster and consume less code; however, the main problem is this addressing mode can only be used for the first 256 bytes

of memory. By default, the HCS12 has its internal registers in the zero-page. This is very useful for most applications; however, some applications require an extensive use of RAM memory or continuous access to EEPROM locations. Using memory mapping to change the RAM or EEPROM to zero-page allows faster access to these memory locations and uses less code.

- Complete use of memory locations:

  Some HCS12 derivatives have overlapped memory locations by default for such reasons as:
  - having more memory than addressable by 64 Kbytes
  - or legacy, allowing compatibility with other devices

  This is useful for many applications, but in some cases, may require a different use of memory resources. In this case the user can select appropriate memory locations for RAM, EEPROM, and internal registers in order to avoid overlapping, thus maximizing the use of the internal resources.

Information available in this quick reference can be found in greater detail in the *Device User Guide* and the *Module Mapping Control Manual* (Freescale order number S12MMCV4).

Each HCS12 device has its own memory resources. For this reason, some parameters such as memory size, allowable mapping boundaries, and register functionality depends on the device. It should be the first consideration to remap the memory.

Memory information is included in the Device User Guide for each microcontroller as follows:

- The Device Memory Map section which contains information about the memory resources and boundaries. Please note that some Device User Guides have different memory maps after reset while others have a useful recommended configuration map instead of the map out at reset.
- HCS12 memory map examples like that shown in Figure 1 for the MC9S12DJ64 and MC9S12DP256B.

For more detailed information, refer to the *Device User Guide* for each microcontroller.

Table 1. MC9S12DJ64 and MC9S12DP256B Memory Characteristics

		Size	Boundary	Start Address After Reset	Available Range After Reset	Special Characteristics
MC9S12DJ64	Registers	1K	2K	$0000	$0000 – $03FF	Mappable only within the first 32K
	RAM	4K	4K	$0000	$0400 – $0FFF	
	EEPROM	1K	2K	$0000	Not Available	Mapped twice in the 2K space
MC9S12DP256B	Registers	1K	2K	$0000	$0000 – $03FF	Mappable only within the first 32K
	RAM	12K	16K	$1000	$1000 – $3FFF	Can be aligned to top or bottom
	EEPROM	4K	4	$0000	$0400 – $0FFF	

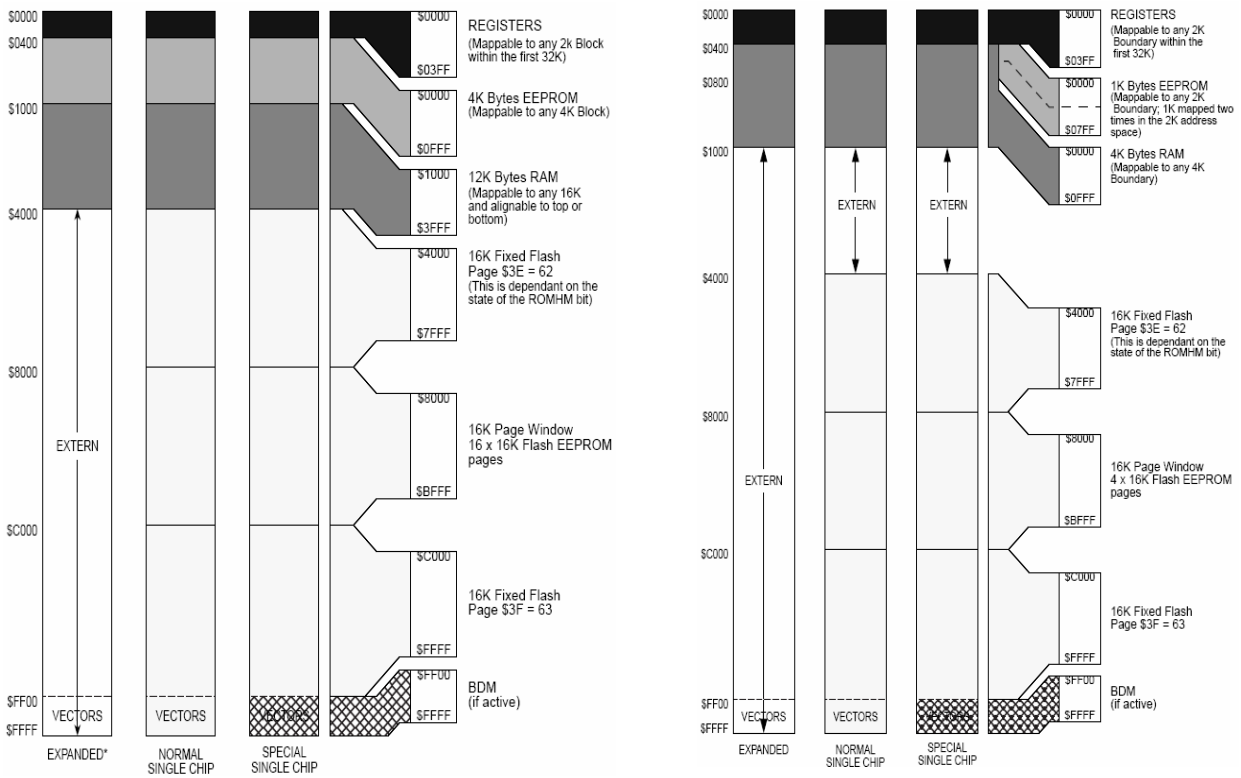

Figure 1. HCS12 Memory Map Examples (MC9S12DP256B vs. MC9S12DJ64)

Although some microcontrollers contain identical addresses for RAM, EEPROM, or registers, only one will be available. Therefore, some memory sections are unavailable. This will be defined by the HCS12 Family memory priority outlined in Table 2.

Table 2. Memory Priority

Priority	Address Space
Highest	BDM (internal to Core) firmware or register space
—	Internal register space
—	RAM memory block
—	EEPROM memory block
—	On-chip FLASH or ROM
Lowest	Remaining external space

Registers Descriptions

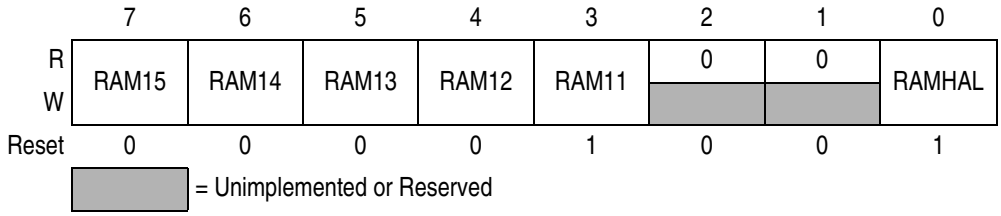

Figure 2. Initialization of Internal RAM Position Register (INITRM)

RAM15:RAM11

These bits determine the upper five bits of the base address for the system's internal RAM.

RAMHAL

0 = Aligns RAM to the lowest address
1 = Aligns RAM to the highest address

The initialization of internal RAM position register (INITRM) is the register to map the internal RAM memory. This required register initializes the position of internal RAM within the on-chip system memory map.

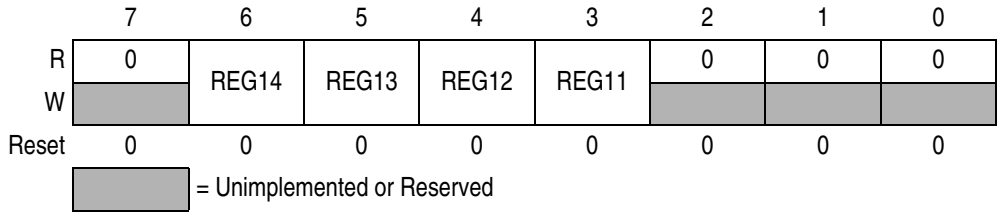

Figure 3. Initialization of Internal Registers Position Register (INITRG)

REG14:REG11

These four bits, in combination with the leading zero supplied by bit 7 of INITRG, determine the upper five bits of the base address for the system's internal registers. For example, the minimum base address is $0000 and the maximum is $7FFF.

The initialization of internal registers position register (INITRG) is the register required to map the internal registers. This register initializes the position of internal registers within the on-chip system memory map. The registers occupy either a 1 Kbyte or 2 Kbyte space and can be mapped to any 2 Kbyte space within the first 32 Kbytes of the system's address space.

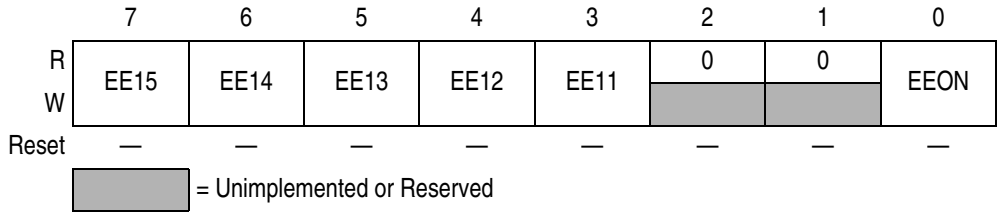

Figure 4. Initialization of Internal EEPROM Position Register (INITEE)

EE15:EE11

These bits determine the upper five bits of the base address for the system's internal EEPROM array.

EEON

0 = Disables the EEPROM from the memory map
1 = Enables the EEPROM

The initialization of internal EEPROM position register (INITEE) is the register required to map the internal EEPROM. This register initializes the position of the internal EEPROM within the on-chip system memory map.

Value Selection

In order to map the required section the following steps should be considered:

1. Consider size and boundary size of the section.

   Example:

   According to their device user guide, the MC9S12DJ64 and MC9S12DP256 have the RAM memory shown in Table 3.

Table 3. MC9S12DJ64 and MC9S12DP256B RAM Characteristics

MCU	RAM Size	RAM Boundary
MC9S12DJ64	4K	4K
MC9S12DP256B	12K	16K

2. Select a desired start address for the memory section boundary.

   Example:

   According to the defined boundaries, the allowed RAM start memory locations for the MC9S12DJ64 and MC9S12DP256B are shown in Table 4.

Table 4. MC9S12DJ64 and MC9S12DP256B Available Addresses

MC9S12DJ64	
Start Address	**End Address**
0×0000	0×0FFF
0×1000	0×1FFF
0×2000	0×2FFF
. . . .	. . . .
0×E000	0×EFFF
0×F000	0×FFFF

MC9S12DP256B	
Start Address	**End Address**
0×0000	0×3FFF
0×4000	0×7FFF
0×8000	0×BFFF
0×C000	0×FFFF

3. Select the proper value for position bits. Since some sections can be mapped only in defined boundaries, some position bits will be unimplemented.

   Example:

   The available values for RAM15:RAM11 for the MC9S12DJ64 and MC9S12DP256B are shown in Table 5.

Table 5. MC9S12DJ64 and MC9S12DP256B RAM15:RAM11 Available Values

MC9S12DJ64					
Boundary Start Address	RAM15	RAM14	RAM13	RAM12	RAM11
0×0000	0	0	0	0	x
0×1000	0	0	0	1	x
0×2000	0	0	1	0	x
. . . .	.	.	.	.	x
0×E000	1	1	1	0	x
0×F000	1	1	1	1	x

MC9S12DP256B					
Boundary Start Address	RAM15	RAM14	RAM13	RAM12	RAM11
0×0000	0	0	x	x	x
0×4000	0	1	x	x	x
0×8000	1	0	x	x	x
0×C000	1	1	x	x	x

4. Fill the additional bits for the register:

   a. If the RAM is being mapped:

      Align RAM memory. Some microcontrollers have a smaller memory size than the memory boundary; the RAMHAL allows the user to align the memory to be available at the lowest address or the higher address.

      Example:

      Since the MC9S12DJ64 has 4K of RAM in a 4K boundary, this bit is unimplemented. On the other hand, the MC9S12DP256 has 12K of RAM in a 16K boundary. In this way, depending on the RAMHAL value, the start location could be 0x0000 or 0x1000 as shown in Table 6.

Table 6. MC9S12DJ64 and MC9S12DP256B Using RAMHAL

MC9S12DJ64			
0×0000	RAMHAL=0		RAMHAL=1
0×0FFF			

MC9S12DP256B			
0×0000	RAMHAL=0		RAMHAL=1
0×1000			
0×2000			
0×3000			
0×3FFF			

   b. If the EEPROM is being mapped:

      If the EEPROM is required, set the EEON bit, otherwise clear it.

5. Write register value:

   Example:

   The available values for INITRM, INITEE, and INITRG for the MC9S12DJ64 and MC9S12DP256B are shown in Table 7, Table 8, and Table 9.

Table 7. INITRG Values for MC9S12DJ64 and MC9S12DP256B

REG14	REG13	REG12	REG11	INITRG	Start Address	End Address
0	0	0	0	0×00	0×0000	0×07FF
0	0	0	1	0×08	0×0800	0×0FFF
0	0	1	0	0×10	0×1000	0×17FF
0	0	1	1	0×18	0×1800	0×1FFF
. . . .	. . . .	. . . .	. . . .	. . . .	. . . .	. . . .
1	1	1	0	0×70	0×7000	0×77FF
1	1	1	1	0×78	0×7800	0×7FFF

Table 8. INITRM Values for MC9S12DJ64 and MC9S12DP256B

MC9S12DJ64								
RAM15	RAM14	RAM13	RAM12	RAM11	RAMHAL	INITRM	Start Address	End Address
0	0	0	0	x	x	0×00	0×0000	0×0FFF
0	0	0	1	x	x	0×10	0×1000	0×1FFF
0	0	1	0	x	x	0×20	0×2000	0×2FFF
0	0	1	1	x	x	0×30	0×3000	0×3FFF
.	.	.	.	x	x	. . .	. . .	. . .
1	1	0	1	x	x	0×D0	0×D000	0×DFFF
1	1	1	0	x	x	0×E0	0×E000	0×EFFF
1	1	1	1	x	x	0×F0	0×F000	0×FFFF

MC9S12DP256B								
RAM15	RAM14	RAM13	RAM12	RAM11	RAMHAL	INITRM	Start Address	End Address
0	0	x	x	x	0	0×00	0×0000	0×2FFF
0	0	x	x	x	1	0×01	0×1000	0×3FFF
0	1	x	x	x	0	0×04	0×4000	0×6FFF
0	1	x	x	x	1	0×41	0×5000	0×7FFF
1	0	x	x	x	0	0×80	0×8000	0×AFFF
1	0	x	x	x	1	0×41	0×9000	0×BFFF
1	1	x	x	x	0	0×C0	0×C000	0×EFFF
1	1	x	x	x	1	0×C1	0×D000	0×FFFF

Table 9. INITEE Values for MC9S12DJ64 and MC9S12DP256B

MC9S12DJ64								
EE15	EE14	EE13	EE12	EE11	EEON	INITEE	Start Address	End Address
0	0	0	0	0	1	0×01	0×0000	0×07FF
0	0	0	0	1	1	0×09	0×0800	0×0FFF
0	0	0	1	0	1	0×11	0×1000	0×17FF
0	0	0	1	1	1	0×19	0×1800	0×1FFF
.	.	.	.	.	1	. . .	. . .	. . .
1	1	1	1	1	1	0×F1	0×F000	0×F7FF
1	1	1	1	1	1	0×F9	0×F800	0×FFFF

MC9S12DP256B								
EE15	EE14	EE13	EE12	EE11	EEON	INITEE	Start Address	End Address
0	0	0	0	x	1	0×01	0×0000	0×0FFF
0	0	0	1	x	1	0×11	0×1000	0×1FFF
0	0	1	0	x	1	0×21	0×2000	0×2FFF
0	0	1	1	x	1	0×31	0×3000	0×3FFF
.	.			x	1	. . .	. . .	. . .
1	1	1	0	x	1	0×E1	0×E000	0×EFFF
1	1	1	1	x	1	0×F1	0×F000	0×FFFF

Required Code and Explanation

The code required to initialize these registers is very simple. But, it is very important to write these registers at the beginning of the program so that further accesses to RAM, registers, and EEPROM have the right meaning.

1. Define the required registers.

   Example for MC9S12DJ64

```
#define ___INITRM      (*(volatile unsigned char *) 0x0010)
#define ___INITRG      (*(volatile unsigned char *) 0x0011)
#define ___INITEE      (*(volatile unsigned char *) 0x0012)
```

NOTE

These registers might be already defined by the stationary.

2. After reset (for example, in _Startup( ) function, using CodeWarrior), write the proper values for these registers.

   Example for MC9S12DJ64:

```
___INITEE = 0x11;  /* lock EEPROM block from 0x1000 to 0x17FF(mapped twice) */
___INITRM = 0x00; /* lock Ram from 0x0000 to 0x0FFF */
___INITRG = 0x18; /* lock registers block from 0x1800 to 0x1FFF */
```

3. At this moment, the blocks are mapped correctly in the MCU; however, we must tell the linker where to accommodate the memory blocks. In the PRM, modify the proper SEGMENTS and PLACEMENTS.

   Example for MC9S12DJ64:

```
SEGMENTS
       RAM = READ_WRITE 0x0000 TO 0x0FFF;
       /* unbanked FLASH ROM */
       ROM_C000 = READ_ONLY  0xC000 TO 0xFEFF;
       /* banked FLASH ROM */
       PAGE_3C = READ_ONLY  0x3C8000 TO 0x3CBFFF;
       PAGE_3D = READ_ONLY  0x3D8000 TO 0x3DBFFF;
       PAGE_3E = READ_ONLY  0x3E8000 TO 0x3EBFFF;
       EEPROM  = READ_ONLY  0x1000 TO 0x13FF;
END
PLACEMENT
PRESTART, STARTUP, ROM_VAR, STRINGS,
VIRTUAL_TABLE_SEGMENT, NON_BANKED, COPY
                          INTO  ROM_C000/*, ROM_4000*/;
DEFAULT_ROM               INTO  PAGE_3C,PAGE_3D,PAGE_3E;
DEFAULT_RAM               INTO  RAM;
MY_EEPROM                 INTO  EEPROM;
END
```

   You can find more information about the PRM file in the SmartLinker manual for CodeWarrior at: http://www.metrowerks.com/MW/Support/dev_resources/Documentation_for_HC12_3.1.htm

4. Select a proper base for all the registers, so they can be accessed properly by the software. Using CodeWarrior, this can be easily done by changing the REG_BASE # define in the microcontroller header file.

   Example for MC9S12DJ64:

   In mc9s12dj64.h, modify the following line:

   ```
   #define REG_BASE 0x0000    /* Base address for the I/O register block */
   ```

   to the following:

   ```
   #define REG_BASE 0x1800    /* Base address for the I/O register block(
                                 0x1800 – 0x1FFF) */
   ```

5. When the EEPROM is being mapped, the following restriction should be considered.

   The programmer software used to burn the code in the HCS12 can have a fixed area for EEPROM memory. If the user's code initializes some EEPROM locations in a different location than that used by the programmer, the data won't be written in EEPROM because the programmer can't recognize these locations as EEPROM.

   For example:

   CodeWarrior programmer uses the following location for EEPROM memory for the MC9S12DJ64[1]:

Start	End
$1000	$17FF

   It must be noted that this restriction applies only to data initialized by the programmer. This is because the EEPROM will be mapped correctly, data can be accessed correctly in the new mapped locations, and the user can write their own code to write and erase EEPROM locations for these new locations.

Example Description

The example included for this quick reference performs the following steps using the steps previous described:

- Initializes the memory blocks of the MC9S12DJ64 to the following locations:

	Start Address	End Address
RAM	0×0000	0×0FFF
EEPROM	0×1000	0×13FF
Registers	0×1800	0×1FFF

- Initializes an array in the new EEPROM location as "Freescale":

  ```
  #pragma CONST_SEG MY_EEPROM
  const unsigned char array [ ]  = "Freescale'"
  #pragma CONST_SEG DEFAULT
  ```

1. According to MCU-id 0x03C9 from True-Time Simulator & Real-Time Debugger, and can be seen in "Menu->ICD-12->Flash" for this and other derivatives.

- Declares a near variable in new RAM location:

```
volatile near unsigned char temp;
```

- Sets the PA0 pin as an output and turns it on, using the registers in new locations:

C Code	Generated Assembly
DDRA = 0x01; PORTA = 0z01;	LDBA #1 STAB 0X1802 STAB 0X1800

- Assigns the value of ‘r’ in “Freescale” to the variable in the new RAM location:

C Code	Generated Assembly
temp = array [1];	LDAA 0X1001 STAA 0X00

NOTE

Variable ‘temp’ is accessed using only eight bits because it’s declared on zero-page. All EEPROM register and RAM locations are accessed using the new mapped address.

The memory map will be remapped as show in Figure 5.

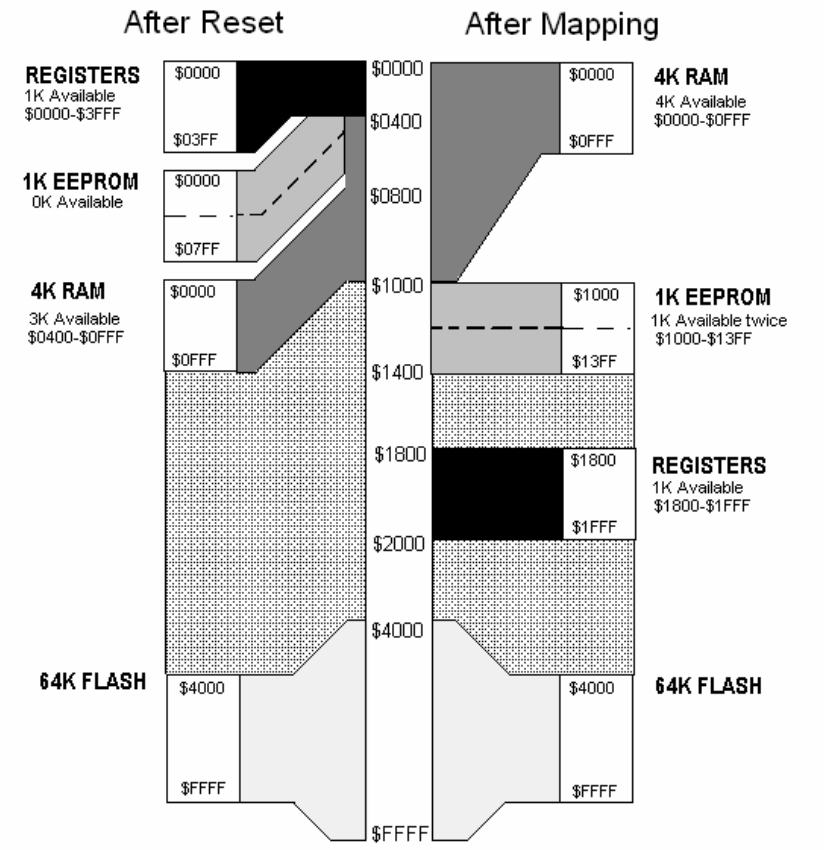

Figure 5. Memory Map Before and After Mapping

Considerations

The main considerations to take into account are that this example was developed using Metrowerks CodeWarrior IDE version 3.1 for HC(S)12, and the available example was expressly made for the MC9S12DJ64. It is important to consider that every microcontroller needs an initialization code which depends on the application and the microcontroller itself.

References

- HCS12 Family
  http://www.freescale.com/webapp/sps/site/taxonomy.jsp?nodeId=0162468636K100
- Module Mapping Control (MMC) V4 Block User Guide
  http://www.freescale.com/files/microcontrollers/doc/ref_manual/S12MMCV4.pdf
- MC9S12DJ64 Device User Guide
  http://www.freescale.com/files/microcontrollers/doc/data_sheet/9S12DJ64DGV1.pdf
- Metrowerks CodeWarrior documentation
  http://www.metrowerks.com/MW/Support/dev_resources/Documentation_for_HC12_3.1.htm

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

AN2881
Rev. 0, 1/2005

freescale™ semiconductor