Freescale Semiconductor
Application Note

Document Number: AN1926
Rev. 3, 03/2010

Production Flash Programming

Techniques for Production Programming the 56F80x, 56F826, and 56F827 Flash Memory Blocks

by: William Hutchings

1 Introduction

This application note presents techniques and detailed information on production programming of the Program, Data, and Boot Flash memory blocks in the 56F801, 56F803, 56F805, 56F807, 56F826, and 56F827 components. This is distinct from the developmental loading of the Flash blocks that is achieved using developmental tools such as the CodeWarrior for Freescale controller debugger.

There are four ways to program the Flash blocks in a factory environment:

- Using the Serial Bootloader present in the Boot Flash
- Using a commercially-available device programmer
- Using the JTAG/OnCE™ port
- Using the parallel Flash programming mode (except 56F801)

Contents

1 Introduction 1
2 Background Information 2
2.1 Considerations for Choosing a Production Programming Method 2
3 Programming Method Details 4
3.1 Serial Bootloader 4
3.2 Bulk Device Loader 5
3.3 In-circuit JTAG/OnCE Port 6
3.4 Parallel Flash Programming Mode 6
4 Conclusion 7
5 References 8
Appendix A S-Record Specification 9

The first two methods do not require any developmental effort to use. Using the JTAG/OnCE port currently requires the user to develop his own loader program. The parallel Flash programming mode requires the user to develop custom software for his board testing devices.

2 Background Information

As a starting point, all methods require an application to be created using the CodeWarrior development tools. This application must be specifically targeted to operate correctly from the internal Flash. The Freescale CodeWarrior Development tools include information as well as code to aid in the development of the application. It is highly recommended that a developer start with the appropriate Processor Expert (PE) stationery. This will greatly facilitate the development of the application, and includes all the elements required to create embedded applications targeting internal Flash or external memory. During development and debug of the application, the CodeWarrior tool can be used to program the internal Flash and to debug the program while running from Flash. Once the development and test are complete, the CodeWarrior tool can also be used to generate the source file containing the executable image of the Program, Data and Boot Flash blocks. In this case, the source file is an S record file. The S record file contains the hex machine code and hex data information formatted in a text file containing a series of S3-type S records. Refer to **Appendix A, "S-Record Specification,"** for a detailed description of the format of the S record file.

The CodeWarrior tool normally uses *.elf* files to store the application executable and to program the Flash blocks in the devices. The user must specifically configure the CodeWarrior tool to create the S record file. Please refer to the **CodeWarrior IDE Targeting 56800 Manual** for information on how to turn on S record generation. The CodeWarrior tool generates three separate types of S record files:

- *output_filename.p.S* contains the Program and Boot Flash image
- *output_filename.x.S* contains the Data Flash image
- *output_filename.S* contains the combined Program, Boot and Data Flash image

The S record file is the source file used by all the methods of production Flash programming. The file to use when performing Flash programming will generally be the combined file, containing the Program, Boot, and Data Flash image.

2.1 Considerations for Choosing a Production Programming Method

A fundamental decision to be made is whether to initially program the devices after they have been soldered to the circuit board, or to initially program the devices before they are placed onto the circuit board. Programming with the Boot Flash Serial Bootloader, programming via the JTAG/OnCE port and the parallel Flash programming mode are primarily intended as in-circuit programing methods. The bulk device load method is an out-of-circuit programming method, which allows programming the 56F800 devices in advance of the production run.

An additional consideration is which Flash blocks are to be programmed. The 56F800 products do not allow for software to run from a Flash block that is concurrently being erased and programmed. Because all 56F800 products have multiple Program Flash blocks, this is not a significant restriction, but does require advance planning with the set-up of the Flash programming algorithms. For example, the Serial Bootloader is programmed into the Boot Flash at the factory, and allows only for the programming of the

Program and Data Flash blocks because it restricts itself to only taking space in the Boot Flash. Therefore, it always runs from Boot Flash and cannot reprogram any portion of the Boot Flash block. The Serial Bootloader can be used to load all Flash blocks by implementing custom loader software that is loaded into the Program Flash, then used to program the Boot Flash. By switching back and forth, all combinations of Flash programming can be implemented. The Freescale Processor Expert for the 56800 devices includes all software components needed to write such an application. Since the JTAG/OnCE port, the bulk device loader, and the parallel mode options do not require that software be run from any of the Flash blocks; they can reprogram any block without special considerations. The 56F800 chips are fully field reprogrammable, requiring no special voltages or external hardware to perform this function. The stock Serial Bootloader fully supports field upgrades.

All in-circuit Flash loading methods require that the target hardware conforms to certain minimal standards. For the Serial Bootloader, these standards require that:

- the SCI port must be accessible for serial communications
- to enable proper SCI communications, the external clock for the components must be set to the recommended frequency. For 56F800 components, 8MHz is recommended; 4MHz is recommended for the 56F826 and 56F827.

Although the known external clock frequencies are necessary to ensure proper SCI communications, it's possible to drive the clock inputs during Serial Bootloading, then run the normal program at a different clock frequency. In general, this will not be necessary, since in all likelihood the best choice for the external operating frequency for the target circuit board will be the same frequency required by the Serial Bootloader. The possible exception is the 56F801, since its Serial Bootloader does not use the internal relaxation oscillator. To use the Serial Bootloader, drive the external clock input with 8MHz during the Serial Bootloading process, then use the internal relaxation oscillator for the main program.

Using the JTAG/OnCE port method requires only that a subset of the JTAG/OnCE pins be made available. Please refer to the section on this programming method for details.

Using the parallel programming method requires that the customer be able to drive 49 pins with different signals. This method places the chip into a special mode by which pin signal definitions are redefined to enable Flash block programming. This places certain restrictions on the board, in that the redefined signals must be capable of being driven to high and low level by the test equipment.

The approximate speeds at which the Flash blocks can be programmed with the various methods are:

- **Serial Bootloader** version 1.0 = 330 words per second
- **Serial Bootloader** version 1.1 and beyond = 1740 words per second
- **Bulk device programmer** = 535 words per second
- **In-circuit JTAG/OnCE port** = 204 words per second (using a parallel port command converter). The upper limit to this method is approximately 535 word per second.
- **Parallel programming mode** = 47,600 words per second maximum

Some versions of the bulk device programmers are multi-site, enabling the programming of several devices at one time.

3 Programming Method Details

3.1 Serial Bootloader

The software and protocol for this loader and how to use it are fully described in the CodeWarrior on-line help and web site (see Section 5, “References”). Please refer to the Serial Bootloader application information in the Serial Bootloader User Manual for the processor being used. This manual has detailed information on how to use the Serial Bootloader and how to prepare your application for loading using the Serial Bootloader. Two versions of the Serial Bootloader exist: Version 1.0 and Version 1.1 and beyond. The two versions are similar, except for the baud rate of the SCI communications, as shown in Table 1.

Table 1. Details of Serial Bootloader Versions 1.0 and 1.1

Version	Baud Rate	Flow Control	Word Format
1.0	19200 bps	Xon/Xoff	8 data bits, no parity, 1 stop bit
1.1 and beyond	115200 bps	Xon/Xoff	8 data bits, no parity, 1 stop bit

Version 1.0 of the Serial Bootloader is being phased out and is being replaced with Version 1.1 and beyond. It is possible that you will receive a processor with either Serial Bootloader during the change-over period.

The Version 1.0 Serial Bootloader will send the following message when reset:

```
c) 2000-2001 Motorola Inc. S-Record loader.
```

The Serial Bootloader Version 1.1 and beyond will contain the version number and send the following message when reset:

```
(c) 2000-2001 Motorola Inc. S-Record loader.
```

Figure 1 shows a possible hardware configuration for using the Serial Bootloader to program the device in-circuit. This is one of many possible hardware configurations that will support the use of the Serial Bootloader to program the device.

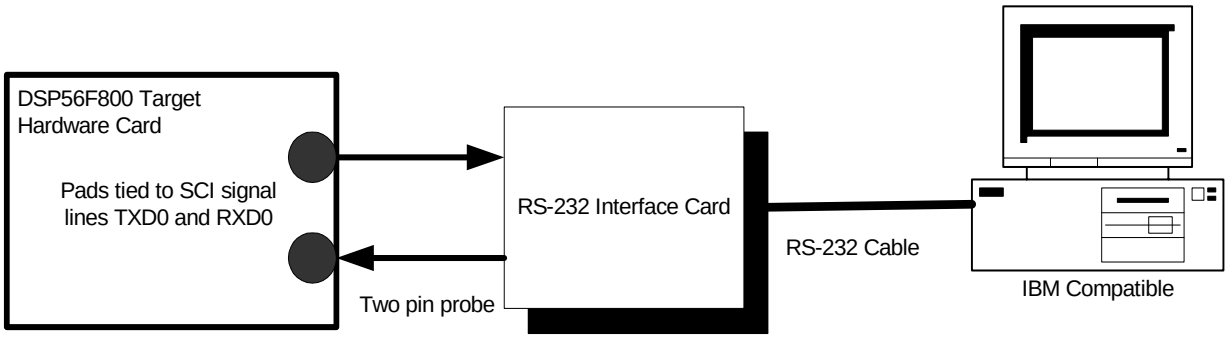

Figure 1. One Hardware Configuration for Serial Bootloader

The configuration in Figure 1 shows how the Serial Bootloader can be used, even though the target hardware has no additional connectors or RS-232 serial drivers. In addition, using an IBM-compatible computer is not required--any host capable of providing the required serial stream and the protocol can be used.

In general, it's easiest to use the Serial Bootloader program by leaving it in place in the Boot Flash without reprogramming the Boot Flash. The Serial Bootloader can be used effectively for field upgrades to the Program and Data Flash. It can also be used to replace the firmware in the Boot Flash, as shown in Figure 2.

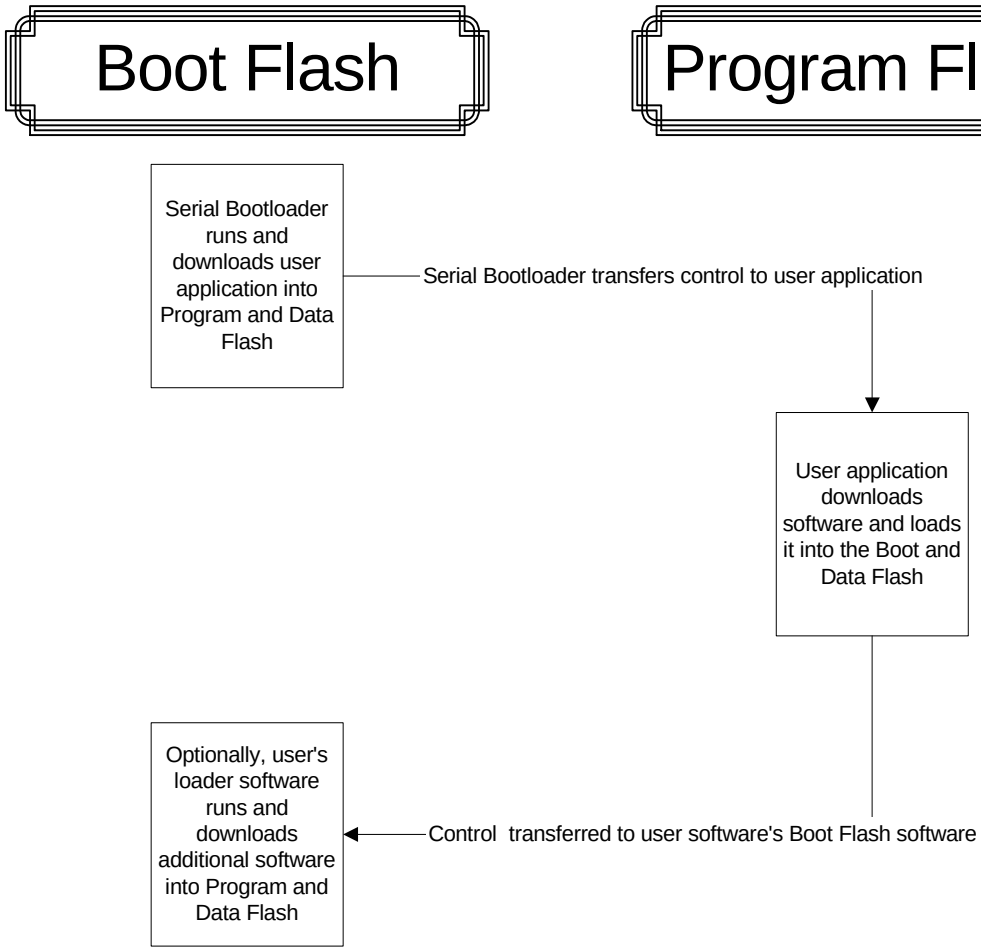

Figure 2. Using the Serial Bootloader to Program Boot Flash

3.2 Bulk Device Loader

In this method, the devices are programmed out-of-circuit in a device loader. The programming can be performed using your own resources, or possibly through your distributor or another value-added reseller. The device programmer can also be used to program the Program, Data, and Boot Flash blocks. The device programmer can be used to load a complete and final application, or simply a custom Bootloader program placed into the Boot Flash area. Later, the final or complete application can be loaded using the custom Bootloader.

At the time of this writing, the 56F800 line of processors is supported with device programmers from BP Microsystems, which can be contacted on the Web at www.bpmicro.com.

3.3 In-circuit JTAG/OnCE Port

To use this method, the user must provide access from his host computer to the JTAG/OnCE port signals on the devices. Table 2 shows the required signals. If the programming set-up uses a standard command converter to interface to the JTAG signals, then a standard JTAG pin header connector must be present on the custom card. An example of this is present on the 56F800 EVM; information on it is also included in the particular EVM user's manual. The EVM user's manual can be downloaded from Freescale's website at: **freescale.com**.

Table 2. Required JTAG/OnCE Signals

Signal	Signal Description	Required
TDI	**Test Data Input**--This input provides a serial data stream to the JTAG and the OnCE module. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor.	Yes
TDO	Test Data Output--This tri-stateable output provides a serial data stream from the JTAG and the OnCE module. It is driven in the Shift-IR and Shift-DR controller states of the JTAG state machine and changes on the falling edge of TCK.	Yes
TCK	**Test Clock Input**--This input provides a gated clock to synchronize the test logic and shift serial data through the JTAG/OnCE port. The maximum frequency for TCK is 1/8 the maximum frequency of the 56F801/803/805/807/826/827 (i.e., 5MHz for TCK if the maximum CLK input is 40MHz). The TCK pin has an on-chip pull-down resistor.	Yes
TMS	Test Mode Select Input--This input sequences the TAP controller's state machine. It is sampled on the rising edge of TCK and has an on-chip pull-up resistor.	Yes
$\overline{\text{TRST}}$	**Test Reset**--This input provides a reset signal to the TAP controller. This pin has an on-chip pull-up resistor.	Yes
$\overline{\text{DE}}$	**Debug Event**--This output signals OnCE events detected on a trigger condition.	No

The CodeWarrior tool can be used to download the program. A Windows application with source code to demonstrate how to program the devices over the JTAG/OnCE port is documented in AN1935. This application is available in the FAQs. Detailed information on how to program using the JTAG/OnCE port is available and can be requested from your Freescale sales representative.

3.4 Parallel Flash Programming Mode

This mode enables the user to directly program all of the Flash blocks that are internal to the 56F800 processors. Programming is performed by placing the chips into a special non-operational mode that brings the control signals required to directly access and program the Flash to the external chip pins. When in this mode, signal definitions for the pins are modified. A total of 49 pins must be driven or read to perform the erase, program, and verify steps that are required.

Clearly, special consideration must be taken with the board design to enable the redefined pins to be driven and read correctly by the tester and not interfered with by circuitry that is tied to the pins and used by the

processor when it is in the normal operating mode. Table 3 shows the pins that are redefined when in the parallel Flash programming modes.

The parallel Flash mode signals are the same as shown in the Flash memory interface section of the appropriate chip's User's Manual. The 56F801 cannot be programmed using this method. The details required to use this method of programming require an NDA with Freescale. Please request this information from your Freescale sales representative.

Table 3. Pins to be Redefined in Parallel Flash Programming Mode

Pin Count 49 Pins Total	Parallel Flash Mode Signal	Chip Signal Name	
		56F803/805/807	56F826/827
1	EXTBOOT		
1	TCS		
1	RESET		
1	TRST		
1	TCK		
1	TMS		
1	TDI		
1	OEN_B	FAULTA2	GPIOB1
16	DIN and DOUT	D0-D15	D0-D15
10	XADR	A5-A14	A5-A15
5	YADR	A0-A4	A0-A4
1	IFREN	TXD0	TXD0
1	XE	PS	PS
1	YE	DS	DS
1	SE	HOME0	TA3
1	OE	RD	RD
1	PROG	WR	WR
1	ERASE	PHASEA0	TA2
1	MAS1	PHASEB0	TA1
1	NVSTR	INDEX0	TA0
1	TMR	RXD0	RXD0

4 Conclusion

The 56F800 components are very flexible in programming Flash blocks. In this application note, several methods have been presented for programming the Flash blocks in a production environment. One of these methods, or a variation of it, should meet your production Flash programming requirements.

5 References

The following materials were used to produce this paper:

- *CodeWarrior IDE Targeting DSP56800 Manual*
- *56F801 Evaluation Module Hardware User's Manual*, DSP56F801EVMUM
- *56F803 Evaluation Module Hardware User's Manual*, DSP56F803EVMUM
- *56F805 Evaluation Module Hardware User's Manual*, DSP56F805EVMUM
- *56F807 Evaluation Module Hardware User's Manual*, DSP56F807EVMUM
- *56F826 Evaluation Module Hardware User's Manual*, DSP56F826EVMUM

Appendix A S-Record Specification

1. Use only S0, S3 and S7 records.
2. Three S-Record files are built with every compile. The first contains the contents of all initialized PROGRAM (P) Memory contents. The second file contains the contents of all initialized DATA (X) Memory contents.
3. The third S-Record file (the “combined” file) is roughly a concatenation of the P file with the X file (P followed by X). The S3 data records targeted for X produced by the linker have a word offset of 0x00200000 added to their word address fields.

   It is also possible to set the linker options in the IDE to generate byte addresses. In this case, the S3 data records targeted for X produced by the linker have a byte offset of 0x00400000 added to their byte address fields.

   There shall be only a single pair of S0 and S7 records for the entire combined file.

   NOTE

   — The P (and combined) file may contain: Program Flash contents, Program RAM contents, Boot Flash contents.

   — The X (and combined) file may contain: Data Flash contents, Data Ram contents.

   — The device programming algorithm shall be address range aware.

   – The byte address fields of S3 data records are examined by the device programming algorithm to determine which Flash Interface Unit (FIU) to access.

   – The word offset of 0x00200000 (or byte offset of 0x00400000) for X data records is required because the P and X memories within the 56F80x devices both start at address 0. (S-Records don't normally support overlapping memories; defining an offset is the most expedient work-around for the problem.).

   – Data specified for RAM locations shall be ignored by the device programming algorithm.

   — A contiguous block of Flash (for example, Program Flash on the 827chip) may actually be comprised of two separate Flash memories that must be accessed through separate FIUs. The Program address space, however, is presented to the user, and within the S-Record file, as a single contiguous address range. Again, the S3 record's address field is examined to determine which FIU to access.
4. Unique codes shall reside in the P, X and combined S0 header records. In this manner, the S0 records will identify the subsequent contents as either P, or X, or combined. The proposed S0 header records are as follows:

Example A-1. S0 Record for the P File

```
S0 0C 00000000 50 52 4F 47 52 41 4D DB
(Hex ASCII for: P  R  O  G  R  A  M )
```

Example A-2. S0 Record for the X File

```
S0 09 00000000 44 41 54 41 DC
(Hex ASCII for: D  A  T  A )
```

Example A-3. S0 Record for the Combined File

```
S0 11 00000000 50 52 4F 47 52 41 4D 26 44 41 54 41 96
```
(Hex ASCII for: P R O G R A M & D A T A)

NOTE

The device programming algorithm will not be expected to recognize contents within any S0 record. The S0 content is only to aid with human recognition of file contents and to assist in content recognition by future tools.

5. S3 data shall not split across a processor word size (16 bits). In other words, there shall be an even number of “data” bytes within each S3 record.

   Sample S-Record files for the 56F800 follow. (Spaces have been inserted for improved readability in this document; “CS” is a space holder for the checksum byte.) The address fields of the S3 record shall be conventional BYTE addresses and byte data shall be represented in the Little Endian format. (The least significant byte is located at the lowest address.)

NOTE

— In this manner, the constituent P or X S-Record files can be used to program any conventional (external) non-volatile memory devices, without the need for any additional S-Record manipulation utility programs

— 56F800 processors have a native 16 bit word, and do not support byte addressing. However, the CW IDE Linker provides the option to generate S-Records with Byte or Word addressing. Likewise, the FIUs are programmed with 16 bit words and all addresses are interpreted as word addresses.

— When using Byte addresses, to get from the S-Record Byte address to the 56F800 16 bit Word address, you'll need to divide the S-Record address field value by 2. For X memory, you must first subtract the 0x00400000 offset. You must also unscramble the Little Endian format to recognize the data words.

— Some programming suppliers require a different offset than what the compiler produces in the case of the combined records. Check with your vendor prior to sending files to see if any conversion of the S-Records offset in the combined file is required.

Example A-4. Sample P File

```
S0 0C 00000000 50 52 4F 47 52 41 4D DB          "PROGRAM"
S3 0D 00000000 10 32 11 32 12 32 13 32 CS
S3 0D 00000008 14 32 15 32 16 32 17 32 CS
S3 0D 00000010 18 32 19 32 1A 32 1B 32 CS
S3 0D 00010000 10 B2 11 B2 12 B2 13 B2 CS       Boot Flash starts at word
S3 0D 00010008 14 B2 15 B2 16 B2 17 B2 CS       address 8000,56F801/3/5/7
S7 05 00000000 CS                               (byte address 10000)
```

Example A-5. Sample X File

```
S0 09 00000000 44 41 54 41 DC                   "DATA"
S3 0D 00002000 10 A2 11 A2 12 A2 13 A2 CS       Note: There is no offset
S3 0D 00002008 14 A2 15 A2 16 A2 17 A2 CS       used in the stand alone
S7 05 00000000 CS                               X S-Record file.
                                                In this example, word
```

address 1000 is the first location to be programmed.

Example A-6. Sample Combined File

```
S0 11 00000000 50 52 4F 47 52 41 4D 26 44 41 54 41 96     "PROGRAM
S3 0D 00000000 10 32 11 32 12 32 13 32 CS                 & DATA"
S3 0D 00000008 14 32 15 32 16 32 17 32 CS
S3 0D 00000010 18 32 19 32 1A 32 1B 32 CS
S3 0D 00010000 10 B2 11 B2 12 B2 13 B2 CS                 Boot Flash starts at word
S3 0D 00010008 14 B2 15 B2 16 B2 17 B2 CS                 address 8000,
                                                          56F801/803/805/807
S3 0D 00402000 10 A2 11 A2 12 A2 13 A2 CS                 Big offset of 0x00400000
S3 0D 00402008 14 A2 15 A2 16 A2 17 A2 CS                 signifies X data follows
S7 05 00000000 CS
```

S-Record Explanation

S0 05 00000000 (data) CS

S0	S0 is a header record S3 is data S7 terminator for block of S3
05	(hex) 5 bytes follow
00000000	32 bits of byte addresses
CS	The least significant byte of the one's complement of the sum of the values represented in the pairs of characters making up the record length, address, and the data fields

After programming the 56F800 with either the combined file or both stand-alone P and X files, a debugger dump would yield results like those in the following Code Examples:

Example A-7. Dump of Program (P) Flash

```
address p:0 = 3210
address p:1 = 3211
address p:2 = 3212
address p:3 = 3213
address p:4 = 3214...
```

Example A-8. Dump of Boot Flash

```
address 0 = B210
address 1 = B211
address 2 = B212
address 3 = B213...
```

Example A-9. Dump of Data (X) Flash

```
address x:1000 = A210
address x:1001 = A211
address x:1002 = A212
address x:1003 = A213
```

```
address x:1004 = A214...
```

THIS PAGE IS INTENTIONALLY BLANK

How to Reach Us:

Home Page:
www.freescale.com

Web Support:
http://www.freescale.com/support

USA/Europe or Locations Not Listed:
Freescale Semiconductor, Inc.
Technical Information Center, EL516
2100 East Elliot Road
Tempe, Arizona 85284
+1-800-521-6274 or +1-480-768-2130
www.freescale.com/support

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)
www.freescale.com/support

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 China Ltd.
Exchange Building 23F
No. 118 Jianguo Road
Chaoyang District
Beijing 100022
China
+86 10 5879 8000
support.asia@freescale.com

For Literature Requests Only:
Freescale Semiconductor Literature Distribution Center
1-800-441-2447 or 303-675-2140
Fax: 303-675-2150
LDCForFreescaleSemiconductor@hibbertgroup.com

Document Number: AN1926
Rev. 3
03/2010

freescale™
semiconductor