Freescale Semiconductor
Application Note

AN2546
Rev. 1, 12/2004

HCS12 Load RAM and Execute Bootloader User Guide

by: Martyn Gallop
Joanne McNamee

Introduction

This User Guide covers the operation and use of a Load RAM And Execute (LRAE) bootloader for the HCS12 microcontroller family from Freescale Semiconductor.

LRAE Bootloader Overview

An LRAE bootloader can be a convenient way to support programming during production or “in-system”, where support for the dedicated HCS12 Background Debug interface (BDM) may not be available. Users must pre-program the HCS12 with the bootloader during pre-production or at a programming vendor. The bootloader was written so that, if required, the bootloader can be kept resident in the MCU for further use.

This bootloader implementation allows user software to be downloaded into the RAM of the MCU using the CAN or SCI serial interfaces. The bootloader polls the CAN and SCI ports for messages. When a message is received, the bootloader attempts to match the incoming communication baud rate against a number of selected baud rates based on common crystal frequencies. After the user software has been downloaded into RAM, execution is transferred to the code resident in RAM.

The bootloader software was developed using a modular approach allowing it to be easily modified to support different HCS12 devices and individual programming requirements as required. The LRAE bootloader was initially evaluated for the following Freescale HCS12 devices:

© Freescale Semiconductor, Inc., 2004. All rights reserved.

D32, D64, Dx128, Dx256, DP512, A32, A64, A128, A256, A512, C32, C128, E128, and B128. This was primarily an exercise in resource allocation as can be seen in the Memory Mapping Section.

Acronyms and Abbreviations

CAN	controller area network
LRAE	load RAM and execute
LSB	least significant byte
MCU	microcontroller unit
MSB	most significant byte
RAM	random access memory
SCI	serial communications interface
UART	universal asynchronous receiver transmitter
Tq	CAN time quanta

Functional Description

Operation

By default, the bootloader assumes control of the MCU following a power-on reset; the reset vector is programmed with the address of the bootloader code.

First, the bootloader checks the value of the FLASH word at address $C000–$C001. If this word is erased, the bootloader starts to execute and poll the CAN and SCI ports for messages. If, however, this word is programmed, the bootloader performs a jump to the address contained in the word, and execution of the application program begins from that location.

If you do not require the bootloader functionality, you must erase the FLASH memory area containing the bootloader code and the reset vector before attempting to reprogram the FLASH. By default, FLASH protection is not enabled for the bootloader code.

Application Start Vector

If you wish to retain the bootloader for use in your application, you must program the FLASH word at address $C000–$C001 with the start address of the application program.

With the bootloader resident, and an application programmed into the FLASH, the effective application startup vector is the address programmed into $C000–$C001.

Memory Map

The bootloader is resident in FLASH page $3E. The bootloader start address is $4000. By default, the MCU reset vector ($FFFE–$FFFF) is programmed with this address.

In some cases, you may wish to reprogram page $3F of the MCU FLASH memory, erasing the MCU vectors. In order to retain the bootloader functionality, you must reprogram the MCU power-on reset vector with the bootloader start address ($4000).

If the application start vector ($C000–$C001) is not programmed, the bootloader remaps the device memory resources before starting the LRAE download.

The RAM on different HCS12 MCUs is of differing sizes and, by default, maps to either the top or the bottom of the lower 16K bytes of the memory map ($0000–$3FFF).

The bootloader remaps the RAM block to a location, as defined in Table 1. LRAE Bootloader RAM Mapping, by initializing the INITRM register with the value 0x39.

The register block remains in the default position, starting at $0000 (0x00 is written to INITRG).

On devices with user EEPROM, the EEPROM is remapped to start at $0800 (0x09 is written to INITEE).

Table 1. LRAE Bootloader RAM Mapping

Device	RAM Size	RAM Mapping	RAM for Download
D32	2K	$3800-$3FFF	$3800-$3FCF
D64	4K	$3000-$3FFF	$3000-$3FCF
D128	8K	$2000-$3FFF	$2000-$3FCF
D256	12K	$1000-$3FFF	$1000-$3FCF
D512	14K	$0800-$3FFF	$0800-$3FCF
A32	2K	$3800-$3FFF	$3800-$3FCF
A64	4K	$3000-$3FFF	$3000-$3FCF
A128	8K	$2000-$3FFF	$2000-$3FCF
A256	12K	$1000-$3FFF	$1000-$3FCF
A512	14K	$0800-$3FFF	$0800-$3FCF
C32	2K	$3800-$3FFF	$3800-$3FCF
C128	4K	$3000-$3FFF	$3000-$3FCF
E128	8K	$2000-$3FFF	$2000-$3FCF
B128	4K	$3000-$3FFF	$3000-$3FCF

The memory range $3FD0-$3FFF is reserved for the bootloader stack and variables. All other RAM, as defined in Table 1. LRAE Bootloader RAM Mapping, is available for the downloaded code.

After execution has been transferred to the downloaded code, the bootloader is redundant and the bootloader reserved RAM area can be utilized as data RAM by the downloaded code.

Table 2. LRAE Bootloader Code Mapping shows the FLASH address range reserved for the bootloader on individual devices.

Table 2. LRAE Bootloader Code Mapping

Device	FLASH Address
Dx512	$4000-$4400
Dx256 - 2 CAN	$4000-$4400
Dx256 - 3 CAN	$4000-$4400
Dx128 - 2 CAN	$4000-$4400
Dx128 - 3 CAN	$4000-$4400
Dx64	$4000-$4400
D32	$4000-$4400
A512	$4000-$4200
A256	$4000-$4200
A128	$4000-$4200
A64	$4000-$4200
A32	$4000-$4200
B128	$4000-$4400
C32	$4000-$4400
C128	$4000-$4400
E128	$4000-$4200

Communication Speeds

The bootloader attempts to match a selection of incoming baud rates on the CAN and SCI, taking into account various oscillator frequencies. The bootloader is designed to function with oscillator frequencies of 4 MHz, 8 MHz and 16 MHz. The tables below give the possible CAN and SCI bus speeds that the bootloader attempts to match at each oscillator frequency.

For the CAN bus, the receive and transmit buffers use the standard ID length (11 bits).

Each CAN bit rate option is selected for ~42,000 bus clock cycles (~5 ms @ 16 MHz).

Table 3. CAN Port Configuration Values

Configuration	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11
Prescaler	1	1	2	4	8	4	8	16	5	10	20
Tseg1	5	13	13	13	13	9	9	9	13	13	13
Sync seg	1	1	1	1	1	1	1	1	1	1	1
Tseg2	2	2	2	2	2	2	2	2	2	2	2
Jump width	2	2	2	2	2	2	2	2	2	2	2
Tq/bit	8	16	16	16	16	12	12	12	16	16	16

Table 4. CAN Configuration Options

	Bit Rates				
Osc. Frequency	**1 Mb/s**	**500 kb/s**	**125 kb/s**	**83.3 kb/s**	**50 kb/s**
16 MHz	#2	# 3	#5	#8	#11
8 MHz	#1	#2	#4	#7	#10
4 MHz		#1	#3	#6	#9

For the SCI bus, the bootloader attempts communication using the prescaler values listed in Table 5. SCI Port Baud Rates. These SCI baud rates are chosen as closest fit to standard baud rates, considering the HCS12 prescaler implementation.

The actual HCS12 baud rate = SCI module clock frequency / (Prescaler * 16)

where the SCI module clock frequency = Oscillator frequency / 2.

The SCI communications timing should have a total error of < 3.9%. Where this is achievable using a common UART baud rate, this is shown in brackets. A host may support the higher baud rates, but not using common UART communications settings.

The bootloader attempts to receive the synchronization byte twice at each baud rate before selecting the next baud rate.

For the SCI bus, the data format is one start bit, eight data bits, one stop bit, no parity.

It is up to the user to ensure that robust systems communication can be achieved at any selected frequency.

Table 5. SCI Port Baud Rates

Prescaler Option	Prescaler Value	Osc (MHz)		
		4	8	16
#1	1	125000	250000	500000
#2	2	62500	125000	250000
#3	4	31250	62500	125000
#4	9	13889 (14400)	27778 (28800)	55556 (57600)
#5	7	17857	35714	71429
#6	13	9615 (9600)	19231 (19200)	38462 (38400)
#7	26	4808 (4800)	9615 (9600)	19231 (19200)
#8	52	2404 (2400)	4808 (4800)	9615 (9600)

Communication Modules Used

The bootloader attempts to communicate with multiple CAN and SCI ports, where they are available on a specific MCU. The communications modules supported are CAN0, CAN1, CAN4, SCI0, and SCI1.

All communication modules use the default I/O assignment. The bootloader does not modify the module rerouting register value.

Hardware Compatibility

The bootloader attempts to avoid contention with existing application hardware. I/O pins on the HCS12 default to input (some with internal pull-devices enabled).

Both the CAN and the SCI bus port pins are configured to operate in normal (push-pull) mode, full-drive, with no internal pull devices enabled.

For the SCI bus, only the receiver is enabled initially. The SCI transmit pin in only enabled as an output when a synchronization byte has been received from a host.

Before attempting to initialize a CAN module, the bootloader initialization tests to confirm that:

1. the Tx pin is pulled up to 5 V,
2. the Tx pin can be driven low,
3. this low state on Tx is echoed on the Rx pin (after ~3 µs, with a 16 MHz oscillator)

If any of these tests fail, the CAN module is not initialized, and the associated pins remain as inputs.

Bootloader Top Level Flow

Start:

Jump to application in FLASH if $C000-$C001 is programmed (with the FLASH application start address)

Configure MCU resources

LRAE Initialize:

- For each CAN module:
  - If CAN physical interface detected
    - Configure for initial bit rate
  - If CAN module synchronizes to bus (= active)
    - CAN enters sleep mode
- For each SCI module:
  - Configure for first SCI baud rate
  - Enable Rx

Main Loop:

- For each active Can module
  - If the CAN module is awake
    - If Address Pointer message has been received
      - Complete download from Can module and jump to downloaded code
    - Else If data not received and timeout completed
      - Change module to next bit rate
    - Else If data not received and timeout not completed
      - Increment module timeout counter
- For each SCI module
  - If data byte has been received
    - If data is valid synchronization byte
      - Transmit acknowledge byte
      - Complete download from SCI and jump to downloaded code
    - Else If data byte is not the synchronization byte (for the second time)
      - Change module to next baud rate

Loop back to Main Loop

CAN Communication Structure

The CAN bus protocol includes message acknowledgement. Therefore, the host machine can simply follow a program flow similar to that shown in Figure 1. CAN Communications Flow, checking that there were no CAN errors before proceeding. The bootloader supports the following commands sent via the CAN bus.

Command	CAN ID	Direction
Set address pointer	0x020	HOST -> MCU
This command contains the start address of where to load data into RAM and will set the download address pointer. The LRAE checksum value is initialized on receipt of this message. The download process is initiated on receipt of the first instance of this command. **Data Parameters**		
Data Byte 1	Address MSB	0x00–0xFF
Data Byte 2	Address LSB	0x00–0xFF

Command	CAN ID	Direction
Load data	0x040	HOST -> MCU
This command will transfer up to 8 bytes of data into RAM and increment address pointer. **Data Parameters**		
Data Byte 1–8	Program data	0x00–0xFF

Command	CAN ID	Direction
Execute from address	0x010	HOST -> MCU

This message contains the address in RAM where the bootloader should start execution of the downloaded code.

Data Parameters

Data Byte 1	Address MSB	0x00–0xFF	
Data Byte 2	Address LSB	0x00–0xFF	

Command	CAN ID	Direction
Checksum Value Message	0x080	HOST -> MCU

This message contains the host calculated checksum value.

Data Parameters

Data Byte 1	Checksum Value	0x00–0xFF	Sum of the values of all bytes from the set address pointer up to the last data byte, inclusive, modulo 256.

Command	CAN ID	Direction
Checksum Status Message	0x600	MCU -> HOST
This response message contains a target checksum status flag and value. **Data Parameters**		
Data Byte 1	Checksum Status	0x80 0x01	Checksum correct. Checksum error.
Data Byte 2	LRAE Checksum value	0x00–0xFF	Sum of the values of all bytes from the set address pointer up to the last data byte, inclusive, modulo 256.

Using the above CAN messages, the host must send CAN messages to the MCU, as outlined in the flowchart in Figure 1. CAN Communications Flow.

First, the host is required to set the address pointer. The bootloader will not synchronize the LRAE bit rate to a CAN bus until it detects a valid Set Address Pointer message.

The program data should then be sent, followed by the checksum. The MCU returns a checksum acknowledge to the host. This tells the host whether or not there was a checksum error.

Because the LRAE checksum is initialized each time a Set Address Pointer message is received, re-send of data or non-contiguous data may be supported by a host implementation.

When all data is downloaded successfully, the host can send an execute command message to start code execution.

Can Communication — Basic Host

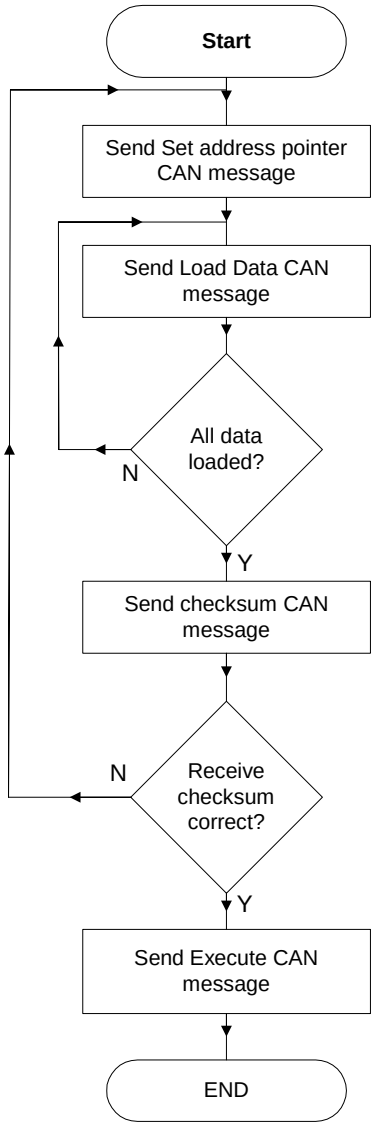

Figure 1. CAN Communications Flow

SCI Communication

The SCI message communication from the host to the target MCU is outlined in the flowchart in Figure 2. SCI Communications Flow Chart. To match the baud rate, the host must:

1. Transmit the synchronization message ($55)
2. Wait 10 bit times for the $55 to transmit + ~1ms delay + 10 bit times to allow reception of an $AA response
3. Repeat until a synchronization acknowledge message ($AA) is received

Once the synchronization acknowledge byte has been received, the MCU has established the correct communication speed.

The host software must then send the messages to set the address pointer of the MCU, followed by the messages giving the number of data bytes that will be transmitted. After this, the program data can be sent to the MCU.

The bootloader then requires a checksum to be sent by the host. The checksum is the sum (modulo 256) of all program data from address pointer to last data byte, inclusive. If the checksum validates correctly, the MCU sends back a checksum acknowledge byte ($80), and execution of the code downloaded into RAM begins. If the checksum does not validate correctly, the bootloader halts and waits for an external reset to be applied to the MCU. The host should implement a timeout following the checksum request.

SCI Communication Structure — Host

The flow of the host-to-SCI communications is outlined in the flowchart in Figure 2. SCI Communications Flow Chart.

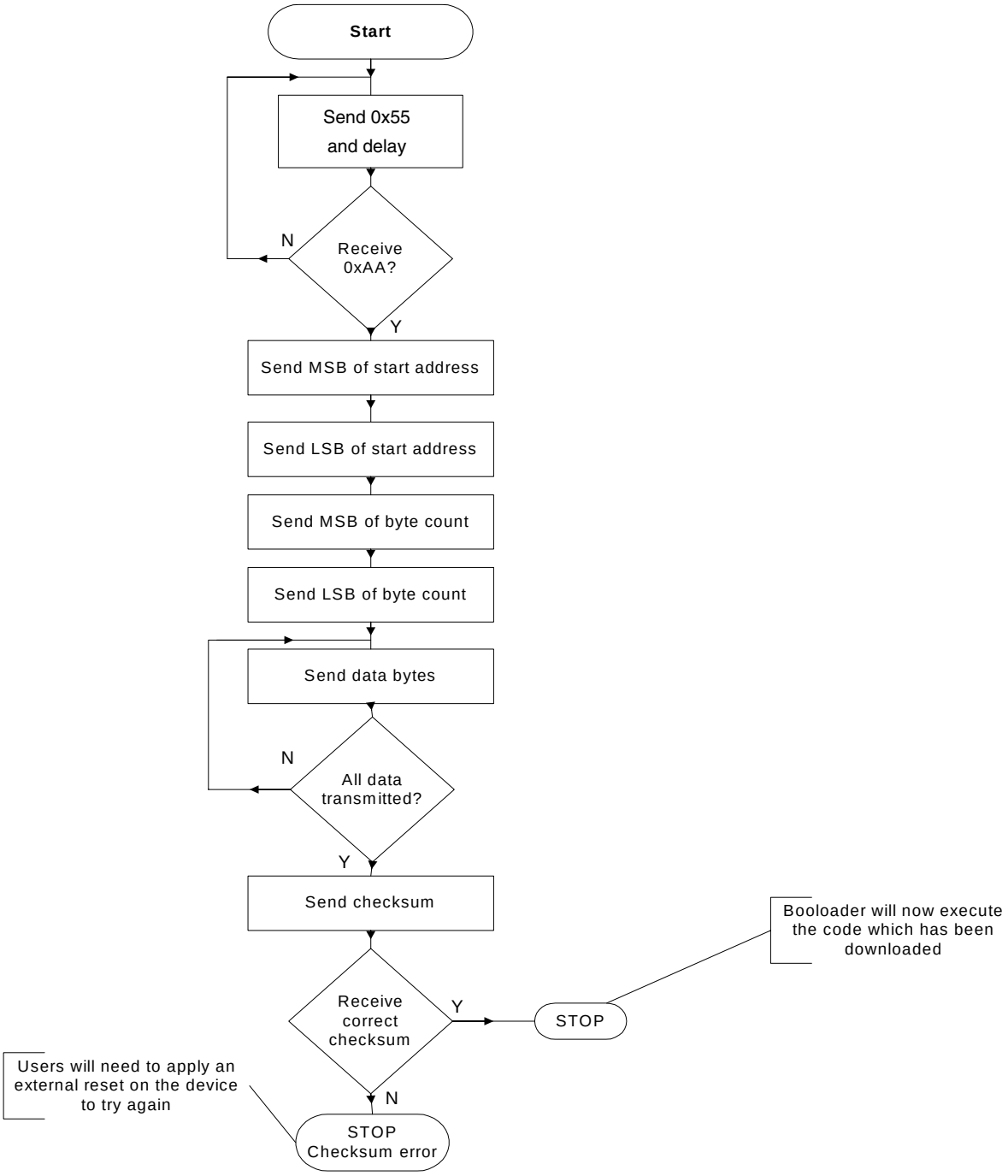

Figure 2. SCI Communications Flow Chart

Constraints and Considerations

The bootloader, by its generic nature, has no knowledge of each user's application. This results in certain compromises and limitations of which you should be aware.

1. The bootloader does not contain FLASH memory programming routines. An HCS12 FLASH programming driver for embedded applications and BDM programming tools, "HCS12 SGF NVM Standard Software Driver" (HCS12SGF25NVMSSD), is available from Freescale's Semiconductor web page at http://www.freescale.com. To locate this, go to Support / Design Tools and select Microcontrollers then Device Drivers. This may be useful for developing a FLASH programming application that can be downloaded and executed via the bootloader.
2. The LREA bootloader does not change the state of any I/O following reset, other than that associated with the appropriate CAN and SCI peripherals. Consideration should be given during design to which ports have default internal pull devices and which default to high impedance inputs in order to evaluate the compatibility of the default state with any devices being driven.
3. The bootloader does not enable the COP watchdog timer. If the downloaded code requires the COP function, then it must enable it.
4. The bootloader provides no support for external watchdog devices. These should either be designed in with a disable capability, or should have a long enough timeout following reset to allow for the code download to complete and to start servicing the watchdog. The download time will be dependent on the specific synchronization time, code size and data rate.
5. Protection is not enabled for the section of FLASH containing the bootloader. When modifying the FLASH memory, care must be taken not to modify the bootloader unintentionally.
6. The bootloader does not use the PLL and therefore the higher SCI baud rates are constrained as explained in the Communication Speeds section. Depending on the host implementation, once the code download to RAM is completed it may be possible for the code in RAM to enable the PLL and switch to a higher baud rate (derived from the higher bus speed). This will be dependent on the host implementation.
7. There is always a risk when programming an application into FLASH memory using downloaded programming algorithms, that an external disturbance may corrupt the programming process.

   It may not be possible to recover from this in all cases. Certain considerations can help minimize the risk of corruption. The higher risk operations are those that modify the following:

   a. The MCU reset vector. While the reset vector is erased, a reset will not restart the bootloader. If the bootloader is to remain resident, then erasing and programming the reset vector sequentially will minimize the window during which a reset could cause the bootloader function to become corrupted.
   b. The application start vector. With the bootloader resident, once the application start vector is programmed, the bootloader LRAE function following reset is disabled. If a premature reset or a FLASH write issue occurs, a reset will then cause a jump to an incomplete application. Programming the application start vector as the last operation can minimize this risk.
   c. The bootloader code. If, for some reason, it is necessary to erase the bootloader code from FLASH, a premature reset or FLASH programming problem will fail to successfully restart the LRAE process.

In any of these three cases, recovery can be facilitated by designing easy access to the signals required for interfacing a BDM interface cable to the MCU.

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

Information in this document is provided solely to enable system and software implementers to use Freescale Semiconductor products. There are no express or implied copyright licenses granted hereunder to design or fabricate any integrated circuits or integrated circuits based on the information in this document.

Freescale Semiconductor reserves the right to make changes without further notice to any products herein. Freescale Semiconductor makes no warranty, representation or guarantee regarding the suitability of its products for any particular purpose, nor does Freescale Semiconductor assume any liability arising out of the application or use of any product or circuit, and specifically disclaims any and all liability, including without limitation consequential or incidental damages. “Typical” parameters that may be provided in Freescale Semiconductor data sheets and/or specifications can and do vary in different applications and actual performance may vary over time. All operating parameters, including “Typicals”, must be validated for each customer application by customer’s technical experts. Freescale Semiconductor does not convey any license under its patent rights nor the rights of others. Freescale Semiconductor products are not designed, intended, or authorized for use as components in systems intended for surgical implant into the body, or other applications intended to support or sustain life, or for any other application in which the failure of the Freescale Semiconductor product could create a situation where personal injury or death may occur. Should Buyer purchase or use Freescale Semiconductor products for any such unintended or unauthorized application, Buyer shall indemnify and hold Freescale Semiconductor and its officers, employees, subsidiaries, affiliates, and distributors harmless against all claims, costs, damages, and expenses, and reasonable attorney fees arising out of, directly or indirectly, any claim of personal injury or death associated with such unintended or unauthorized use, even if such claim alleges that Freescale Semiconductor was negligent regarding the design or manufacture of the part.

Freescale™ and the Freescale logo are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners.

© Freescale Semiconductor, Inc. 2004. All rights reserved.

AN2546
Rev. 1, 12/2004

freescale™ semiconductor