

Using the HCS12 NVM Standard Software Drivers with the Cosmic Compiler

By **Gordon Doughman**
Field Applications Engineer
Software Specialist

Introduction

The NVM Standard Software Drivers (SSD) for the HCS12 Family of microcontrollers provide position independent, ROM-able software to allow erasure and programming of any member of the HCS12 Family containing 0.25 μm Split-Gate Flash (SGF) Non-Volatile Memory (NVM) technology. While the SSD software was developed using the C programming language, it is only distributed in position independent binary form. Rather than distribute the software as a binary object file, which is specific to a particular compiler tool set, it is distributed as a C-data array for embedded applications and S-Record format for use with BDM programming tools. Theoretically, this should allow the drivers to be used with any compiler tool set.

Unfortunately, the tool set used to compile the SSD software, the Metrowerks Codewarrior tools, places function arguments on the stack using the Pascal calling convention. In this convention, the caller pushes the arguments on the stack from left to right rather than using the standard C calling convention which pushes function arguments on the stack from right to left. In addition, the Codewarrior calling convention passes the last or only function argument in the CPU registers. While this poses no problem when the SSD software is used with the Codewarrior toolset, it prevents direct use with other compiler toolsets such

Callback Functions

as the Cosmic compiler for the HCS12 Family. Fortunately, because the Cosmic compiler also passes the last or only function argument in the same CPU registers as the Codewarrior tools, compatibility with the SSD software can be achieved by simply modifying one of the header files distributed with the driver software.

Because the *HCS12 SGF NVM Standard Software Driver User's Manual* provides complete, detailed information on the integration, use and troubleshooting of the SSD functions this application note will simply supplement the information contained in *Section 3, API Specification*, of that document.

Callback Functions

Many of the SSD functions support concurrency in a polled environment through the use of a Callback mechanism. To keep the SSD functions from monopolizing the CPU for extended periods of time a user supplied function pointer is passed as one of the function parameters. This function is then periodically called during the execution of the SSD function. However, because there is no way to disabled the Callback mechanism, a pointer to an empty function must be passed in the Callback parameter even when concurrency in a polled environment is not required.

Because the Callback function pointer is limited to a 16-bit value, the Callback function must reside within the S12's 64K memory space. This restricts its placement to one of the two fixed Flash pages, EEPROM or RAM. For those applications not performing program or erase operations on Flash block zero, the Callback function may conveniently be placed on either the upper or lower fixed Flash pages. However, for S12 devices containing only a single Flash block or for applications such as bootloaders, which require at least a portion of Flash block zero to be erased and reprogrammed, the Callback function must be placed in EEPROM or RAM.

For SSD driver applications not requiring concurrency in a polled environment, a single RTS instruction (0x3d) can be placed into a char variable and the address of this variable can be passed to the SSD function as the address of the Callback function.

Compiler Command Line Option

By default, the Cosmic compiler automatically widens parameters of type char to short before being pushed onto the stack. Because the Metrowerks compiler does not perform this automatic widening operation, the Cosmic compiler's **+nowiden** command line option must be used with all source files in projects using the SSD software.

Revised Standard Software Drivers API

The header file, *ssd_sgf.h*, included with the SSD software distribution contains the function prototypes for each of the SSD implemented NVM functions. Each of these function prototypes must be modified to reverse the order of the declared parameters. Presented in each of the following sections is the revised function prototype and parameter description for each of the functions available in the SSD package.

NVMInit()

This function sets up the NVM module clock according to the oscillator clock and the bus clock. It also initializes the control registers for Flash and EEPROM. *NVMInit* must be called prior to program or erase operations to ensure that both the Flash and EEPROM module clock dividers are correctly configured.

NOTE

This function returns bit mapped error codes so more than one error condition can be communicated with a single return code.

The NVM module's clocks must be properly synchronized with the oscillator clock and the bus clock. *NVMInit* ensures that all of the clock synchronization requirements are met. Consult one of the NVM module specifications for more information regarding the NVM module clock requirements.

```
UINT16 NVMInit(UINT16 funcPtr,
               UINT16 pDescriptor,
               UINT16 regBase,
               BOOL BDMEnable,
               UINT16 oscClock)
```

Figure 1. NVMInit() Function Prototype

Table 1. NVMInit() Parameter Description

Argument	Type	Description	Range
funcPtr	UINT16	Function pointer to <i>RangeCheck</i> .	If this value is 0xFFFF the RangeCheck function is not called. In this case memory resource intersections are the user's responsibilities. Otherwise, the <i>RangeCheck</i> function is called to verify that the requested NVM operation is consistent with the user-defined memory map.
pDescriptor	UINT16	Pointer to the SSD configuration descriptor.	Any address within the MCU address space (0x0000 – 0xFFFF).
regBase	UINT16	Base address of the register block.	This value should be on a 2K byte boundary within the first 32K bytes of the 64K byte MCU address space.
BDMEnable	BOOL	Select alternate function returns.	FALSE selects a normal function return; TRUE executes the BGND instruction at the function exit.
oscClock	UINT16	Target oscillator clock	One <i>oscClock</i> equals 10 kHz

NVMMass Erase()

This function automatically erases multiple NVM blocks. Both Flash and EEPROM can be erased during the same function call.

```

UINT16 NVMMassErase(UINT16 funcPtr,
                    UINT16 regBase,
                    void (*CallBack)(void)
                    BOOL BDMEnable,
                    BOOL ISREnable
                    UINT16 blocks);

```

Figure 2. NVMMassErase() Function Prototype

Table 2. MVMMassErase() Parameter Description

Argument	Type	Description	Range
funcPtr	UINT16	Function pointer to <i>RangeCheck</i> .	If this value is 0xFFFF the RangeCheck function is not called. In this case memory resource intersections are the user's responsibilities. Otherwise, the <i>RangeCheck</i> function is called to verify that the requested NVM operation is consistent with the user-defined memory map.
regBase	UINT16	Base address of the register block.	This value should be on a 2K byte boundary within the first 32K bytes of the 64K byte MCU address space.
CallBack	void(*) (void)	Address of void callback function pointer	Any valid void function address
BDMEnable	BOOL	Select alternate function returns.	FALSE selects a normal function return; TRUE executes the BGND instruction at the function exit.
ISREnable	BOOL	Select Interrupt or polling logic.	TRUE selects interrupt logic. FALSE selects polling logic
blocks	UINT16	A bit-mapped argument defining the NVM regions to be erased. if <i>blocks</i> is 0x0000, the return code is SGF_OK.	The LSB selects EEPROM. Higher order bits select Flash.

NVMSectorErase()

This function automatically erases multiple four byte EEPROM sectors given a starting address and the number of sectors to erase. This function also automatically erases multiple 512 byte Flash sectors given a starting address and the number of sectors to erase. Only one type of NVM – either Flash or EEPROM – can be erased during a single call.

```
UINT16 NVMSectorErase (UINT16 funcPtr,
                      UINT16 regBase,
                      void (*CallBack)(void),
                      BOOL BDMEnable,
                      BOOL ISREnable,
                      UINT16 number,
                      UINT32 dest);
```

Figure 3. NVMSectorErase() Function Prototype

Table 3. NVMSectorErase() Parameter Description

Argument	Type	Description	Range
funcPtr	UINT16	Function pointer to <i>RangeCheck</i> .	If this value is 0xFFFF the <i>RangeCheck</i> function is not called. In this case memory resource intersections are the user's responsibilities. Otherwise, the <i>RangeCheck</i> function is called to verify that the requested NVM operation is consistent with the user-defined memory map.
regBase	UINT16	Base address of the register block.	This value should be on a 2K byte boundary within the first 32K bytes of the 64K byte MCU address Space.
CallBack	void(*) (void)	Address of void callback function pointer	Any valid void function address
BDMEnable	BOOL	Select alternate function returns.	FALSE selects a normal function return; TRUE executes the BGND instruction at the function exit.
ISREnable	BOOL	Select Interrupt or polling logic.	TRUE selects interrupt logic. FALSE selects polling logic
number	UINT16	The number of sectors to be erased. If <i>number</i> is zero, the return value is SGF_OK.	The <i>dest</i> and <i>number</i> parameters define a continuous sequence of NVM sectors that must lie entirely within either Flash or EEPROM but not both.
dest	UINT32	The first sector address.	<ol style="list-style-type: none"> Any location within MCU EEPROM address space or the virtual Flash memory. For Flash, the lowest nine bits are masked. For EEPROM, the lowest two bits are masked.

BlankCheck()

This function reads the memory range and checks for the erased state (0xFFFF). The scope of this function is either Flash or EEPROM, but only one NVM region can be checked per function call.

```
UINT16 BlankCheck(UINT16 funcPtr,
                  UINT16 regBase,
                  void (*CallBack)(void),
                  BOOL BDMEnable
                  UINT16 *compareData,
                  UNIT32 *compareAddress,
                  UNIT32 size,
                  UINT32 dest);
```

Figure 4. BlankCheck() Function Prototype

Table 4. BlankCheck() Parameter Description

Argument	Type	Description	Range
funcPtr	UINT16	Function pointer to <i>RangeCheck</i> .	If this value is 0xFFFF the <i>RangeCheck</i> function is not called. In this case memory resource intersections are the user's responsibilities. Otherwise, the <i>RangeCheck</i> function is called to verify that the requested NVM operation is consistent with the user-defined memory map.
regBase	UINT16	Base address of the register block.	This value should be on a 2K byte boundary within the first 32K bytes of the 64K byte MCU address space.
CallBack	void(*) (void)	Address of void callback function pointer	Any valid void function address
BDMEnable	BOOL	Select alternate function returns.	FALSE selects a normal function return; TRUE executes the BGND instruction at the function exit.
compareData	UINT16 *	The value of the first non-blank destination data.	<ol style="list-style-type: none"> <i>compareData</i> should lie within the MCU address space. <i>*compareData</i> is valid only when the function returns SGF_ERROR_NOT_BLANK.
compareAddress	UINT32 *	Destination address of the first non-blank word.	<ol style="list-style-type: none"> <i>compareAddress</i> should lie within the MCU address space. <i>*compareAddress</i> is valid only when the function returns SGF_ERROR_NOT_BLANK.
size	UINT32	The size of the blank check region in bytes. If <i>size</i> is zero, the return value is SGF_OK.	<ol style="list-style-type: none"> The <i>dest</i> and <i>size</i> parameters define a continuous sequence of NVM words that must lie entirely within either Flash or EEPROM but not both. This value must be a multiple of two.
dest	UINT32	The first sector address.	<ol style="list-style-type: none"> Any location within MCU EEPROM address space or the virtual Flash memory. This value must be a multiple of two.

NVMProgram()

This function programs the specified NVM regions with the contents of a source data buffer. The scope of this function is either Flash or EEPROM, but only one NVM region can be programmed per function call.

```

UINT16 NVMProgram (UINT16 funcPtr,
                  UINT16 pDescriptor,
                  UINT16 regBase,
                  void (*CallBack) (void),
                  BOOL BDMEnable,
                  BOOL ISREnable,
                  UINT16 source,
                  UINT32 size,
                  UINT32 dest);

```

Figure 5. NVMProgram() Function Prototype

Table 5. NVMProgram() Parameter Description

Argument	Type	Description	Range
funcPtr	UINT16	Function pointer to <i>RangeCheck</i> .	If this value is 0xFFFF the <i>RangeCheck</i> function is not called. In this case memory resource intersections are the user's responsibilities. Otherwise, the <i>RangeCheck</i> function is called to verify that the requested NVM operation is consistent with the user-defined memory map.
pDescriptor	UINT16	Pointer to the configuration descriptor.	Any address within the MCU address space.
regBase	UINT16	Base address of the register block.	This value should be on a 2K byte boundary within the first 32K bytes of the 64K byte MCU address space.
CallBack	void(*) (void)	Address of void callback function pointer	Any valid void function address
BDMEnable	BOOL	Select alternate function returns.	FALSE selects a normal function return; TRUE executes the BGND instruction at the function exit.
ISREnable	BOOL	Select Interrupt or polling logic.	TRUE selects interrupt logic. FALSE selects polling logic
source	UINT16	Address of the source data buffer.	This address must lie within MCU address space.
size	UINT32	The size of the blank check region in bytes. If <i>size</i> is zero, the return value is SGF_OK.	<ol style="list-style-type: none"> The <i>dest</i> and <i>size</i> parameters define a continuous sequence of NVM words that must lie entirely within either Flash or EEPROM but not both. This value must be a multiple of two.
dest	UINT32	The first sector address.	<ol style="list-style-type: none"> Any location within MCU EEPROM address space or the virtual Flash memory. This value must be a multiple of two.

ProgramVerify()

This function verifies that a source data buffer matches a corresponding region of NVM. The scope of this function is either Flash or EEPROM, but only one NVM region can be checked per function call.

```

UINT16 ProgramVerify (UINT16 funcPtr,
                     UINT16 pDescriptor,
                     UINT16 regBase,
                     void (*CallBack)(void),
                     BOOL BDMEnable,
                     UINT16 *compareSource,
                     UINT16 *compareData,
                     UINT32 *compareAddress
                     UINT16 source,
                     UINT32 size,
                     UINT32 dest);

```

Figure 6. ProgramVerify() Function Prototype

Table 6. ProgramVerity() Parameter Description

Argument	Type	Description	Range
funcPtr	UINT16	Function pointer to <i>RangeCheck</i> .	If this value is 0xFFFF the RangeCheck function is not called. In this case memory resource intersections are the user's responsibilities. Otherwise, the <i>RangeCheck</i> function is called to verify that the requested NVM operation is consistent with the user-defined memory map.
pDescriptor	UINT16	Pointer to the configuration descriptor.	Any address within the MCU address space.
regBase	UINT16	Base address of the register block.	This value should be on a 2K byte boundary within the first 32K bytes of the 64K byte MCU address space.
CallBack	void*(void)	Address of void callback function pointer	Any valid void function address
BDMEnable	BOOL	Select alternate function returns.	FALSE selects a normal function return; TRUE executes the BGND instruction at the function exit.
compareSource	UINT16 *	The value at the first source address that fails to verify.	1. <i>compareSource</i> should lie within the MCU address space. 2. <i>*compareSource</i> is valid only when the function returns SGF_ERROR_VERIFY.
compareData	UINT16 *	The value of the first destination address that fails to verify.	1. <i>compareData</i> should lie within the MCU address space. 2. <i>*compareData</i> is valid only when the function returns SGF_ERROR_NOT_BLANK.

Table continued on next page

Table 6. ProgramVerity() Parameter Description (Continued)

Argument	Type	Description	Range
compareAddress	UINT32 *	Address of the first destination address that fails to verify.	<ol style="list-style-type: none"> 1. <i>compareAddress</i> should lie within the MCU address space. 2. *<i>compareAddress</i> is valid only when the function returns SGF_ERROR_NOT_BLANK.
source	UINT16	Address of the source data buffer.	This address must lie within MCU address space.
size	UINT32	The size of the blank check region in bytes. If <i>size</i> is zero, the return value is SGF_OK.	<ol style="list-style-type: none"> 1. The <i>dest</i> and <i>size</i> parameters define a continuous sequence of NVM words that must lie entirely within either Flash or EEPROM but not both. 2. This value must be a multiple of two.
dest	UINT32	The first sector address.	<ol style="list-style-type: none"> 1. Any location within MCU EEPROM address space or the virtual Flash memory. 2. This value must be a multiple of two.

ParallelProgram()

This function supports both serial (i.e., single-block) programming and parallel (i.e., multi-block) programming for Flash. That is, ParallelProgram will program individual pagesets consisting of one or more words per pageset. In this context a pageset is defined by the set of words that is taken from one or more blocks. By definition, each word within a pageset lies at the same offset from the beginning of its block.

In addition, this function will also automatically program buffers containing multiple pagesets. With unlimited buffer space, the maximum number of pagesets that can be programmed in one function call is 0x8000 for Flash block size of 64 Kbytes, 0x4000 for Flash block size of 32 Kbytes. In most systems the maximum number of pagesets that can be programmed in one function call is limited by the amount of available RAM.

```

UINT16 ParallelProgram (UINT16 regBase,
                        void (*CallBack) (void)
                        BOOL BDMEnable,
                        UINT16 source,
                        UINT16 pagesetNumber,
                        UINT16 offset,
                        UINT16 flashBlocks);

```

Figure 7. ParallelProgram() Function Prototype

Table 7. ParallelProgram() Parameter Description

Argument	Type	Description	Range
regBase	UINT16	Base address of the register block.	This value should be on a 2K byte boundary within the first 32K bytes of the 64K byte MCU address Space.
CallBack	void(*) (void)	Address of void callback function pointer	Any valid void function address
BDMEnable	BOOL	Select alternate function returns.	FALSE selects a normal function return; TRUE executes the BGND instruction at the function exit.
source	UINT16	Address of the source data buffer.	This address must lie within MCU address space.
pagesetNumber	UINT16	The number of pagesets, where a pageset is a group of 2-byte data pieces to be programmed.	The pagesetNumber depends on the user's buffer size and the Flash block size, the maximum value is 0x8000 for 64KB block size, 0x4000 for 32KB block size.
offset	UINT16	Relative offset within the Flash block.	1. The offset should be aligned on a 2-byte boundary. 2. For block size of 64KB, 0<=offset<64K, for block size of 32KB, 0<=offset<32K.
FlashBlocks	UINT16	A bit-mapped argument defining the Flash blocks to be parallel programmed. The return code is SGF_OK if no blocks are enabled.	This value should be on a 2K byte boundary within the first 32K bytes of the 64K byte MCU address space.

ParallelProgramVerify()

This function is used to verify the data programmed by the ParallelProgram function. It uses the same offset and pageset scheme as the ParallelProgram function. The source buffer will contain the interleaved data that will be compared against a corresponding Flash region.

```
UINT16 ParallelProgramVerify (UNIT16 regBase,
                             void (*CallBack)(void),
                             BOOL BDMEnable,
                             UINT16 *compareSource,
                             UINT16 *compareData,
                             UINT32 *compareAddress,
                             UINT16 source,
                             UINT16 pagesetNumber,
                             UINT16 offset,
                             UINT16 flashBlocks);
```

Figure 8. ParallelProgramVerify() Function Prototype

Table 8. ParallelProgramVerify() Parameter Description

Argument	Type	Description	Range
regBase	UINT16	Base address of the register block.	This value should be on a 2K byte boundary within the first 32K bytes of the 64K byte MCU address space.
CallBack	void(*) (void)	Address of void callback function pointer	Any valid void function address
BDMEnable	BOOL	Select alternate function returns.	FALSE selects a normal function return; TRUE executes the BGND instruction at the function exit.
compareSource	UINT16 *	The value at the first source address that fails to verify.	1. <i>compareSource</i> should lie within the MCU address space. 2. <i>*compareSource</i> is valid only when the function returns SGF_ERROR_VERIFY.
compareData	UINT16 *	The value of the first destination address that fails to verify.	1. <i>compareData</i> should lie within the MCU address space. 2. <i>*compareData</i> is valid only when the function returns SGF_ERROR_NOT_BLANK.
compareAddress	UINT32	Address of the first destination address that fails to verify.	1. <i>compareAddress</i> should lie within the MCU address space. 2. <i>*compareAddress</i> is valid only when the function returns SGF_ERROR_NOT_BLANK.
source	UINT16	Address of the source data buffer.	This address must lie within MCU address space.

Table continued on next page

Table 8. ParallelProgramVerify() Parameter Description (Continued)

Argument	Type	Description	Range
pagesetNumber	UINT16	The Number of pagesets, where a pageset is a group of 2-byte data pieces to be programmed.	The pagesetNumber depends on the user's buffer size and the Flash block size, the maximum value is 0x8000 for 64KB block size, 0x4000 for 32KB block size.
offset	UINT16	Relative offset within the Flash block.	<ol style="list-style-type: none"> 1. The offset should be aligned on a 2-byte boundary. 2. For block size of 64KB, $0 \leq \text{offset} < 64K$, for block size of 32KB, $0 \leq \text{offset} < 32K$.
FlashBlocks	UINT16	A bit-mapped argument defining the Flash blocks to be parallel programmed. The return code is SGF_OK if no blocks are enabled.	This value should be on a 2K byte boundary within the first 32K bytes of the 64K byte MCU address space.

Checksum()

This function performs a 16-bit word sum without carry over the specified memory range. This function provides a rapid method of checking data integrity. The scope of this function is either Flash or EEPROM, but only one NVM region can be checked per function call.

```
UINT16 Checksum (UINT16 funcPtr,
                 UINT16 regBase,
                 void (*CallBack) (void),
                 BOOL BDMEnable,
                 UINT16 *sum,
                 UINT32 size,
                 UINT32 dest);
```

Figure 9. CheckSum() Function Prototype

Table 9. CheckSum() Parameter Description

Argument	Type	Description	Range
funcPtr	UINT16	Function pointer to <i>RangeCheck</i> .	If this value is 0xFFFF the <i>RangeCheck</i> function is not called. In this case memory resource intersections are the user's responsibilities. Otherwise, the <i>RangeCheck</i> function is called to verify that the requested NVM operation is consistent with the user-defined memory map.
regBase	UINT16	Base address of the register block.	This value should be on a 2K byte boundary within the first 32K bytes of the 64K byte MCU address space.
CallBack	void(*) (void)	Address of void callback function pointer	Any valid void function address
BDMEnable	BOOL	Select alternate function returns.	FALSE selects a normal function return; TRUE executes the BGND instruction at the function exit.
sum	UINT16 *	The checksum value for the specified memory range.	<ol style="list-style-type: none"> 1. <i>sum</i> should lie within the MCU address space. 2. <i>sum</i> is valid only when the function returns SGF_OK
Size	UINT32	The size of the checksum region in bytes. If <i>size</i> is zero, the return value is SGF_OK.	<ol style="list-style-type: none"> 1. The <i>dest</i> and <i>size</i> parameters define a continuous sequence of NVM words that must lie entirely within either Flash or EEPROM but not both. 2. The value must be a multiple of two.
dest	UINT32	The first sector address.	<ol style="list-style-type: none"> 1. Any location within MCU EEPROM address space or the virtual Flash memory. 2. This value must be a multiple of two.

FlashSecurityBypass()

This function temporarily bypasses the HCS12 security if the correct security keys are provided and if backdoor access is enabled.

NOTE

This function can only temporarily bypass the security – the contents of the Flash Protection/Options Field that determine the post-reset security state are not changed by this function.

```

UINT16 FlashSecurityBypass (UINT16 regBase,
                            void (*CallBack) (void)
                            BOOL BDMEnable
                            UINT16 *key);
    
```

Figure 10. FlashSecurityBypass() Function Prototype

Table 10. FlashSecurityBypass() Parameter Description

Argument	Type	Description	Range
regBase	UINT16	Base address of the register block.	This value should be on a 2K byte boundary within the first 32K bytes of the 64K byte MCU address space.
CallBack	void(*) (void)	Address of void callback function pointer	Any valid void function address
BDMEnable	BOOL	Select alternate function returns.	FALSE selects a normal function return; TRUE executes the BGND instruction at the function exit.
key	UINT16 *	A pointer to the four 16-bit key words.	key should lie within the MCU address space.

RangeCheck()

This function performs range checking for the given destination range and returns the memory type of the destination range only if the destination range is entirely of one memory type. If a memory resource of higher precedence overlaps the destination range, an error code is returned.

```
UINT16 RangeCheck (Descriptor *config,
                  UNIT16 regBase,
                  UNIT32 size,
                  UINT32 dest);
```

Figure 11. RangeCheck() Function Prototype

Table 11. RangeCheck() Parameter Description

Argument	Type	Description	Range
config	Descriptor *	Pointer to the configuration descriptor.	Any address within MCU address space.
regBase	UINT16	Base address of the register block.	This value should be on a 2K byte boundary within the first 32K bytes of the 64K byte MCU address space.
size	UINT32	Size of the region to be checked in bytes.	Any 32-bit value.
dest	UINT32	Destination starting address.	Any address within MCU address space or the virtual address space.

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. 2005 All rights reserved.