Freescale Semiconductor
Application Note

Document Number: AN4171
Rev. 0, 07/2010

U-Boot for i.MX25 Based Designs

Source Code Overview and Customization

by *Multimedia Applications Division*
Freescale Semiconductor, Inc.
Austin, TX

The Das Universal Bootloader (U-Boot) is a firmware/bootloader for hardware platforms. The U-Boot is widely used in embedded designs. The U-Boot supports common processor architectures such as ARM®, Power Architecture®, Microprocessor without Interlocked Pipeline Stages (MIPS), and x86®. In addition to the bootstrapping functionality, the U-Boot also supports other features that are part of the open source project, which is available under General Purpose Line (GPL). For example: device drivers, networking and file systems support, utilities to assist board bring up, testing, and so on.

The U-Boot firmware is ported to operate on several i.MX application processors and development boards. However, customers are often required to adapt to some key areas of the source code to make the source code operate on a new hardware platform based on the i.MX processor.

This application note deals with the i.MX25 3-stack U-Boot source code where adaptation is required. Also, this application note define guidelines for configuring Eclipse IDE for U-Boot development. For more information, See Section Appendix A, “Configuring Eclipse IDE for U-Boot Development.”

Contents

freescale semiconductor

1 Requirements

The requirements for the U-Boot project are as follows:

- Host computer with a Linux operating system
- Basic knowledge of Linux
- U-Boot source code for the i.MX platforms. See Section 3, “Getting the U-Boot Source Code,” for information about the U-Boot source code
- *i.MX25 Multimedia Applications Processor Reference Manual* (IMX25RM)
- *i.MX25 PDK 1.6 Linux User’s Guide*
- *i.MX25 PDK Hardware User’s Guide* (924-76349)
- Basic knowledge of C language and ARM assembly language
- Eclipse IDE with C/C++ development plug-in (required if the reader wants to follow the instructions in the Section Appendix A, “Configuring Eclipse IDE for U-Boot Development.”)

2 U-Boot Overview

The U-Boot project is a combination of two small bootloaders—PPCboot and ARMboot—these bootloaders are merged to create a U-Boot that provides support for expanded number of processors and boards. The home page of this project is available at http://www.denx.de/wiki/U-Boot/WebHome. The source code and documentation are distributed under the GPL license and is available free of cost.

The U-Boot project uses some portions of the Linux kernel code and maintains a similar source code structure and configuration scheme. This fact along with its set of features, stability, support for many processors and boards, easiness of porting, and active community of developers enhancing and supporting the project have contributed to make U-Boot, the most used bootloaders. The U-Boot is widely used in the embedded space where low-cost and reliability are critical.

The features of the U-Boot firmware are as follows:

- Bootstrap the hardware platform
- Load an OS image and transfer control to execute the OS
- Network download—TFTP, BOOTP, DHCP, and NFS
- Serial download—s-record and binary through Kermit
- Flash management—copy, erase, protect, cramfs, and jffs2
- Support for Flash types—CFI NOR Flash, NAND Flash, and MMC/SD cards
- Memory utilities—copy, dump, crc, check, and mtest
- IDE, SATA, boot from disk—raw block, ext2, fat, and reiserfs
- Interactive shell—choice of simple or busybox shell with many scripting features

For further information about the U-Boot project and FAQ, visit the U-Boot home page available at http://www.denx.de/wiki/U-Boot/WebHome.

3 Getting the U-Boot Source Code

The U-Boot source code is shipped along with the Linux Board Support Package (BSP) for the i.MX25 Platform Development Kit (PDK). This BSP is embedded in the Linux Software Development Kit (SDK). The Linux SDK for the i.MX25 processor and documentation is available at http://www.freescale.com/imx25pdk

At the time of creating this application note, the latest available version of Linux SDK was IMX25_SDK16_LINUX_BSP, and it contained the BSP based on the Linux kernel version 2.6.28. To install Linux BSP in the host computer, refer to the relevant documents.

After successful installation of the Linux BSP, the Linux Target Image Builder (LTIB) and GNU tool chain (for ARM) are ready for use. In this application note, the LTIB installation path is referenced as `<LTIB_DIR>`.

To obtain the U-Boot source code for the i.MX platforms, use the following command:

```
cd <LTIB_DIR>
./ltib -m prep -p u-boot
```

From this set of commands, the U-Boot source code package is extracted and the i.MX patches are applied. The patched source code is located at:

```
<LTIB_DIR>/rpm/BUILD/u-boot-2009.01
```

To rebuild the source code using LTIB, use the following command:

```
./ltib -m scbuild -p u-boot
```

Executing this command configures the U-Boot for the i.MX25 3-stack platform and the following binaries are generated:

- u-boot—file in Executable and Linkable Format (ELF) with symbols and debugging information.
- u-boot.bin—plain binary file. This file is programmed to a boot media (NAND, NOR, SD, and so on) to bootstrap the i.MX25 3-stack board.

NOTE

The term 3-stack board is used to describe the i.MX development platform, consisting of three boards—CPU, debug, and personality.

It is recommended to verify with the Freescale representative if new U-Boot patches or code is available for the i.MX platforms, prior to starting a code customization.

4 Source Code Tree Overview

The U-Boot source code structure is similar to the one used by the Linux Kernel. This section gives an overview of the source code tree. To list the directory tree, use the following command:

```
cd <LTIB_DIR>/rpm/BUILD/u-boot-2009.01
ls
```

Table 1 outlines the top-level directories in the source code tree and their description.

Table 1. U-Boot Source Code Top-Level Directories

Directory	Description
api	U-Boot machine/arch independent API for external applications
api_examples	Example applications using the API
board	Board dependent files/directories
common	Misc architecture independent functions
cpu	CPU specific files
Disk	Code for disk drive partition handling
Doc	Basic documentation files
drivers	Device drivers for common peripherals
examples	Example code for standalone applications
Fs	Common file systems support
include	Header files (.h)
lib_arm	Files generic to the ARM architecture
lib_avr32	Files generic to the AVR32 architecture
lib_blackfin	Files generic to the blackfin architecture
libfdt	Flat tree manipulation library
lib_generic	Files generic to all architectures
lib_i386	Files generic to the i386 architecture
lib_m68k	Files generic to the m68k architecture
lib_microblaze	Files generic to the microblaze architecture
lib_mips	Files generic to the MIPS architecture
lib_nios	Files generic to the Altera NIOS architecture
lib_ppc	Files generic to the PowerPC architecture
lib_sh	Files generic to the SH architecture
lib_sparc	Files generic to the SPARC architecture
nand_spl	Support for NAND Flash boot with stage 0 boot loader
net	Networking support (bootp,tftp, rarp, nfs, and so on)
onenand_ipl	One NAND initial program loader

Table 1. U-Boot Source Code Top-Level Directories (continued)

Directory	Description
patches	Patches for the i.MX platforms (applied during the command ./ltib -m prep -p u-boot)
post	Power on self test
tools	Tools for building S-Record files, U-Boot images, and so on

Table 2 outlines the list of files in the top-level directory and their description.

Table 2. U-Boot Source Code Top-Level Files

File	Description
README	This file gives information about the U-Boot project. Several sections of this application note are based on the information from this file.
Makefile	The top-level `Makefile`. This file is used when executing the board configuration and the build processes. The new board configurations are to be added to this file.
MAKEALL	This script is used to configure and build all the supported boards in one step. The list of boards in this file must be updated manually when a new board is added.
CREDITS	The author and main contributors of the U-Boot project are listed in this file (includes their email).
COPYING	This file contains the license of the U-Boot source code.

4.1 i.MX25 Related Source Files

The i.MX25 application processors (based on ARM926EJ-S) and its development platform (3-stack board) are added to the U-Boot project.

Table 3 outlines the 3-stack related source files in the source code tree and their description.

Table 3. i.MX25 3-Stack Related Source Files

Directory/File	Description
board/freescale/mx25_3stack/dcdheader.S	Image header that is appended to the u-boot.bin file; includes Device Configuration Data (DCD)
board/freescale/mx25_3stack/lowlevel_init.S	Board low-level initialization routines in the assembly language
board/freescale/mx25_3stack/mx25_3stack.c	Board initialization routines in the C language
board/freescale/mx25_3stack/u-boot.lds	Linker script
board/freescale/mx25_3stack/config.mk	Defines the base address for binary (TEXT_BASE)
cpu/arm926ejs/cpu.c	CPU specific code in the C language: interrupts, stack, mmu, and cache setup routines
cpu/arm926ejs/interrupts.c	Nothing really useful here
cpu/arm926ejs/start.S	CPU low-level initialization code, the first function executed when U-Boot starts is defined here
cpu/arm926ejs/mx25/generic.c	Routines for calculating CPU and peripheral clocks and a function to call the on-chip Ethernet initialization routine

Table 3. i.MX25 3-Stack Related Source Files (continued)

Directory/File	Description
cpu/arm926ejs/mx25/gpio.c	Routines for setting up the General Purpose Input/Output (GPIO) pins
cpu/arm926ejs/mx25/interrupts.c	Starts a timer and provides functions around the timer count. Also, implements the `reset_cpu` function
cpu/arm926ejs/mx25/iomux.c	IOMUX setup routines
cpu/arm926ejs/mx25/serial.c	On-chip Universal Asynchronous Receiver/Transmitter (UART) driver and serial I/O functions
include/asm-arm/arch-mx25/gpio.h	GPIO function definitions
include/asm-arm/arch-mx25/iomux.h	IOMUX control definitions and functions
include/asm-arm/arch-mx25/mmc.h	Nothing is defined here
include/asm-arm/arch-mx25/mx25-regs.h	On-chip modules base addresses and registers definitions
include/asm-arm/arch-mx25/mx25.h	Definitions of functions to get clocks
include/asm-arm/arch-mx25/mx25_pins.h	i.MX25 I/O pin list
include/asm-arm/arch-mx25/mxc_nand.h	NAND Flash Controller (NFC) registers definitions and macros
include/asm-arm/arch-mx25/sdhc.h	Secure Digital Host Controller register definitions and functions
include/configs/mx25_3stack.h	i.MX25 3-stack board high-level configuration
lib_arm/board.c	This file implements high-level board initialization functions and allows the user to configure the initialization sequence
drivers/mtd/nand/mxc_nand.c	NFC low-level driver
drivers/mtd/nand/nand.c	NAND Flash definitions and initialization function
drivers/mtd/nand/nand_base.c	NAND Flash generic to the Memory Technology Device (MTD) driver
drivers/mtd/nand/nand_bbt.c	Bad block table support for the NAND Flash driver
drivers/mtd/nand/nand_ecc.c	Error correction code support for NAND Flash
drivers/mtd/nand/nand_ids.c	NAND Flash chips ID list
drivers/mtd/nand/nand_util.c	Utilities to work with NAND Flash, write and read skipping bad blocks, lock the NAND Flash during accesses, and so on
drivers/mmc/fsl_esdhc.c	Functions to use the MMC/SD card
drivers/mmc/fsl_mmc.c	I/O control access for the MMC/SD cards
common/env_mmc.c	Functions to store and retrieve the environment variables from the MMC/SD card
drivers/i2c/mxc_i2c.c	I^2C driver for the i.MX architecture
drivers/net/smc911x.c	SMSC911x Ethernet device driver (used for SMSC LAN9217)
drivers/net/mxc_fec.c	On-chip Fast Ethernet Controller (FEC) device driver

5 Create a New Board Based on i.MX25 3-Stack

In the process of adapting U-Boot to a custom design, it is recommended to create a new board directory within the code tree where all the files and new configurations can be stored. This way, the original files that are used as base (in this case, the i.MX25 3-stack board) remains unchanged and available for comparison. If the device drivers or any other non-board specific code is adapted, it is a good practice to take a backup copy of the original code and make it available in the source tree for comparison. If required, see Section Appendix A, "Configuring Eclipse IDE for U-Boot Development," for information about Eclipse IDE configuration before proceeding with the following sections.

To create a new board based on the i.MX25 3-stack, perform the following steps:

1. Clean the source code tree (all the output files of previous build are deleted):

```
make distclean
```

2. Copy the contents of the current `mx25_3stack` board directory to a new directory and provide a meaningful name to identify the design. This application note uses `mx25_custom` as a new directory name.

```
cp -r board/freescale/mx25_3stack/ board/freescale/mx25_custom
```

3. Rename the `mx25_3stack.c` file accordingly:

```
mv board/freescale/mx25_custom/mx25_3stack.c
board/freescale/mx25_custom/mx25_custom.c
```

4. Adjust the `board/freescale/mx25_custom/Makefile` file to fit the new file name:

   Change the line, `COBJS  := mx25_3stack.o`, to `COBJS  := mx25_custom.o`

5. Copy the contents of the current `mx25_3stack` board configuration file to a new file and provide a meaningful name. This application note uses `mx25_custom.h` as a new file name.

```
cp include/configs/mx25_3stack.h include/configs/mx25_custom.h
```

6. Create an entry in the top-level directory, `Makefile`, for the new custom board configuration. This file is sorted in the alphabetical order:

```
mx25_custom_config  : unconfig
        @$(MKCONFIG) $(@:_config=) arm arm926ejs mx25_custom freescale mx25
```

NOTE

The U-Boot project developers recommend to add any new board to the `MAKEALL` script too and run the script to verify if the new code has not broken any other platform builds. This is necessary if a patch is submitted back to the U-Boot community. For further information, consult the U-Boot `README` file.

7. Adapt to any fixed paths. In this case, the linker script, `mx25_custom/u-boot.lds` has one path.

   Replace `mx25_3stack` with `mx25_custom` using the following command:

```
board/freescale/mx25_custom/dcdheader.o
```

8. Set the `CROSS_COMPILE` and `PATH` environment variables in the console as the build process is executed manually (without LTIB):

```
export CROSS_COMPILE=arm-none-linux-gnueabi-
export
```

```
PATH=/opt/freescale/usr/local/gcc-4.1.2-glibc-2.5-nptl-3/arm-none-linux-gnueabi/bin
/:$PATH
```

9. Configure the system for the new board:

```
make mx25_custom_config
```

10. Build the new board. Verify that no errors are found and the U-Boot binaries are created:

```
make
```

The new board is a replica of the i.MX25 3-stack board. The next step is to adapt some portions of the code to make it suitable for the new hardware design.

The following sections provide guidelines to proceed further with the code customization process.

6 Customize the Code

This section describes the key areas within the source code where customizing is required. Also, note that depending on the design and requirements, the code is modified either more or less than what is explained in this application note.

6.1 Internal Boot versus External Boot

The i.MX25 applications processor provides different boot modes and these are described in detail in the *i.MX25 Multimedia Applications Processor Reference Manual* (IMX25RM).

The boot modes of the i.MX25 processor are as follows:

- Internal boot mode—allows selection of all boot sources such as NOR, NAND, MMC/SD, OneNAND, Parallel Advanced Technology Attachment (P-ATA), Serial ROM/Flash, and so on. After Power On Reset (POR) or reset, the processor's ROM code samples the boot pins or eFuses and loads the first set of code from the selected boot media. This code must have a Flash header at a particular offset and it varies depending on the boot source. The Flash header stores information about the application in a specific structure. It can also store DCD, which is a block of data processed by the i.MX25 to configure the hardware at boot time. This enables the configuration of some on-chip modules and external peripherals before moving to the entry point of the application.
- External boot mode—allows selection of only the NOR and NAND Flash as the boot sources. After POR or reset, the i.MX25 processor samples the boot pins or eFuses, and jumps directly to the base address of the selected boot source (base of NFC buffer in the case of NAND Flash). No Flash header is required to identify the application and the hardware configuration is carried out by the loaded application.

6.2 Flash Header

The Flash header `board/mx25_custom/dcdheader.S` is appended to the top of the `u-boot.bin` file as indicated by the linker script. One of the elements of the Flash header is the DCD, which is a block of data processed by the i.MX25 ROM code to configure some on-chip modules and external peripherals at boot up. For more information, refer to the System Boot chapter of the *i.MX25 Multimedia Applications Processor Reference Manual* (IMX25RM).

The Flash header is appended to the image when the following configurations are set:

```
#define CONFIG_FLASH_HEADER     1
#define CONFIG_FLASH_HEADER_OFFSET 0x400
#define CONFIG_FLASH_HEADER_BARKER 0xB1
```

In addition, if DCD is used and the SDRAM initialization is performed by the DCD data, the user can set the following configuration to disable the U-Boot relocation to RAM, because it is already performed by the i.MX ROM code:

```
#define CONFIG_SKIP_RELOCATE_UBOOT
```

6.3 Customize SDRAM Initialization

If the SDRAM device is modified in the custom platform, the i.MX25 Enhanced SDRAM Controller and initialization sequence code require adaptation to operate with the new device.

In this case, modify the Flash header (DCD data)—open the `dcdheader.s` file and modify the values of the DCDGEN macros (add/remove values) in accordance with the specification sheet of SDRAM devices and the *i.MX25 Multimedia Applications Processor Reference Manual* (IMX25RM).

The DCDGEN macro transforms an identifier number, address of a register, a value to write to this register, and length of the access into the corresponding data, which is to be appended to the U-Boot binary.

NOTE

Ensure to adjust the length of the DCD structure if data is added or removed from it.

If the SDRAM base or the size is changed, the following values in the custom board configuration file need to be modified:

```
#define PHYS_SDRAM_1            CSD0_BASE
/* iMX25 V-1.0 has 128MB but V-1.1 has only 64MB */
#ifdef CONFIG_MX25_3DS_V10
#define PHYS_SDRAM_1_SIZE       (128 * 1024 * 1024)
#else
#define PHYS_SDRAM_1_SIZE       (64 * 1024 * 1024)
#endif
```

6.4 Check if the CPLD Code is Required

The i.MX PDKs (3-stack) contain a debug board in which glue logic is implemented in a Complex Programmable Logic Device (CPLD). This device is memory mapped to the on-chip WEIM at Chip Select 5 (CS5).

The CPLD supports the following features:

- A 16-bit slave interface to the CPU data bus
- Address decode and control for the external Ethernet controller
- Address decode and control for the external UART controller
- Level shift for Ethernet signals and UART signals

- Control and status registers for various board functions

When the new board does not have the CPLD or the memory range of CS5 (0xB600_0000 - 0xB7FF_FFFF) is used for a different purpose, it is necessary to remove the code targeting the CPLD to avoid possible errors.

To clarify further on this point, imagine that the new board has an external Ethernet controller that is attached directly to the i.MX processor at CS5. If CPLD initialization code is executed, it can corrupt the external Ethernet chip initialization and prevent the device driver of this peripheral from operating correctly.

One such case is when the new board requires CS5 for a different purpose, it is advisable to have only the CS5 initialization and remove the CPLD (alias debug board or peripheral bus controller) initialization.

Another case is when CS5 is not used in the new board, there is no need to execute the unused code. In such case, remove both the CS5 initialization and the CPLD initialization.

To remove/comment the CS5 and/or CPLD code in U-Boot source files, look for references to the following keywords—CS5, DBG, and PBC. Also, look for other references to the CS5 address space.

For example, in the i.MX25 custom board, if no CS5 is required, comment out the following code:

In the `dcdheader.s` file, remove the CS5 configuration data from the DCD:

```
/* WEIM config-CS5 init -- CPLD */
DCDGEN( 1, 4, 0xB8002050, 0x0000D843) /* CS5_CSCRU */
DCDGEN( 2, 4, 0xB8002054, 0x22252521) /* CS5_CSCRL */
DCDGEN( 3, 4, 0xB8002058, 0x22220A00) /* CS5_CSCRA */
```

NOTE

Ensure to adjust the length of the DCD structure when data is removed from it.

In the `lowlevel_init.s` file, remove the following code:

```
/* Init Debug Board CS5 */
        REG 0xB8002050, 0x0000D843
        REG 0xB8002054, 0x22252521
        REG 0xB8002058, 0x22220A00
```

If CS5 needs to be initialized without wanting the code to interact with the CPLD logic, look for accesses made to the CS5 address space. For example, refer to the following constants (these are not found in the current U-Boot code of i.MX25 3-stack but can be found in the future U-Boot code):

```
#define PBC_LED_CTRL          (0x20000)
#define PBC_SB_STAT           (0x20008)
#define PBC_ID_AAAA           (0x20040)
#define PBC_ID_5555           (0x20048)
#define PBC_VERSION           (0x20050)
#define PBC_ID_CAFE           (0x20058)
#define PBC_INT_STAT          (0x20010)
#define PBC_INT_MASK          (0x20038)
#define PBC_INT_REST          (0x20020)
#define PBC_SW_RESET          (0x20060)
```

6.5 Board Initialization Sequence

As part of the U-Boot boot up process, the `start_armboot` function executes the initialization sequence of a board. This sequence defines the order in which other routines are called and it is customized by the user. To adapt it, modify the `init_sequence[]` array defined in the `lib_arm/board.c` file:

```
init_fnc_t *init_sequence[] = {
        cpu_init,               /* basic cpu dependent setup */
        board_init,             /* basic board dependent setup */
        interrupt_init,         /* set up exceptions */
        env_init,               /* initialize environment */
        init_baudrate,          /* initialze baudrate settings */
        serial_init,            /* serial communications setup */
        console_init_f,         /* stage 1 init of console */
        display_banner,         /* say that we are here */
#if defined(CONFIG_DISPLAY_CPUINFO)
        print_cpuinfo,          /* display cpu info (and speed) */
#endif
#if defined(CONFIG_DISPLAY_BOARDINFO)
        checkboard,             /* display board info */
#endif
#if defined(CONFIG_HARD_I2C) || defined(CONFIG_SOFT_I2C)
        init_func_i2c,
#endif
        dram_init,              /* configure available RAM banks */
        display_dram_config,
        NULL,
};
```

6.6 Include, Exclude, or Remap Device Drivers

After the build, the U-Boot binary should only include the code to be used at the target board. The i.MX25 3-stack board configuration file includes device drivers such as I^2C, UART, FEC, NAND, and so on for both the on-chip and off-chip peripherals.

In the process of customizing U-Boot, the drivers included in the custom board configuration file must be reviewed to verify if all of these drivers are needed for the design. Depending on the requirements, include or exclude the device drivers, or remap them in case the base address has changed in the design. Some examples are described in the following sections.

6.6.1 UART Driver

The current configuration includes the UART driver using the `CONFIG_MX25_UART` constant and selects the UART1 driver using the `CONFIG_MX25_UART1` constant. To remap the UART driver, refer to the `mx25-regs.h` file and identify the base address of the UART driver which is to be used and perform the following steps:

1. Change the ‘1’ used in the `#define CONFIG_MX25_UART1` file with the UART number which is used.
2. Change the ‘1’ used in the `#ifdef CONFIG_MX25_UART1` in `cpu/arm926ejs/mx25/serial.c` file with the UART number which is used.
3. Change the physical base address in the `#define UART_PHYS 0x43f90000` in `cpu/arm926ejs/mx25/serial.c` file with the base address of the UARTx which is used.

4. Change the IOMUX and pad configuration for the UARTx in `board/freescale/mx25_3stack/mx25_3stack.c` with the new UART number.

```
/* setup pins for UART1 */
        /* UART 1 IOMUX Configs */
        mxc_request_iomux(MX25_PIN_UART1_RXD, MUX_CONFIG_FUNC);
        mxc_request_iomux(MX25_PIN_UART1_TXD, MUX_CONFIG_FUNC);
        mxc_request_iomux(MX25_PIN_UART1_RTS, MUX_CONFIG_FUNC);
        mxc_request_iomux(MX25_PIN_UART1_CTS, MUX_CONFIG_FUNC);
        mxc_iomux_set_pad(MX25_PIN_UART1_RXD, PAD_CTL_HYS_SCHMITZ | PAD_CTL_PKE_ENABLE|
                                              PAD_CTL_PUE_PUD | PAD_CTL_100K_PU);
        mxc_iomux_set_pad(MX25_PIN_UART1_TXD, PAD_CTL_PUE_PUD | PAD_CTL_100K_PD);
        mxc_iomux_set_pad(MX25_PIN_UART1_RTS, PAD_CTL_HYS_SCHMITZ | PAD_CTL_PKE_ENABLE|
                                              PAD_CTL_PUE_PUD | PAD_CTL_100K_PU);
        mxc_iomux_set_pad(MX25_PIN_UART1_CTS, PAD_CTL_PUE_PUD | PAD_CTL_100K_PD);
```

6.6.2 SMSC Ethernet Driver

As mentioned in Section 6.4, “Check if the CPLD Code is Required.” the SMSC LAN9217 device located in the debug card is interfaced through the CPLD logic and therefore mapped out at an offset within CS5 (check CPLD memory map). In the case of SMSC device, this offset is 0 and therefore the CS5 base address is same as the SMSC driver.

If the SMSC LAN9217 or a compatible device is present in the new board, the driver code must be included to the U-Boot build. To do so, add the following definitions in the custom board configuration file:

```
/*Support LAN9217*/
#define CONFIG_SMC911X          1
#define CONFIG_SMC911X_16_BIT 1
#define CONFIG_SMC911X_BASE CS5_BASE_ADDR
```

While including or excluding the Ethernet device drivers, assign a suitable value to the multiple Ethernet interface definitions. The U-Boot build uses the following configurations to know how many Ethernet devices are present in the system.

```
#define CONFIG_HAS_ETH1
#define CONFIG_NET_MULTI 1
```

6.6.3 MMC Driver and Commands

Depending on the need, the MMC device driver is included or excluded from the U-Boot build. To do so, add or remove the following definitions from the board configuration file:

```
#define CONFIG_FSL_MMC //Includes the MMC driver
#define CONFIG_MMC              1 //Required for other definitions inside the MMC driver
#define CONFIG_CMD_MMC //Enables the MMC U-Boot commands
#define CONFIG_DOS_PARTITION    1 //Enables DOS partition read/write
#define CONFIG_CMD_FAT          1 //Enables the U-Boot FAT commands
#define CONFIG_MMC_BASE         0x0 //Defines the base of MMC card
#define CONFIG_ENV_IS_IN_MMC    1 //Environment variables will be stored in MMC card
#define CONFIG_ENV_OFFSET       (768 * 1024) //Offset within the MMC card where the
environment variables will be stored at
```

6.6.4 NOR Flash Driver and Commands

The NOR Flash driver (Common Flash Interface) and the Flash commands are included only when the following definitions are used. In this case, CS0 is the base for the NOR Flash. Modify the values according to the new board configuration:

```
#define CONFIG_SYS_FLASH_BASE           CS0_BASE_ADDR
#define CONFIG_SYS_MAX_FLASH_BANKS 1    /* max number of memory banks */
#define CONFIG_SYS_MAX_FLASH_SECT 512   /* max number of sectors on one chip */
/* Monitor at beginning of flash */
#define CONFIG_SYS_MONITOR_BASE CONFIG_SYS_FLASH_BASE
#define CONFIG_SYS_MONITOR_LEN          (512 * 1024)
/*-----------------------------------------------------------------------
* CFI FLASH driver setup
*/
#define CONFIG_SYS_FLASH_CFI            1/* Flash memory is CFI compliant */
#define CONFIG_FLASH_CFI_DRIVER         1/* Use drivers/cfi_flash.c */
/* A non-standard buffered write algorithm */
#define CONFIG_FLASH_SPANSION_S29WS_N   1
#define CONFIG_SYS_FLASH_USE_BUFFER_WRITE 1/* Use buffered writes (~10x faster) */
#define CONFIG_SYS_FLASH_PROTECTION     1/* Use hardware sector protection */
```

6.6.5 NAND Flash Driver and Commands

When the `CONFIG_MX25` and `CONFIG_CMD_NAND` macros are defined, the NAND Flash driver and the commands are included to the U-Boot build. Since disabling the `CONFIG_MX25` macro impacts other functionalities, it is recommended to create a specific `#define` macro for the NAND low-level driver (`mxc_nand.c`), so that the NAND Flash Driver can be enabled or disabled like the other drivers.

For the NAND driver and MTD subsystem, it is important to highlight the place where the NAND chip IDs are defined. This is because sometimes it is necessary to add a new NAND manufacturer or Device ID to the list of supported NANDs. To do so, check the following structures in the `drivers/mtd/nand/nand_ids.c` file:

```
struct nand_flash_dev nand_flash_ids[] = {
.....
.....
        {"NAND 128MiB 1,8V 16-bit",     0x49, 512, 128, 0x4000, NAND_BUSWIDTH_16},
        {"NAND 128MiB 3,3V 16-bit",     0x74, 512, 128, 0x4000, NAND_BUSWIDTH_16},
        {"NAND 128MiB 3,3V 16-bit",     0x59, 512, 128, 0x4000, NAND_BUSWIDTH_16},
        {"NAND 256MiB 3,3V 8-bit",      0x71, 512, 256, 0x4000, 0},
.....
.....
        {NULL,}
};
struct nand_manufacturers nand_manuf_ids[] = {
        {NAND_MFR_TOSHIBA, "Toshiba"},
        {NAND_MFR_SAMSUNG, "Samsung"},
        {NAND_MFR_FUJITSU, "Fujitsu"},
.....
.....
        {0x0, "Unknown"}
};
```

6.6.6 I²C Driver

The I²C communications channel is used to interface with the Power Management IC (PMIC) in the i.MX25 3-stack board. In this board, the I²C port 1 is used with the base address 0x43F80000. If PMIC is relocated to another I²C port or if it is changed, make sure to modify the code at the following locations:

- `include/configs/mx25_custom.h`:

```
#define CONFIG_CMD_I2C
#define CONFIG_HARD_I2C                                                 1
#define CONFIG_I2C_MXC          1
#define CONFIG_SYS_I2C_PORT I2C1_BASE_ADDR
#define CONFIG_SYS_I2C_SPEED            40000
#define CONFIG_SYS_I2C_SLAVE            0xfe
```

- `board/freescale/mx25_custom/mx25_custom.c` (inside the `board_init` function):

```
mxc_request_iomux(MX25_PIN_I2C1_CLK, MUX_CONFIG_SION);
mxc_request_iomux(MX25_PIN_I2C1_DAT, MUX_CONFIG_SION);
mxc_iomux_set_pad(MX25_PIN_I2C1_CLK, 0x1E8);
mxc_iomux_set_pad(MX25_PIN_I2C1_DAT, 0x1E8);
```

Also, if `BOARD_LATE_INIT` macro is defined in the board configuration file, the function `board_late_init` in the `board/freescale/mx25_custom/mx25_custom.c` file is included and executed. This function performs a write operation to a PMIC register and it works if the PMIC I²C port is remapped. However, if PMIC is modified, the function does not work. Therefore, adapt or exclude this code from the U-Boot build.

6.7 Miscellaneous Customizations

This section describes the various types of customizations with the help of code.

6.7.1 Environment Variables and Auto Boot Command

The U-Boot shell allows the user to set environment variables similar to the Linux shell. These variables can be defined at the U-Boot prompt using the `setenv` command or can be hardcoded in the source code. One of these variables, `bootcmd` is executed automatically when the auto boot feature is enabled. To configure these elements, refer to the custom board configuration file and modify the following code:

```
#define CONFIG_BOOTDELAY        3
#define CONFIG_LOADADDR         0x80800000      /* loadaddr env var */
#define CONFIG_EXTRA_ENV_SETTINGS                                       \
        "netdev=eth0\0"                                                  \
        "ethprime=fec\0"                                                 \
        "bootargs_base=setenv bootargs console=ttymxc0,115200\0"         \
        "bootargs_nfs=setenv bootargs $(bootargs) root=/dev/nfs "        \
                "ip=dhcp nfsroot=$(serverip):$(nfsrootfs),v3,tcp\0"       \
        "bootcmd=run bootcmd_net\0"                                      \
        "bootcmd_net=run bootargs_base bootargs_mtd bootargs_nfs; "      \
                "tftpboot 0x81000000 uImage; bootm\0"
```

6.7.2 Change I and U-Boot Prompt

When the U-Boot boots up and before it reaches the prompt, there are some debug messages displayed in the console and one of these messages is the name of the board. This is printed when executing the

checkboard function in the board/freescale/mx25_custom/mx25_custom.c file. If required, replace the name of the board with a suitable string.

```
int checkboard(void)
{
        printf("Board: i.MX25 MAX PDK (3DS)\n");
        return 0;
}
```

The U-Boot prompt is displayed after all the setup functions are executed. The string that is displayed at the prompt can be changed in the include/configs/mx25_custom.h file with the following definition.

```
#define CONFIG_SYS_PROMPT       "MX25 U-Boot > "
```

6.7.3 Change the Linux Machine Type and Address of ATAGs

When the U-Boot is used to boot a Linux Operating System (OS), the kernel parameters are placed in a special area in memory in the form of ATAGs (if this feature is enabled in the board configuration file). The address of this location in memory is user configurable. In addition, one of these parameters passed to the kernel is the machine type, which is a number used to identify the board and it must match between Linux and U-Boot. If the machine type does not match, the Linux kernel does not boot up. To change these parameters, refer to the board/freescale/mx25_custom/mx25_custom.c file and modify the following lines of code:

```
gd->bd->bi_arch_number = MACH_TYPE_MX25_3DS;    /* board id for linux */
gd->bd->bi_boot_params = 0x80000100;    /* address of boot parameters */
```

The ATAGs are enabled with the following definitions in the board configuration file:

```
#define CONFIG_CMDLINE_TAG              1       /* enable passing of ATAGs */
#define CONFIG_SETUP_MEMORY_TAGS        1
#define CONFIG_INITRD_TAG               1
```

7 Enable Debugging Information

While customizing U-Boot, debugging is the most time consuming activity. During this phase, it is useful to have as much information as possible to detect the root cause for errors. For this purpose, the U-Boot source code contains several functions or macros that, when enabled, print extra information in the console at runtime. Some examples are as follows:

In the include/common.h file, two debug macros are defined. When the #define DEBUG macro is set in this file, all the files that include common.h and use the debug(fmt, args…) or debugX(level, fmt, args…) macro prints the additional information. If too much information is printed, enable the #define DEBUG macro only in a particular file(s) before including common.h. In both cases, the source code needs to be recompiled.

In addition, there are other files that have their own debug macros or functions. In MTD subsystem and NAND driver, the #define CONFIG_MTD_DEBUG file and a debug level are used to print the additional information. Other examples are #define DEBUG_SPI in the SPI subsystem, #define DEBUG_I^{2}C in the I^{2}C subsystem, and #define DEBUG_JFFS2.

8 Revision History

Table 4 provides the revision history for this application note.

Table 4. Document Revision History

Rev. Number	Date	Substantive Change(s)
0	07/2010	Initial release

Appendix A Configuring Eclipse IDE for U-Boot Development

To assist during the source code customization process, it is recommended to set up an Integrated Development Environment (IDE) in the host computer. This section provides the instructions to set-up the Eclipse IDE (for C/C++ developers).

Eclipse installation is beyond the scope of this application note. For information about installing Eclipse in the host computer, refer to the following link—http://www.eclipse.org/cdt/

After installing the Eclipse IDE in the Linux host, perform the following steps to configure the Eclipse IDE for the U-Boot development:

1. Open Eclipse.
2. Click on File > New > Project.
3. In the New Project wizard, select C > Standard Make C Project. (See Figure 1)

Figure 1 shows the new project wizard of the Eclipse IDE.

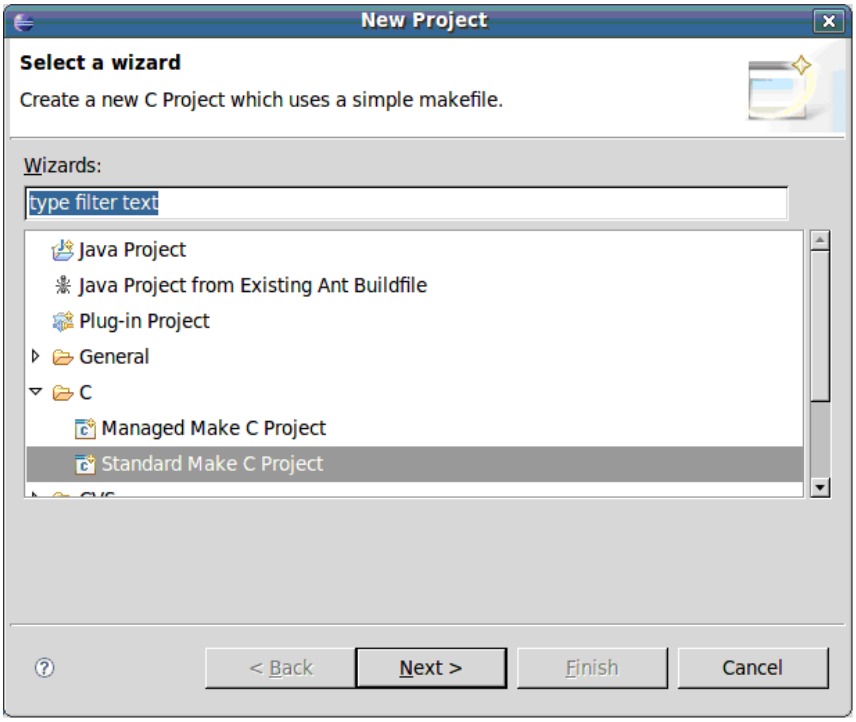

Figure 1. Eclipse IDE New Project Wizard

4. Click Next.
5. The C/Make Project window appears. Type a project name in the Project name field and deselect the Use default location check box.
6. Click on the Browse button to search for the path where the U-Boot source code is located. (See Figure 2)

Figure 2 shows the name and location of the project.

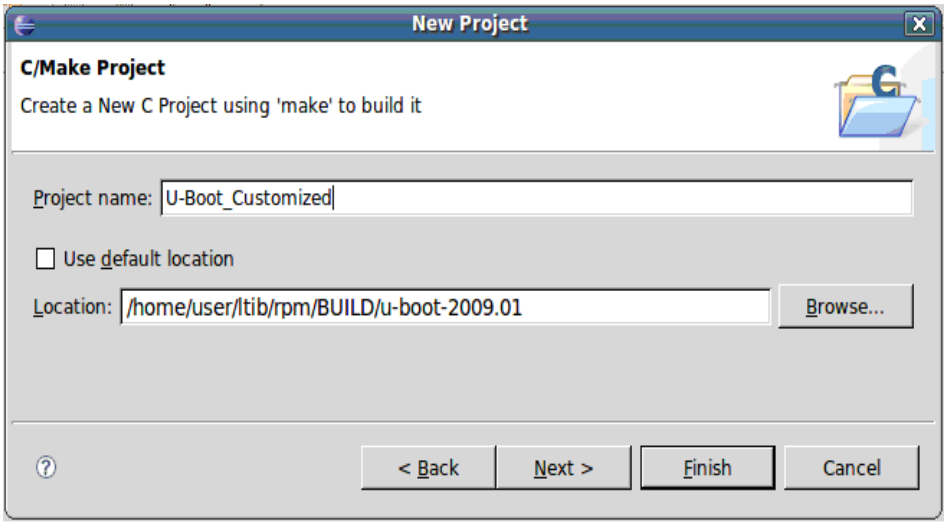

Figure 2. Project Name and Location

7. Click on the Finish button to close the wizard.
8. In the Eclipse main window, deselect the Project > Build automatically option.
9. Configure the project properties. Click on Project > Properties to open the properties window.
10. Select the C/C++ Include Paths and Symbols option and perform the following steps in this window:
    - Disable all the automatically discovered paths and symbols (multiple selection is allowed to disable all of them at once).
    - Add the include path from workspace. For example—U-Boot_Customized/include. (See Figure 3)

Figure 3 shows the paths to be included from the workspace.

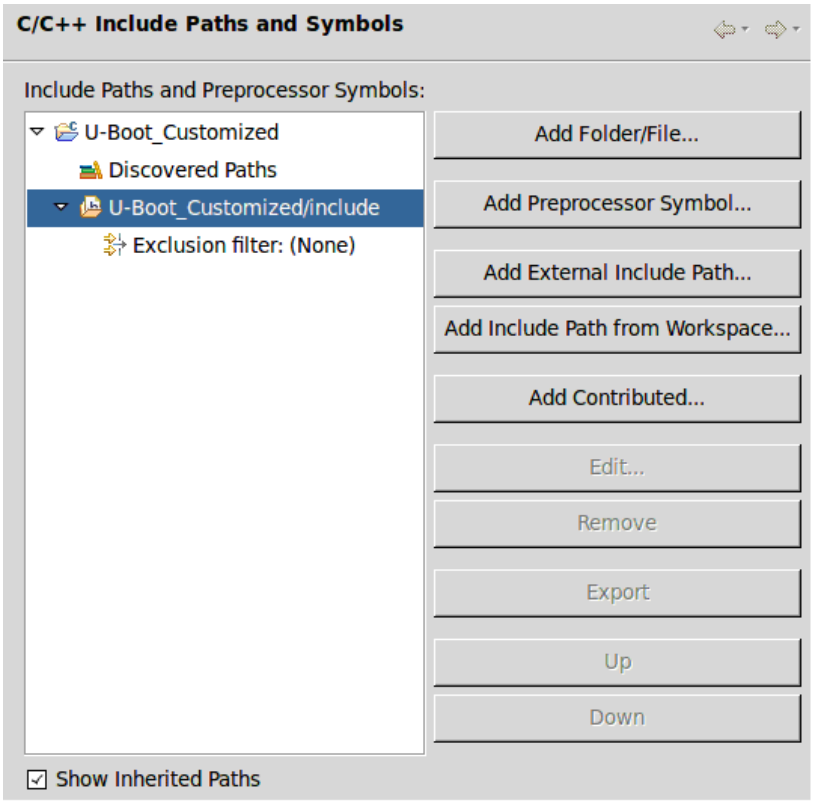

Figure 3. Include Path from Workspace

11. Select the C/C++ Indexer and perform the below steps:
    – It is recommended to enable the fast C/C++ Indexer to have assisted source code navigation. Optionally, the full indexer can be selected, but this takes more time to complete.
12. Select the C/C++ Make Project and perform the below steps:
    – Make Builder tab:

      Deselect the Build on resource save (Auto Build) option.

      Select Stop on first build error option.
    – Environment tab:

      Select the Replace native environment with specified environment radio button.

      Add the environment variables listed in Table 5. (See Figure 4)

Table 5 shows the environment variables that are to be set.

Table 5. Environment Variables to Set

Variable	Value
CROSS_COMPILE	arm-none-linux-gnueabi
PATH	/opt/freescale/usr/local/gcc-4.1.2-glibc-2.5-nptl-3/arm-none-linux-gnueabi/bin/:/usr/local/sbin:/usr/local/bin:/usr/sbin:/usr/bin:/sbin:/bin

Figure 4 shows the environment variables in the Eclipse make builder.

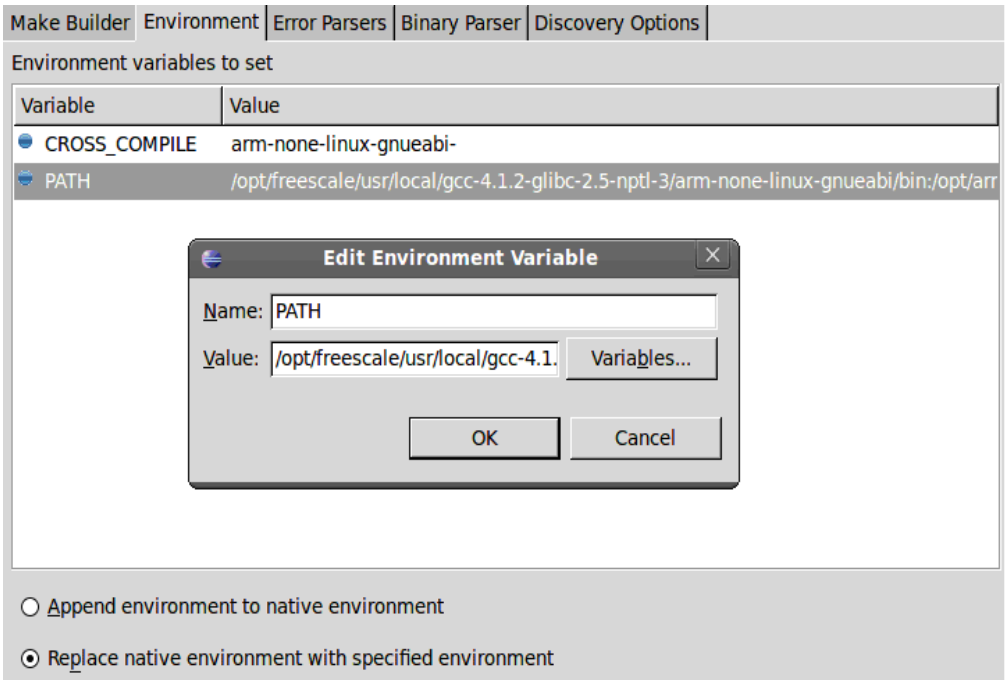

Figure 4. Environment Variables in Eclipse Make Builder

– Binary Parser tab:

Deselect the Elf parser.

Select the GNU Elf Parser and configure the addr2line and c++filt commands as listed in Table 6.

Table 6 shows the GNU binary parser selection.

Table 6. GNU Binary Parser Selection

Binary Parser Options	Value
addr2line	/opt/freescale/usr/local/gcc-4.1.2-glibc-2.5-nptl-3/arm-none-linux-gnueabi/bin/arm-none-linux-gnueabi-addr2line
c++filt	/opt/freescale/usr/local/gcc-4.1.2-glibc-2.5-nptl-3/arm-none-linux-gnueabi/bin/arm-none-linux-gnueabi-c++filt

– Discovery Options tab:

Deselect the Automate discovery of paths and symbols option.

13. Click OK to save and close the properties window.
14. Go to Project > Create Make Target to open a new window. For the U-Boot project, create the make targets listed in Table 7. (See Figure 5)

Table 7 shows the target names and their descriptions.

Table 7. Make Targets to Create

Target Name	Make Target	Description
Dist Clean	distclean	Full clean up of the source tree
i.MX25 3-Stack	mx25_3stack_config	Configure the U-Boot source tree to be built for an i.MX25 3-stack board.
i.MX25 Custom	mx25_custom_config	Configure the U-Boot source tree to be built for a custom i.MX25 based design.

The make targets are used to configure the system for the target board before executing the build process. If the system is not configured, an error is shown as below:

```
Make all
System not configured - see README
Make: *** [all] Error 1
```

Additionally, the Dist Clean target is used to perform a full clean up of the source tree (remove all the resulting files of previous build).

Figure 5 shows the making of the targets in eclipse projects.

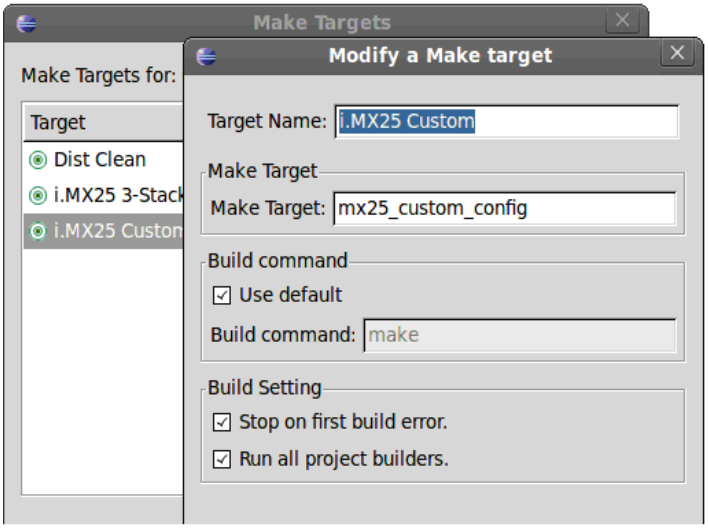

Figure 5. Make Targets in Eclipse Project

After successful configuration of the Eclipse IDE, follow the build steps below:

- Build the Dist Clean make target (optional).
- Configure the system using the desired make target (from the list above).
- Build the project.

After successful build, the output files are placed in the U-Boot source code path. (See Figure 6)

Figure 6 shows the output of the build process at the console.

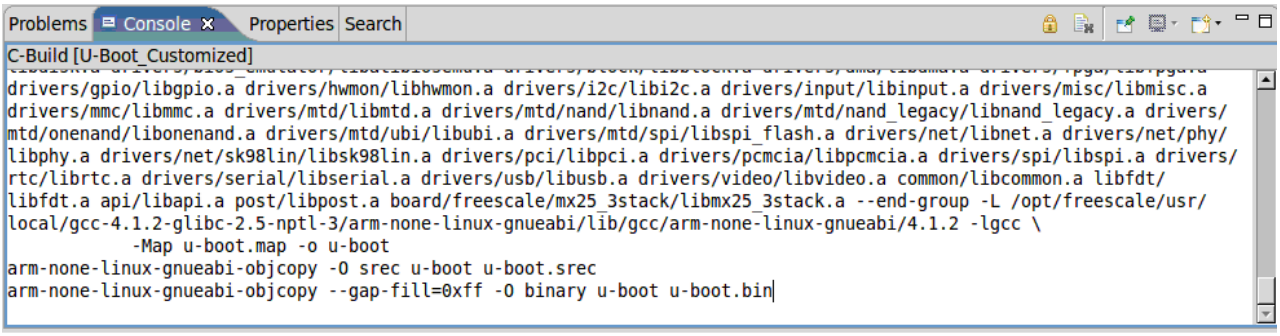

Figure 6. Console Output of Building Process

THIS PAGE INTENTIONALLY LEFT 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
+1-303-675-2140
Fax: +1-303-675-2150
LDCForFreescaleSemiconductor
@hibbertgroup.com

Document Number: AN4171
Rev. 0
07/2010