Freescale Semiconductor
Application Note

Document Number: AN4173
Rev. 0, 07/2010

U-Boot for i.MX51 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 the 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.MX51 Evaluation Kit (EVK) 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 Appendix A, “Configuring Eclipse IDE for U-Boot Development.”

Contents

1 Requirements

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

- Host computer with a Linux Operating System (OS)
- 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
- *MCIMX51 Multimedia Applications Processor Reference Manual* (MCIMX51RM)
- *i.MX51 EVK 1.6 Linux User’s Guide*
- *i.MX51 EVK Hardware User’s Guide*
- 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 Appendix A, “Configuring Eclipse IDE for U-Boot Development.”)

2 U-Boot Project 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 such as 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—Trivial File Transfer Protocol (TFTP), Bootstrap Protocol (BOOTP), Dynamic Host Configuration Protocol (DHCP), and Network File System (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 Multi Media Card (MMC) or Secured Digital (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 sent along with the Linux Board Support Package (BSP) for the i.MX51 EVK. This BSP is present inside the Linux Software Development Kit (SDK). The Linux SDK for the i.MX51 processor and documentation is available at http://www.freescale.com/imx51evk

At the time of creating this application note, the latest available version of Linux SDK was IMX_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 patch 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.MX51 EVK 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.MX51 EVK platform.

NOTE

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

In the U-Boot source code, the acronym BBG is used to refer to the i.MX51 EVK platform (this was the former name of the platform). In addition, there is another i.MX51 platform (3-stack) within the U-Boot source code. This application note focuses only on the EVK but if required, refer to the i.MX51 3-stack code.

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 commands:

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

Table 1 outlines the top-level directory tree and a brief description of each directory.

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 or 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 NIOS architecture
lib_nios2	Files generic to the NIOS2 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 the NAND Flash boot with stage 0 boot loader
net	Networking support (bootp, tftp, rarp, nfs, and so on)

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

Directory	Description
onenand_ipl	One NAND initial program loader
patches	Patches for the i.MX platforms (these are already applied during -prep with LTIB)
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 The i.MX51 Related Source Files

The i.MX51 application processors (based on ARM Cortex-A8) and its development platform EVK are added to the U-Boot project.

Table 3 outlines the files and directories of the i.MX51 processor and a brief description of each directory.

Table 3. i.MX51 Related Source Files

Directory/File	Description
board/freescale/imx51/board-imx51.h	CPLD definitions (not used for the i.MX51 EVK)
board/freescale/imx51/flash_header.S	Image header that is appended to the u-boot.bin file; includes Device Configuration Data (DCD)
board/freescale/imx51/imx51.c	Board and SoC initialization routines in C language
board/freescale/imx51/lowlevel_init.S	Board low-level initialization routines in the assembly language
board/freescale/imx51/u-boot.lds	Linker script
board/freescale/imx51/config.mk	Defines the base address for binary (TEXT_BASE)
cpu/arm_cortexa8/cpu.c	CPU setup code in the C language: interrupts, stack, mmu, and cache setup routines
cpu/arm_cortexa8/start.S	CPU low-level initialization code. The first function executed when the U-Boot starts is defined here
cpu/arm_cortexa8/mx51/crm_regs.h	Clock and reset module register definitions and masks

Table 3. i.MX51 Related Source Files (continued)

Directory/File	Description
cpu/arm_cortexa8/mx51/generic.c	Routines for calculating the CPU and peripheral clocks and a function to call the on-chip Ethernet initialization routine and one function to print CPU information
cpu/arm_cortexa8/mx51/interrupts.c	Starts a timer and provides functions around the timer count. Also, implements the `reset_cpu` function
cpu/arm_cortexa8/mx51/iomux.c	IOMUX setup routines
cpu/arm_cortexa8/mx51/mxc_nand_load.S	Low-level NAND boot support for i.MX51
cpu/arm_cortexa8/mx51/serial.c	On-chip Universal Asynchronous Receiver/Transmitter (UART) driver and serial I/O functions
include/asm-arm/arch-mx51/imx_spi.h	Serial Peripheral Interface (SPI) functions, structures and masks definitions
include/asm-arm/arch-mx51/imx_spi_nor.h	SPI NOR Flash definitions and masks
include/asm-arm/arch-mx51/imx_spi_pmic.h	SPI Power Management Integrated Circuit (PMIC) functions definitions
include/asm-arm/arch-mx51/iomux.h	IOMUX control definitions and functions
include/asm-arm/arch-mx51/mmc.h	Nothing defined in here
include/asm-arm/arch-mx51/mx51.h	On-chip modules base addresses and registers definitions
include/asm-arm/arch-mx51/mx51_pins.h	i.MX51 I/O pin list
include/asm-arm/arch-mx51/mxc_nand.h	NAND Flash Controller (NFC) registers definitions and macros
include/asm-arm/arch-mx51/sdhc.h	Secure Digital Host Controller (SDHC) register definitions and functions
include/configs/imx51.h	i.MX51 EVK 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 the 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 for working with NAND Flash, write and read skipping bad blocks, lock the NAND Flash during accesses, and so on
drivers/mtd/spi/imx_spi_nor_atmel.c	SPI NOR driver for an ATMEL device
drivers/mtd/spi/imx_spi_nor_sst.c	SPI NOR driver for an SST device
drivers/spi/imx_spi.c	On-chip SPI driver
drivers/spi/imx_spi_pmic.c	PMIC driver, defines R/W functions
drivers/mmc/fsl_esdhc.c	Functions to use MMC/SD cards
drivers/mmc/fsl_mmc.c	I/O control access for MMC/SD cards
common/env_mmc.c	Functions to store and retrieve environment variables from MMC/SD card

Table 3. i.MX51 Related Source Files (continued)

Directory/File	Description
common/env_sf.c	Functions to store and retrieve environment variables from SPI Flash
common/cmd_sf.c	Commands for serial (SPI) Flash
common/cmd_spi.c	Commands for SPI control
drivers/net/mxc_fec.c	On-chip Fast Ethernet Controller (FEC) device driver

5 Create a New Board Based on the i.MX51 EVK

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 are stored. By creating a new board directory, the original files that are used as base (in this case, the i.MX51 EVK board) are not changed 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 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.MX51 EVK, 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 `imx51` board directory to a new directory and provide a meaningful name to identify the design. This application note uses `imx51_custom` as a new directory name.

```
cp -r board/freescale/imx51/ board/freescale/imx51_custom
```

3. Copy the contents of the current i.MX51 EVK board configuration file to a new file and provide a meaningful name. This application note uses `imx51_custom.h` as a new file name.

```
cp include/configs/imx51.h include/configs/imx51_custom.h
```

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

```
imx51_custom_config  : unconfig
        @$(MKCONFIG) $(@:_config=) arm arm_cortexa8 imx51_custom freescale mx51
```

NOTE

The U-Boot project developers recommend to add any new board to the `MAKEALL` script 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 more information, refer to the U-Boot `README` file.

5. Adapt to any fixed paths. In this case, the linker script, `imx51_custom/u-boot.lds`, has two paths.

```
board/freescale/imx51_custom/flash_header.o

board/freescale/imx51_custom/libimx51.a
```

6. 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
```

7. Configure the system for the new board:

   ```
   make imx51_custom_config
   ```

8. 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.MX51 EVK 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 needs to be modified accordingly.

7 Boot Modes

The i.MX51 applications processor provides two internal boot modes and these are described in detail in the *MCIMX51 Multimedia Applications Processor Reference Manual* (MCIMX51RM).

The i.MX51 has no external boot modes but provides the following internal boot modes:

- 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 ROM code of the processor 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.MX51 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.
- Internal boot mode (ROM select)—is equivalent to the Internal boot, `BOOT_MODE[1:0] = 00`, with the only difference being General Purpose Input/Output (GPIO) boot override pins are ignored, regardless of the `BT_GPIO_SEL` setting. The boot program uses only the boot eFuse settings. This allows the user to burn fuses on the closed production device with no external muxes on the BOOT_MODE, pull-ups/pull-downs, and no uncertainty of serial downloader is invoked by the unknown boot pin values during the initial boot of the device.

7.1 Flash Header

The Flash header, `board/imx51_custom/flash_header.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.MX 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 *MCIMX51 Multimedia Applications Processor Reference Manual* (MCIMX51RM).

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
```

7.2 Customize SDRAM Initialization

If the SDRAM device is changed in the custom platform, the i.MX51 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 `flash_header.S` file and modify the values of the MXC_DCD_ITEM macros (add/remove values) in accordance with the specification sheet of SDRAM devices and the *MCIMX51 Multimedia Applications Processor Reference Manual* (MCIMX51RM).

The MXC_DCD_ITEM macro transforms an identifier number, address of a register, 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

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

The pads for the SDRAM device needs to be configured before attempting its initialization. Configure the IOMUXC properly with DCD before executing the following step.

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

```
/*-----------------------------------------------------------------------
* Physical Memory Map
*/
#define CONFIG_NR_DRAM_BANKS    1
#define PHYS_SDRAM_1            CSD0_BASE_ADDR
#define PHYS_SDRAM_1_SIZE       (512 * 1024 * 1024)
```

7.3 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 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,
};
```

7.4 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.MX51 EVK board configuration file includes device drivers such as I^{2}C, SPI, 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 is changed in the design. Some examples are described in the following sections.

7.4.1 UART Driver

The current configuration includes the UART driver using the `CONFIG_MX51_UART` constant and selects the UART1 driver using the `CONFIG_MX51_UART1` constant. To remap the UART driver, perform the following steps:

1. Change the `1` used in the `#define CONFIG_MX51_UART1` file with the UART number that is used.
2. Change the IOMUX and pad configuration for the UARTx in the `board/freescale/imx51_custom/imx51.c` file with the new UART number.

```
static void setup_uart(void)
{
unsigned int pad = PAD_CTL_HYS_ENABLE | PAD_CTL_PKE_ENABLE |
PAD_CTL_PUE_PULL | PAD_CTL_DRV_HIGH;
mxc_request_iomux(MX51_PIN_UART1_RXD, IOMUX_CONFIG_ALT0);
mxc_iomux_set_pad(MX51_PIN_UART1_RXD, pad | PAD_CTL_SRE_FAST);
mxc_request_iomux(MX51_PIN_UART1_TXD, IOMUX_CONFIG_ALT0);
mxc_iomux_set_pad(MX51_PIN_UART1_TXD, pad | PAD_CTL_SRE_FAST);
mxc_request_iomux(MX51_PIN_UART1_RTS, IOMUX_CONFIG_ALT0);
mxc_iomux_set_pad(MX51_PIN_UART1_RTS, pad);
mxc_request_iomux(MX51_PIN_UART1_CTS, IOMUX_CONFIG_ALT0);
mxc_iomux_set_pad(MX51_PIN_UART1_CTS, pad);
}
```

7.4.2 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
```

7.4.3 SPI NOR Flash Driver and Commands

The i.MX51 EVK board provides an ATMEL SPI NOR Flash and its driver. To use this driver, the i.MX SPI must be included in the U-Boot build. The following configuration is used to configure the driver and select the slave number for the ATMEL SPI NOR device:

```
#define CONFIG_FSL_SF           1
#define CONFIG_CMD_SPI
#define CONFIG_CMD_SF
#define CONFIG_SPI_FLASH_IMX_ATMEL      1
#define CONFIG_SPI_FLASH_CS     1
#define CONFIG_IMX_SPI
```

7.4.4 NAND Flash Driver and Commands

If the NAND Flash driver is used in the custom design, the driver must be included in the U-Boot build. This is done by setting the `CONFIG_MX51` and `CONFIG_CMD_NAND` options.

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 ID 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"}
};
```

7.4.5 PMIC Driver

The ATLAS Application Processor Light (APL) PMIC is connected to the i.MX51 EVK through a SPI port. Therefore, the PMIC driver uses the i.MX SPI driver. To include this SPI driver and configure the slave for PMIC, use the following configurations:

```
#define CONFIG_IMX_SPI
#define CONFIG_IMX_SPI_PMIC
#define CONFIG_IMX_SPI_PMIC_CS 0
```

Also, if `BOARD_LATE_INIT` is defined in the board configuration file, the `board_late_init` function in the `board/freescale/imx51_custom/imx51.c` file is included and executed. This function calls the `power_init` routine that sets up the ATLAS APL and if required, adapt the code accordingly.

7.5 Miscellaneous Customizations

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

7.5.1 Environment Variables and Auto Boot Command

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

```
      #define CONFIG_BOOTDELAY        3
      #define CONFIG_PRIME    "FEC0"
      #define CONFIG_LOADADDR         0x90800000      /* loadaddr env var */
      #define CONFIG_EXTRA_ENV_SETTINGS                                      \
               "netdev=eth0\0"                                       \
               "ethprime=FEC0\0"                                     \
               "uboot_addr=0xa0000000\0"                             \
               "uboot=u-boot.bin\0"                  \
               "kernel=uImage\0"                             \
               "nfsroot=/opt/eldk/arm\0"                             \
               "bootargs_base=setenv bootargs console=ttymxc0,115200\0"\
               "bootargs_nfs=setenv bootargs ${bootargs} root=/dev/nfs "\
               "ip=dhcp nfsroot=${serverip}:${nfsroot},v3,tcp\0"\
               "bootcmd=run bootcmd_net\0"                           \
               "bootcmd_net=run bootargs_base bootargs_nfs; "        \
               "tftpboot ${loadaddr} ${kernel}; bootm\0"     \
               "prg_uboot=tftpboot ${loadaddr} ${uboot}; "           \
               "protect off ${uboot_addr} 0xa003ffff; "      \
               "erase ${uboot_addr} 0xa003ffff; "            \
               "cp.b ${loadaddr} ${uboot_addr} ${filesize}; "  \
               "setenv filesize; saveenv\0"
```

7.5.2 Change the Board Name and the 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/imx51_custom/imx51.c` file along with the latest root cause of reset. If required, replace the name of the board with a suitable string.

```
int checkboard(void)
{
         printf("Board: MX51 BABBAGE ");
         if (system_rev & CHIP_REV_2_5) {
                  printf("2.5 [");
         } else if (system_rev & CHIP_REV_2_0) {
                  printf("2.0 [");
         } else if (system_rev & CHIP_REV_1_1) {
                  printf("1.1 [");
         } else {
                  printf("1.0 [");
         }
         switch (__REG(SRC_BASE_ADDR + 0x8)) {
         case 0x0001:
                  printf("POR");
                  break;
         case 0x0009:
                  printf("RST");
                  break;
         case 0x0010:
         case 0x0011:
                  printf("WDOG");
                  break;
         default:
                  printf("unknown");
         }
                  printf("]\n");
                  return 0;
}
```

The U-Boot prompt is displayed after all the setup functions are executed. The string displayed at the prompt can be changed in the `include/configs/imx51_custom.h` file using the following definition:

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

7.5.3 Change the Linux Machine Type and Address of ATAGs

When the U-Boot is used to boot a Linux 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 the 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 the parameters, refer to the `board/freescale/imx51_custom/imx51.c` file and modify the following code:

```
gd->bd->bi_arch_number = MACH_TYPE_MX51_BABBAGE;        /* board id for linux */
/* address of boot parameters */
gd->bd->bi_boot_params = PHYS_SDRAM_1 + 0x100;
```

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

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

8 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 of 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 print 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 the cases, the source code needs to be recompiled.

In addition, there are other files that have their own debug macros or functions. In the 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`.

9 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

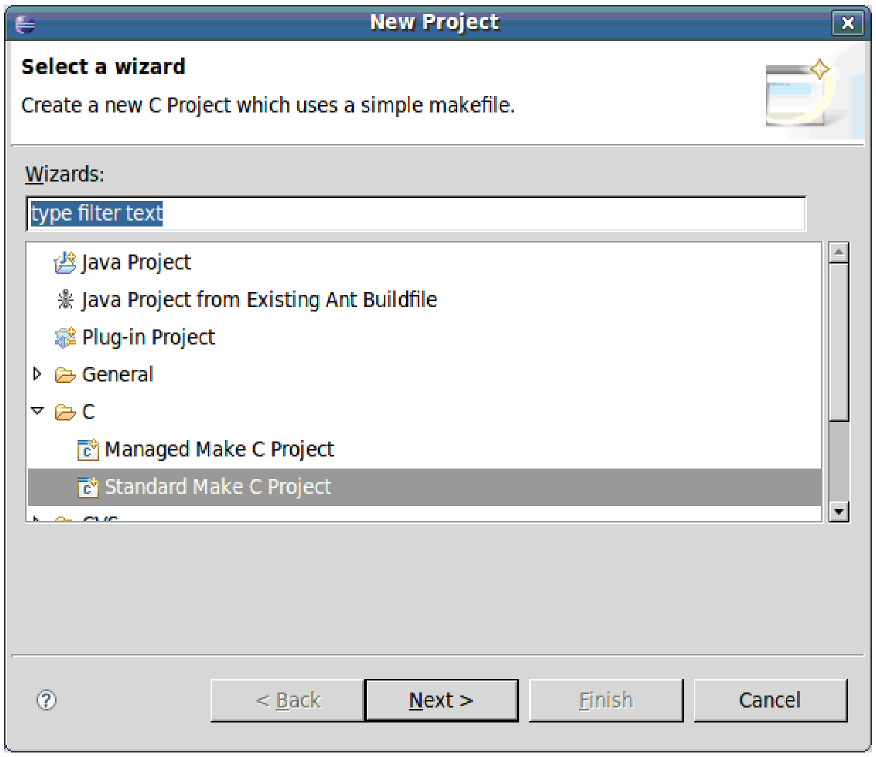
To assist during the source code customization process, it is recommended to set up an Integrated Development Environment (IDE) in the host computer. This section provides the instructions to set-up the Eclipse IDE (for C/C++ developers).

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.


New Project

Select a wizard

Create a new C Project which uses a simple makefile.

Wizards:

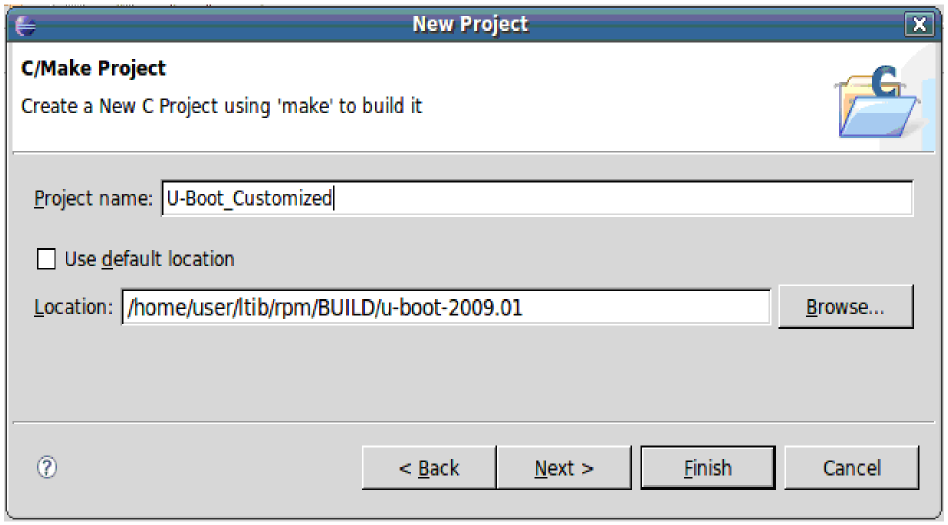
type filter text

- Java Project
- Java Project from Existing Ant Buildfile
- Plug-in Project
- General
- C
  - Managed Make C Project
  - Standard Make C Project

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.


New Project

C/Make Project

Create a New C Project using 'make' to build it

Project name: U-Boot_Customized

[ ] Use default location

Location: /home/user/ltib/rpm/BUILD/u-boot-2009.01

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.

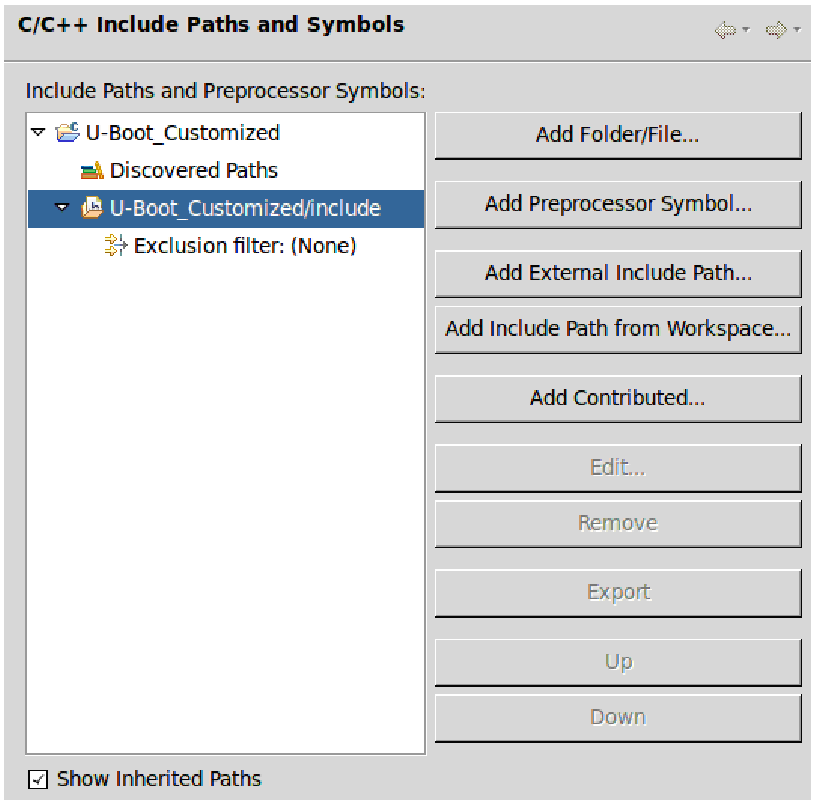

C/C++ Include Paths and Symbols
Include Paths and Preprocessor Symbols:
U-Boot_Customized
Discovered Paths
U-Boot_Customized/include
Exclusion filter: (None)
Add Folder/File...
Add Preprocessor Symbol...
Add External Include Path...
Add Include Path from Workspace...
Add Contributed...
Edit...
Remove
Export
Up
Down
Show Inherited Paths

Figure 3. Include Path from Workspace

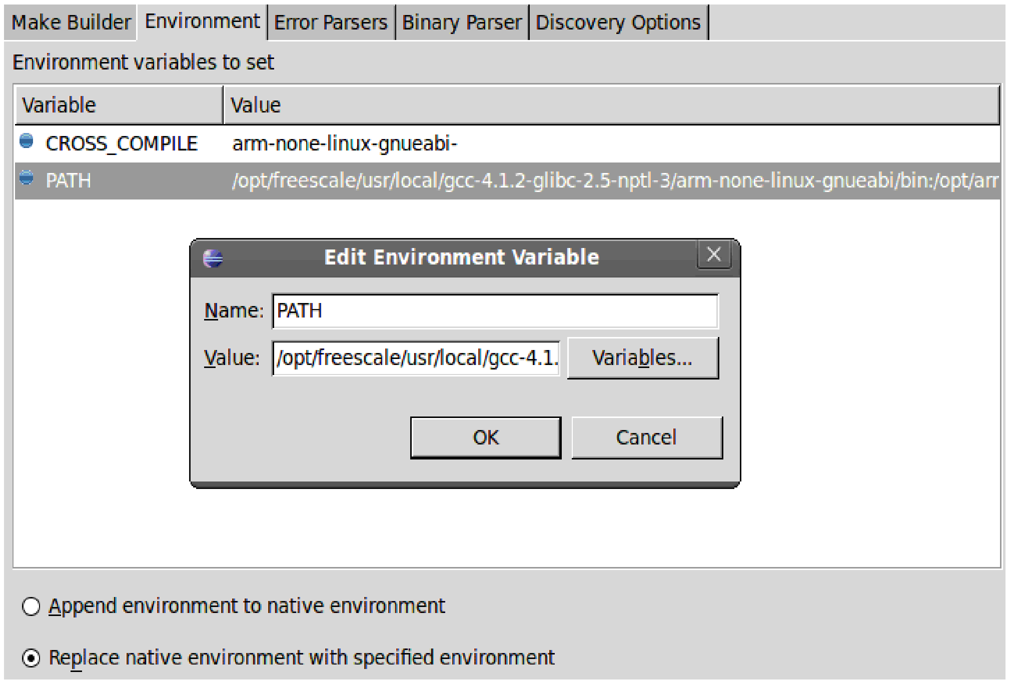
11. Select the C/C++ Indexer and perform the following steps:
    – It is recommended to enable the fast C/C++ Indexer to assist source code navigation. Optionally, select the full indexer, but this takes more time to complete.
12. Select the C/C++ Make Project and perform the following steps:
    – 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.



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:/opt/arr

Name: PATH

Value: /opt/freescale/usr/local/gcc-4.1.

Append environment to native environment [✓ Replace native environment with specified environment]

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

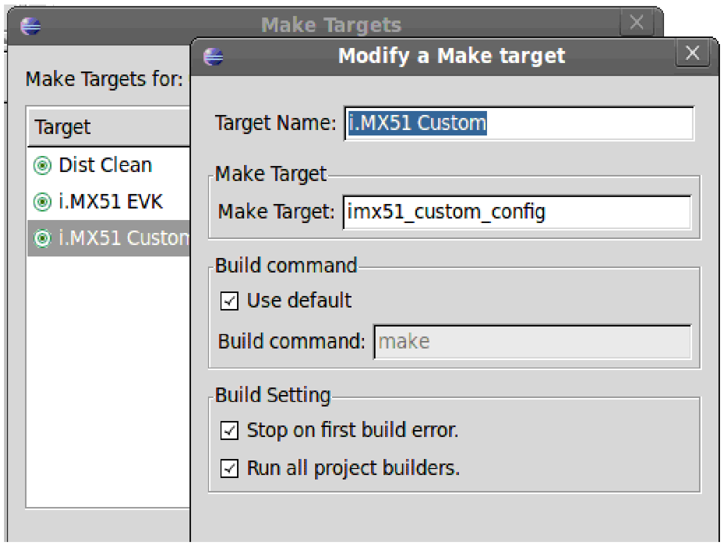
Target Name	Make Target	Description
Dist Clean	distclean	Full clean up of the source tree
i.MX51 EVK	imx51_config	Configure the U-Boot source tree to be built for an i.MX51 EVK board
i.MX51 Custom	imx51_custom_config	Configure the U-Boot source tree to be built for a custom i.MX51 based design

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

```
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.



Target Name: i.MX51 Custom

Make Target

Make Target: imx51_custom_config

Build command

☑ Use default

Build command: make

Build Setting

☑ Stop on first build error.

☑ Run all project builders.

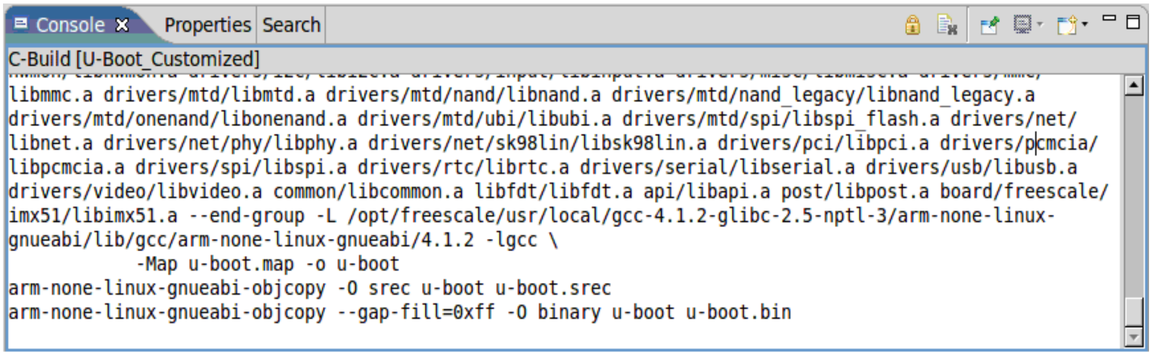
Figure 5. Make Targets in Eclipse Project

After successful configuration of the Eclipse IDE, perform the following steps:

- 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.



```
libmmc.a drivers/mtd/libmtd.a drivers/mtd/nand/libnand.a drivers/mtd/nand_legacy/libnand_legacy.a
drivers/mtd/onenand/libonenand.a drivers/mtd/ubi/libubi.a drivers/mtd/spi/libspi_flash.a drivers/net/
libnet.a drivers/net/phy/libphy.a drivers/net/sk98lin/libsk98lin.a drivers/pci/libpci.a drivers/pcmcia/
libpcmcia.a drivers/spi/libspi.a drivers/rtc/librtc.a drivers/serial/libserial.a drivers/usb/libusb.a
drivers/video/libvideo.a common/libcommon.a libfdt/libfdt.a api/libapi.a post/libpost.a board/freescale/
imx51/libimx51.a --end-group -L /opt/freescale/usr/local/gcc-4.1.2-glibc-2.5-nptl-3/arm-none-linux-
gnueabi/lib/gcc/arm-none-linux-gnueabi/4.1.2 -lgcc \
            -Map u-boot.map -o u-boot
arm-none-linux-gnueabi-objcopy -O srec u-boot u-boot.srec
arm-none-linux-gnueabi-objcopy --gap-fill=0xff -O binary u-boot u-boot.bin
```

Figure 6. Console Output of Building Process

THIS PAGE INTENTIONALLY LEFT BLANK

THIS PAGE INTENTIONALLY LEFT BLANK

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: AN4173
Rev. 0
07/2010