

U-Boot[®] for i.MX35 Based Designs

Source Code Overview and Customization

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

1 Introduction

The Das U-Boot, also known as U-Boot, is a firmware/bootloader for the hardware platforms, which 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 (available under General Public License—GPL), such as device drivers, networking, and file systems support utilities that assist board bring up and testing.

The U-Boot firmware is ported to operate on the i.MX applications processors and development boards. However, the customers should get adapted to some key areas of the source code that makes the U-Boot firmware to operate on a new hardware platform based on the i.MX processor.

In this application note, the reader is guided through the areas of the i.MX35 3-stack U-Boot source code where the adaptation is required. Also, this application note defines guidelines for configuring the Eclipse Integrated

Contents

1	Introduction	1
2	Requirements	2
3	U-Boot Overview	2
4	Getting U-Boot Source Code	3
5	Source Code Tree Overview	4
6	Creating New Board Based on i.MX35 3-Stack	7
7	Customizing the Code	9
8	Enabling Debugging Information	17
9	Revision History	18
A	Configuration of Eclipse IDE for U-Boot Development	19

Development Environment (IDE) for the U-Boot development in [Appendix A, “Configuration of Eclipse IDE for U-Boot Development.”](#)

2 Requirements

The requirements to build a U-Boot project are as follows:

- Host computer with a Linux operating system (OS) and basic knowledge in Linux
- U-Boot source code for the i.MX platforms. Refer to the [Section 4, “Getting U-Boot Source Code,”](#) for information about the U-Boot source code.
- *i.MX35 (MCIMX35) Multimedia Applications Processor Reference Manual (IMX35RM)*
- *i.MX35 PDK 1.6 Linux User’s Guide (926-77208)*
- *i.MX35 PDK Hardware User’s Guide (924-76347)*
- Basic knowledge in C and ARM assembly languages
- Eclipse IDE with C/C++ development plug-in (required if the reader desires to follow the instructions in [Appendix A, “Configuration of Eclipse IDE for U-Boot Development.”](#))

3 U-Boot Overview

The U-Boot is derived from two small bootloaders—PPCBoot and ARMboot. These bootloaders are merged to create the universal bootloader or U-Boot that provides support for an expanded number of processors and boards. Refer to the link, <http://www.denx.de/wiki/U-Boot/WebHome>, to know more about the U-Boot. The source code and documentations are distributed under the GPL license.

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

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

- Bootstraps the hardware platform
- Loads an OS image and transfers control to the OS (for execution)
- Supports network protocols—Trivial File Transfer Protocol (TFTP), Bootstrap Protocol (BOOTP), Dynamic Host Configuration Protocol (DHCP), and Network File System (NFS)
- Supports serial download—s-record and binary (through Kermit)
- Supports Flash management—copy, erase, protect, cramfs, and jffs2
- Supports Flash types—CFI NOR Flash, NAND Flash, and MultiMediaCard/Secure Digital (MMC/SD) cards
- Supports memory utilities—copy, dump, crc, check, and mtest
- Supports IDE and Serial Advanced Technology Attachment (SATA) boot from disk—raw block, ext2, fat, and reiserfs
- Supports interactive shell—choice of simple or busybox shell with many scripting features

For more information about the U-Boot, refer to the link, <http://www.denx.de/wiki/U-Boot/WebHome>.

4 Getting U-Boot Source Code

The U-Boot source code is shipped along with the Linux Board Support Package (BSP) for the i.MX35 Platform Development Kit (PDK). The BSP is embedded in the Linux Software Development Kit (SDK). The Linux SDK for the i.MX35 and documentations are available at www.freescale.com/imx35pdk.

NOTE

The latest available Linux SDK is IMX35_SDK16_LINUX_BSP that contains the BSP based on the Linux kernel version 2.6.28. To install the Linux BSP in the host computer, check the accompanying documents.

Some important points with respect to the U-Boot source code are as follows:

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

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

```
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 sbuild -p u-boot
```

Executing this command configures the U-Boot for the i.MX35 3-stack platform and the following binaries are generated:

- `u-boot`—is a file in Executable and Linkable Format (ELF) with symbols and debugging information.
- `u-boot.bin`—is a plain binary file and is programmed to a boot media (NAND, NOR, SD, and so on) to bootstrap the i.MX35 3-stack board.

NOTE

The term 3-stack board is used to describe an i.MX development platform that consists 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 before the code customization.

5 Source Code Tree Overview

The U-Boot source code organization is similar to the Linux kernel source code organization. In this section, an overview of the source code tree is presented. [Table 1](#) shows the top-level directory tree and a brief description of each directory.

The command to list the directory tree is as follows:

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

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

Directory	Description
api	U-Boot machine/arch independent API for the external applications
api_examples	Example applications that uses the API
board	Board dependant files/directories
common	Misc architecture independent functions
cpu	CPU specific files
disk	Code for the 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_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 Scalable Processor Architecture (SPARC) architecture
nand_spl	Support for the NAND Flash boot with stage 0 bootloader

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

Directory	Description
net	Networking support (bootp, tftp, rarp, nfs, and so on)
onenand_ipl	OneNAND initial program loader
patches	Patches for the i.MX platforms (already applied during the command, <code>./ltib -m prep -p u-boot</code>)
post	Power on self test
tools	Tools for building S-Record files, U-Boot images, and so on

The files listed in [Table 2](#) are also in the top-level directory.

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 <code>Makefile</code> is used while executing the board configuration and build processes. The new board configurations should be added to this file.
MAKEALL	This script in this file is used to configure and build all the supported boards in one step. The list of boards in this file should 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 for the U-Boot source code.

5.1 i.MX35 Related Source Files

The i.MX35 applications processor (based on ARM1136JF-S) and its development platform (3-stack board) are added to the U-Boot. [Table 3](#) shows the files and directories related to the i.MX35 porting and their description.

Table 3. i.MX35 3-Stack Related Source Files

Directory/File	Description
board/freescale/mx35_3stack/board-mx35_3stack.h	The i.MX35 3-stack board definitions including the CPLD and SDRAM configuration constants
board/freescale/mx35_3stack/flash_header.S	Image header that can be appended to the <code>u-boot.bin</code> file. The file includes Device Configuration Data (DCD)
board/freescale/mx35_3stack/lowlevel_init.S	Board low-level initialization routines in the assembly language
board/freescale/mx35_3stack/mx35_3stack.c	More board routines in the C language
board/freescale/mx35_3stack/config.mk	Defines the base address for binary (<code>TEXT_BASE</code>).
board/freescale/mx35_3stack/u-boot.lds	Linker script
cpu/arm1136/cpu.c	CPU specific code in the C language

Table 3. i.MX35 3-Stack Related Source Files (continued)

Directory/File	Description
cpu/arm1136/start.S	Includes the CPU low-level initialization code. The first function executed when the U-Boot starts is defined in this file
cpu/arm1136/mx35/crm_regs.h	The i.MX35 clock and reset module (CRM) definitions
cpu/arm1136/mx35/generic.c	Routines to calculate the CPU and peripheral clocks and a function to call the on-chip Ethernet initialization routine
cpu/arm1136/mx35/interrupts.c	Starts a timer, and provides functions around the timer count. The file also implements the <code>reset_cpu</code> function
cpu/arm1136/mx35/iomux.c	The i.MX35 IOMUX and GPIO setup functions
cpu/arm1136/mx35/mxc_nand_load.S	Low-level NAND boot support for the i.MX35 3-stack
cpu/arm1136/mx35/serial.c	On-chip UART driver and serial I/O functions
include/asm-arm/arch-mx35/iomux.h	IOMUX control definitions and functions
include/asm-arm/arch-mx35/mmc.h	Nothing is defined in this file
include/asm-arm/arch-mx35/mx35.h	On-chip modules base addresses and register definitions
include/asm-arm/arch-mx35/mx35_pins.h	The i.MX35 I/O pin list
include/asm-arm/arch-mx35/mxc_nand.h	NAND Flash Controller (NFC) register definitions
include/asm-arm/arch-mx35/sdhc.h	Secure Digital Host Controller (SDHC) register definitions and functions
include/configs/mx35_3stack.h	The i.MX35 3-stack board high-level configuration
include/configs/mx35_3stack_mmc.h	The i.MX35 3-stack board high-level configuration (for MMC boot)
lib_arm/board.c	Implements high-level board initialization functions and allows the user to configure the initialization sequence
drivers/mtd/cfi_flash.c	Implements a Common Flash Interface (CFI) driver for the NOR Flashes
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 the NAND Flash
drivers/mtd/nand/nand_ids.c	NAND Flash chip ID list
drivers/mtd/nand/nand_util.c	Utilities to work with the NAND Flash, write and read skipping bad blocks, and lock the NAND Flash during accesses
drivers/mmc/fsl_esdhc.c	Functions to use the MMC/SD cards
drivers/mmc/fsl_mmc.c	I/O control access for the MMC/SD cards

Table 3. i.MX35 3-Stack Related Source Files (continued)

Directory/File	Description
common/env_mmc.c	Functions to store and retrieve the environment variables from the MMC/SD card
drivers/i2c/mxc_i2c.c	I ² C driver for the i.MX architecture
drivers/net/smc911x.c	SMSC911x Ethernet device driver (used for Standard Microsystems Corporation—SMSC LAN9217)
drivers/net/mxc_fec.c	On-chip Fast Ethernet Controller (FEC) device driver

6 Creating New Board Based on i.MX35 3-Stack

In the process of adapting the 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 configuration are contained. In this way, the original files that are used as base (in this case, the i.MX35 3-stack board) remain untouched and available for comparison. If device drivers or another non-board specific code is to be adapted, it is a good practice to make a backup copy of the original code and to have it available in the source tree for comparison. Refer to [Appendix A, “Configuration of Eclipse IDE for U-Boot Development,”](#) to configure Eclipse IDE before proceeding with the following sections.

The steps to create a new board based on the i.MX35 3-stack are as follows:

1. Clean the source code tree to delete the output files of the previous build:


```
make distclean
```
2. Copy the current `mx35_3stack` board directory to a new directory with a meaningful name to identify the design. In this application note, the directory is named as `mx35_custom`:


```
cp -r board/freescale/mx35_3stack/ board/freescale/mx35_custom
```
3. Rename the `board-mx35_3stack.h` and `mx35_3stack.c` files accordingly:


```
mv board/freescale/mx35_custom/board-mx35_3stack.h
board/freescale/mx35_custom/board-mx35_custom.h
mv board/freescale/mx35_custom/mx35_3stack.c board/freescale/mx35_custom/mx35_custom.c
```
4. Adjust the `board/freescale/mx35_custom/Makefile` file to fit the new file name:

Change the line: `COBJS := mx35_3stack.o` for:

```
COBJS := mx35_custom.o
```
5. Adjust the `flash_header.S` and `lowlevel_init.S` files to fit the new file name:

Change the line, `#include "board-mx35_3stack.h"`, to `#include "board-mx35_custom.h"`

- Copy the current i.MX35 3-stack board configuration files and give meaningful names. In this application note, the names, `mx35_custom.h` and `mx35_custom_mmc.h`, are used:

```
cp include/configs/mx35_3stack.h include/configs/mx35_custom.h
cp include/configs/mx35_3stack_mmc.h include/configs/mx35_custom_mmc.h
```

NOTE

The configuration files, `mx35_custom.h` and `mx35_custom_mmc.h`, are not required to be copied if only one boot media is used. In this case, select the configuration file that most suits the requirements and create a new file. These configuration files are further explained in [Section 7.2, “Board Configuration File Differences.”](#)

- Create two entries in the top-level directory, `Makefile`, for the new custom board configurations. These files are arranged in the alphabetical order:

```
mx35_custom_config      \
mx35_custom_mmc_config: unconfig
@$(MKCONFIG) $(@:_config=) arm arm1136 mx35_custom freescale mx35
```

NOTE

The U-Boot project developers recommend adding new board to the `MAKEALL` script and running this script to verify if the new code has not broken any other platform build. This is necessary if a patch is planned to be submitted back to the U-Boot community. For more information, refer to the U-Boot `README` file.

- Adopt any fixed paths. In this case, the linker script, `mx35_custom/u-boot.lds`, has two paths. Replace `mx35_3stack` with `mx35_custom`:

```
board/freescale/mx35_custom/flash_header.o
board/freescale/mx35_custom/libmx35_custom.a
```

- 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
```

- Configure the system for the new board. Use either the `mx35_custom_config` or `mx35_custom_mmc_config` board configuration depending on the boot media to be used:


```
make mx35_custom_config
```

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


```
make
```

The new board is an exact copy of the i.MX35 3-stack board. The next step is to adapt some portions of the code to make the code suitable for the new hardware design. See [Section 7, “Customizing the Code,”](#) for the guidelines to customize the code.

7 Customizing the Code

In this section, examples for the key places within the source code are described where customization is required.

NOTE

Depending on the design and requirements, the described code should be modified.

7.1 Internal Boot vs. External Boot

The i.MX35 applications processor provides different boot modes. These boot modes are described in the *i.MX35 (MCIMX35) Multimedia Applications Processor Reference Manual (IMX35RM)*. A brief introduction to the internal and external boot modes is given as follows:

- Internal boot mode—allows selection of all boot sources (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 ROM code samples the boot pins or eFuses and loads the first set of codes from the selected boot media. These codes should have a Flash header at a particular offset (varies depending on the boot source). The Flash header stores the application information in a specific structure. The Flash header can also store the DCD, which is a block of data processed by the i.MX35 to configure the hardware during the 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.MX35 samples the boot pins or eFuses and moves directly to the base address of the selected boot source (base address of the NFC buffer in the case of NAND Flash). The Flash header is not required to identify the application, and the hardware configuration is carried out by the loaded application.

7.2 Board Configuration File Differences

In [Section 6, “Creating New Board Based on i.MX35 3-Stack,”](#) the following board configuration files are created:

- `include/configs/mx35_custom.h`
- `include/configs/mx35_custom_mmc.h`

The configurations stored in these files impact the way the U-Boot source codes are compiled and how the board operates when the binary in these files are executed during boot up. The differences between these two board configuration files are as follows:

- The `mx35_custom.h` file enables the NAND and NOR Flash drivers. This file does not include any Flash header. Therefore, the U-Boot binary can boot the board in external boot mode only, either from the NOR or NAND Flash.
- The `mx35_custom_mmc.h` file enables the MMC card driver, MMC commands, and Flash header and allows the file to boot from the MMC card (should use the internal boot). This file also includes the NAND and NOR device drivers.

7.3 Flash Header

When the final device requires the i.MX35 external boot to be disabled (through the eFuse, `DIR_BT_DIS`), the only option to boot the board is to use the internal boot mode. In this mode, a Flash header is required to identify the application. Therefore, the following definitions should be present in the board configuration file:

- Include the device driver for the boot media (MMC, NOR, or NAND).
- Add the Flash header and barker (identifier used by the i.MX35 ROM code):


```
#define CONFIG_FLASH_HEADER      1//Includes the Flash header to the binary
#define CONFIG_FLASH_HEADER_BARKER 0xB1//Identifier
```
- For NAND or MMC Flash, configure the offset at:


```
#define CONFIG_FLASH_HEADER_OFFSET 0x400//Required to be set if the Flash header is used
```
- For NOR Flash, configure the offset at:


```
#define CONFIG_FLASH_HEADER_OFFSET 0x1000//Required to be set if the Flash header is used
```
- Rebuild the U-Boot source to generate the `u-boot.bin` file.

Also, if DCD is used and the SDRAM initialization is performed by the DCD data, the user can set the configuration, `#define CONFIG_SKIP_RELOCATE_UBOOT`, to disable the U-Boot relocation to RAM (as the U-Boot relocation is already performed by the i.MX ROM code).

7.4 Customizing SDRAM Initialization

If the SDRAM device changes in the custom platform, then the i.MX35 enhanced SDRAM controller and initialization sequence code should adapt to operate with the new device. The portions of the code that should be focused to modify such initialization sequence are as follows:

- If a U-Boot binary with the Flash header (including the DCD data) is used—Open the `flash_header.s` file and modify the values of the `DCDGEN` macros (or add/remove values) in accordance with the SDRAM device specification sheet and *i.MX35 (MCIMX35) Multimedia Applications Processor Reference Manual (IMX35RM)*. The `DCDGEN` macro transforms the identifier number, register address, value to be written into this register, and length of the access to the corresponding data to be appended into the U-Boot binary. The format of the DCD data is described thoroughly in the *i.MX35 (MCIMX35) Multimedia Applications Processor Reference Manual (IMX35RM)*.

NOTE

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

- If external boot (without DCD) is used—Open the `lowlevel_init.s` file and modify the assembly code of the `setup_sdram` and `setup_sdram_bank` macros, if necessary. Generally, a fine-tuning of the SDRAM timing parameters is the primary customization. The code in the `lowlevel_init.s` file uses the constant values declared in the `board-mx35_3stack.h` file and is recommended to modify the referenced values (`ESDCTL_*`) instead of writing a value directly into the assembly instruction.

If the chip select of the SDRAM or SDRAM size is changed, the following values should be adjusted in the board custom configuration file:

```
#define PHYS_SDRAM_1          CSD0_BASE_ADDR
#define PHYS_SDRAM_1_SIZE    (128 * 1024 * 1024)
```

NOTE

Here, the i.MX35 3-stack U-Boot is using only half of the available physical memory. The board has a total memory capacity of 256 Mbytes that consists of four 16-bit memory chips mapped with two separate chip selects. The two chip select configurations are not described in this application note.

7.5 Necessity of CPLD Code

The i.MX PDKs (3-stack) contain a debug board, and the glue logic in the debug board is implemented in the Complex Programmable Logic Device (CPLD). This device is memory mapped to the on-chip Wireless External Interface Module (WEIM) at chip select 5 (CS5) and provides the following features:

- 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 Universal Asynchronous Receiver Transmitter (UART) controller
- Level shift for the Ethernet and UART signals
- Control and status registers for the various board functions

List of situations to be checked for the necessity of the CPLD code are as follows:

- When the new board does not have the CPLD or memory range of the CS5 (0xB600_0000–0xB7FF_FFFF) is used for different purposes, the code targeting the CPLD should be removed to avoid possible errors. This point can be explained with the following example. Consider a case where the new board has an external Ethernet controller directly attached to the i.MX processor at the CS5. When the CPLD initialization code is executed, the external Ethernet chip initialization gets corrupted and prevents the Ethernet controller device driver from operating correctly.
- When the CS5 is required in the new board for a different purpose, then the CS5 initialization should be kept and only the CPLD (alias debug board or peripheral bus controller) initialization should be removed.
- When the CS5 is not used in the new board, then the code should not be executed. In such cases, both the CS5 initialization and CPLD initialization should be removed.
- To remove or comment out the CS5 and/or CPLD code in the U-Boot source files, look for references to the keywords—CS5, DBG, and PBC. Also, look for other references to the CS5 address space. For example, in the i.MX35 custom board, if CS5 is not required comment out the following sets of codes:

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

```
//WEIM config-CS5 init
DCDGEN(1, 4, 0xB8002054, 0x444a4541)
```

```

DCDGEN(1_1, 4, 0xB8002050, 0x0000dcf6)
DCDGEN(1_2, 4, 0xB8002058, 0x44443302)

//WEIM config-CS5 init
DCDGEN(1, 4, 0xB8002050, 0x0000d843)
DCDGEN(1_1, 4, 0xB8002054, 0x22252521)
DCDGEN(1_2, 4, 0xB8002058, 0x22220a00)

```

NOTE

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

- In the `lowlevel_init.s` file, remove the `init_debug_board` macro and references to the macro:

```

/* CPLD on CS5 setup */
.macro init_debug_board
ldr r0, =DBG_BASE_ADDR
ldr r1, =DBG_CSCR_U_CONFIG
str r1, [r0, #0x00]
ldr r1, =DBG_CSCR_L_CONFIG
str r1, [r0, #0x04]
ldr r1, =DBG_CSCR_A_CONFIG
str r1, [r0, #0x08]
.endm /* init_debug_board */

```

- When the CS5 is required to be initialized and the code interacting with the CPLD logic is not required, then look for the accesses made to the CS5 address space. For example, see the constants below (this code is not found in the current i.MX35 3-stack U-Boot code. However, this 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)

```

7.6 Board Initialization Sequence

The `start_armboot` function executes the board initialization sequence as part of the U-Boot boot-up process. This sequence defines the order in which other routines are to be called and is user customizable. To adapt this sequence, 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,    /* initialize baudrate settings */
    serial_init,      /* serial communications setup */
    console_init_f,   /* stage 1 init of console */
    display_banner,   /* say that we are here */
#ifdef CONFIG_DISPLAY_CPUINFO
    print_cpuinfo,    /* display cpu info (and speed) */
#endif

```

```

#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,
};

```

7.7 Include, Exclude or Remap Device Drivers

After the build, the U-Boot binary should only include the code that is useful to the target board. The i.MX35 3-stack board configuration files should include device drivers for the on-chip and off-chip peripherals such as I²C, UART, FEC, NAND, NOR, MMC, SMSC Ethernet controller, and so on. In the U-Boot customization process, the included drivers in the custom board configuration files are required to be reviewed and verified if all the files are required in the design. The device drivers should be included, excluded or remapped in case the base address changes the design. This is described with examples in the following sections.

7.7.1 UART Driver

The current UART driver uses the `UART1_BASE_ADDR` constant as the base address. In case a different UART is required to be used, adjust the definition, `#define CONFIG_MX35_UART UART1_BASE_ADDR`, in the board configuration file to remap this driver. Check the available UARTs in the i.MX35 System On Chip (SoC) and use a definition from the `mx35.h` file.

7.7.2 SMSC Ethernet Driver

As described in [Section 7.6, “Board Initialization Sequence,”](#) the SMSC LAN9217 device located in the debug card is interfaced through the CPLD logic and therefore, mapped out at an offset within the CS5 (check CPLD memory map). In the case of the SMSC device, this offset is 0. Therefore, the CS5 base address is same for the SMSC driver. If the SMSC LAN9217 or a compatible device is not present in the new board, then the driver code should be excluded from the U-Boot build. If the SMSC LAN9217 or a compatible device is present in the new design and the base address of the device is changed, then a driver remap is required. For the driver remap, adjust the following definitions in the custom board configuration files:

```

/*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, modify the multiple Ethernet interface definitions to a suitable value. The U-Boot uses the following configurations to estimate the number of Ethernet devices present in the system:

```

#define CONFIG_HAS_ETH1
#define CONFIG_NET_MULTII 1

```

7.7.3 MMC Driver and Commands

Depending on the requirement, the MMC device driver can be included or excluded from the U-Boot build. For that, add or remove the following definitions in 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
```

7.7.4 NOR Flash Driver and Commands

The NOR Flash driver (Common Flash Interface—CFI) and Flash commands are included when the following definitions are used. In this case, CS0 is the base of the NOR Flash:

NOTE

The user should 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 */
```

7.7.5 NAND Flash Driver and Commands

When the `CONFIG_MX35` and `CONFIG_CMD_NAND` macros are defined, the NAND Flash driver and commands are included in the U-Boot build. However, disabling the `CONFIG_MX35` macro impacts other functionalities. Therefore, it is recommended to create a specific macro for the NAND low-level driver (`mx35_nand.c`). Hence, the `CONFIG_MX35` macro can be enabled or disabled like the other drivers.

It is important to highlight the location of the NAND chip ID definition for the NAND driver and MTD subsystem. This helps to add a new NAND manufacturer or device ID to the list of supported NANDs.

The structure in the `drivers/mtd/nand/nand_ids.c` file adds new NAND manufacturer IDs to the list of supported NANDs:

```
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"}
};
```

7.7.6 I²C Driver

The I²C communication channel is used to interface with the Power Management IC (PMIC) in the i.MX35 3-stack board. In this board, the I²C port 1 is used with the base address, `0x43F80000`. If the PMIC is relocated to an another I²C port or the PMIC is changed, the codes at the following locations should be changed:

- `include/configs/mx35_custom.h` OR `include/configs/mx35_custom_mmc.h`:


```
#define CONFIG_CMD_I2C
#define CONFIG_HARD_I2C      1
#define CONFIG_I2C_MXC      1
#define CONFIG_SYS_I2C_PORT      I2C_BASE_ADDR
#define CONFIG_SYS_I2C_SPEED      100000
#define CONFIG_SYS_I2C_SLAVE      0xfe
```
- `board/freescale/mx35_custom/mx35_custom.c` (inside the `board_init` function):


```
/* setup pins for I2C1 */
mxc_request_iomux(MX35_PIN_I2C1_CLK, MUX_CONFIG_SION);
mxc_request_iomux(MX35_PIN_I2C1_DAT, MUX_CONFIG_SION);

mxc_iomux_set_pad(MX35_PIN_I2C1_CLK, pad);
mxc_iomux_set_pad(MX35_PIN_I2C1_DAT, pad);
```

Also, if the `BOARD_LATE_INIT` macro is defined in the board configuration file, the `board_detect` and `board_late_init` functions in the `board/freescale/mx35_custom/mx35_custom.c` file should be included. These functions perform the read and write operations for the PMIC registers. If the PMIC I²C port is remapped, then these two functions work properly. However, if the PMIC is changed, these functions do not work. Therefore, the user should adapt or exclude this code from the U-Boot build.

7.8 Miscellaneous Customizations

This section describes the various types of customizations.

7.8.1 Environment Variables and Auto Boot Command

The U-Boot shell allows the user to set environment variables like the Linux shell. These variables can be defined at the U-Boot prompt using the `setenv` command or hardcoded into the source code before the compilation. The `bootcmd` variable in this variable set is executed automatically when the auto boot feature is enabled. To configure these elements, go to the board configuration files and adapt the following code:

```
#define CONFIG_BOOTDELAY          3
#define CONFIG_LOADADDR          0x80800000      /* loadaddr env var */
#define CONFIG_EXTRA_ENV_SETTINGS \
    "netdev=eth0\0" \
    "ethprime=smc911x\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"
```

7.8.2 Changing Board Name and U-Boot Prompt

While the U-Boot is booting, some debug messages are displayed in the console before the prompt is reached. One of these messages is the board name. This is displayed when the `checkboard` function in the `board/freescale/mx35_custom/mx35_custom.c` file is executed along with the root cause of the most recent reset.

NOTE

The board name can be replaced with a suitable string.

The `checkboard` function is given as follows:

```
int checkboard(void)
{
    printf("Board: MX35 3STACK [");
    switch (__REG(CCM_BASE_ADDR + CLKCTL_RCSR) & 0x0F) {
        case 0x0000:
            printf("POR");
            break;
        case 0x0002:
            printf("JTAG");
            break;
    }
}
```



```

        case 0x0004:
            printf("RST");
            break;
        case 0x0008:
            printf("WDT");
            break;
        default:
            printf("unknown");
    }
    printf("]\n");
    return 0;
}

```

The U-Boot prompt is displayed in the shell when all the setup functions are executed. The string displayed with the prompt can be changed in the `include/configs/mx35_custom.h` file with the macro definition,

```
#define CONFIG_SYS_PROMPT      "MX35 U-Boot > "
```

7.8.3 Changing Linux Machine Type and ATAG Address

When the U-Boot is used to boot the Linux OS, the kernel parameters are placed in a special memory area in ATAG format (if this feature is enabled in the board configuration file). The address of this memory location is user-configurable, and one of the parameters passed to the kernel is the machine type. This parameter is a number used to identify the board and should match between the Linux and U-Boot. Otherwise, the Linux kernel does not boot.

To change these parameters, modify the following lines of code in the `board/freescale/mx35_custom/mx35_custom.c` file:

```
gd->bd->bi_arch_number = MACH_TYPE_MX35_3DS;    /* board id for linux */
gd->bd->bi_boot_params = 0x80000100;          /* address of boot parameters */

```

The definitions in the board configuration file that enables the ATAGs are as follows:

```
#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 Enabling Debugging Information

Debugging is the most time-consuming activity while customizing the U-Boot. To have information that detects the root cause of errors is helpful in debugging. For this purpose, the U-Boot source code contains several functions or macros that print extra information in the console during the run-time. This can be described in the following instances:

- In the `include/common.h` file, two debug macros are defined. When the `#define DEBUG` macro definition is set in this file, all the files that include the `common.h` file use the `debug(fmt, args...)` or `debugX(level, fmt, args...)` macros to print the additional information. To reduce the information that is displayed, enable the `#define DEBUG` macro definition in a particular file or files before including the `common.h` file. In both cases, the source code should be recompiled.
- There are other files, such as MTD subsystem and NAND driver, that have their own debug macros or functions. In the MTD subsystem, `#define CONFIG_MTD_DEBUG` and a debug level is used to print

Revision History

the additional information. Other such files are the Serial Peripheral Interface (SPI) subsystem, I²C subsystem, and Journalling Flash File System 2 (JFFS2) subsystem that use the macros `DEBUG_SPI`, `DEBUG_I2C`, and `DEBUG_JFFS2`, respectively.

9 Revision History

Table 4 provides a revision history for this application note.

Table 4. Document Revision History

Rev. Number	Date	Substantive Change(s)
0	03/07/2003	Initial release.

Appendix A Configuration of Eclipse IDE for U-Boot Development

To assist the source code customization process, IDE should be set up in the host computer. In this section, the instructions to set-up Eclipse IDE (for C/C++ developers) is given. Refer to the link, <http://www.eclipse.org/cdt/>, for the information regarding the installation of Eclipse in the host computer.

Once the Eclipse IDE is installed in the Linux host, the steps to configure the Eclipse IDE for the U-Boot development are as follows:

1. Open the Eclipse window, and click File > New > Project.
2. In the new project wizard, select C > Standard Make C Project as shown in [Figure 1](#) and click Next.

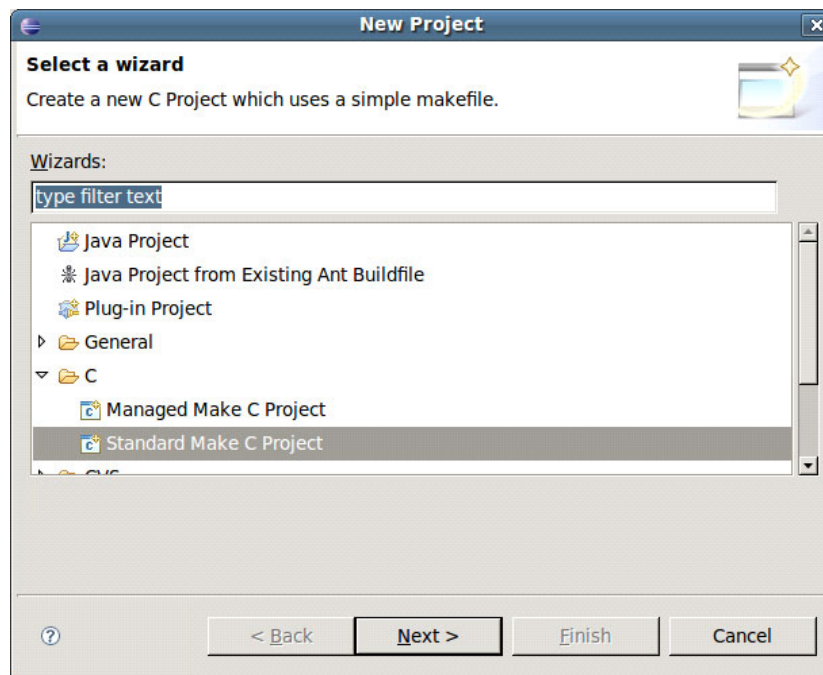


Figure 1. Eclipse IDE New Project Wizard

At the end of this step, the C/Make Project window appears.

3. In the C/Make Project window, type a project name in the box and un-check the Use default location checkbox as shown in [Figure 2](#).

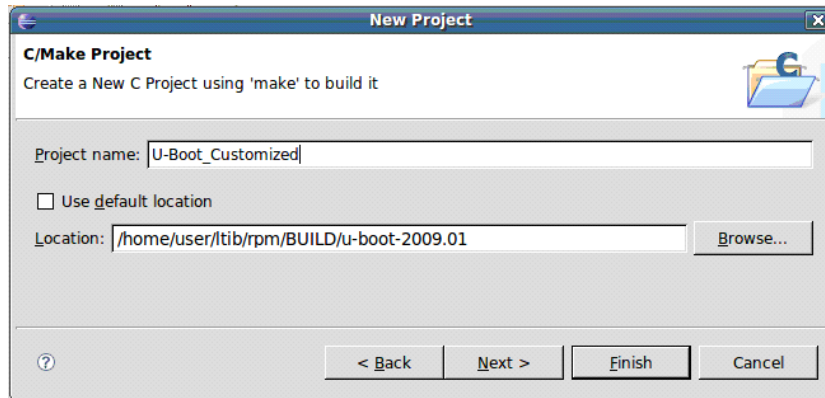


Figure 2. Project Name and Location

4. Click on Browse to search for the path where the U-Boot source code is located. Then, click on Finish to close the wizard.
5. In the Eclipse main window, deselect Project > Build automatically.
6. To configure the project properties, click Project > Properties to open the Properties window:
 - a) Select C/C++ Include Paths and Symbols, and do the following steps:
 - Disable all the automatically discovered paths and symbols (multiple selection is allowed to disable all at once).
 - Click Add include path from workspace. Then, click on U-Boot_Customized/include as shown in [Figure 3](#).

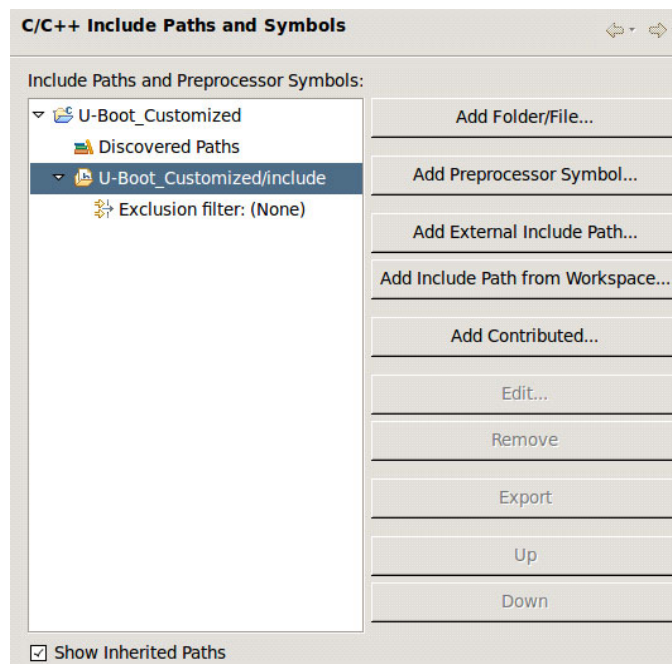
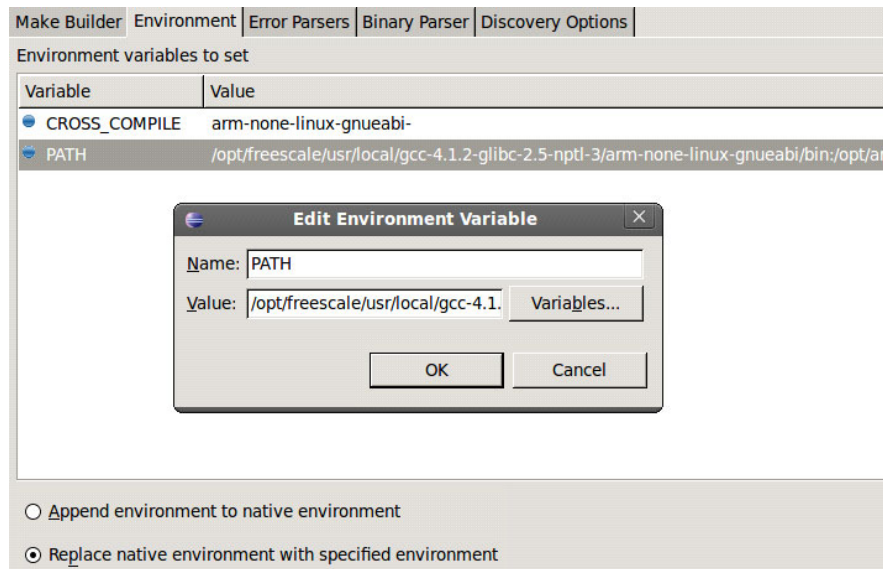


Figure 3. Include Path from Workspace

- b) Select C/C++ Indexer, and perform the following step:
- It is recommended to enable Fast C/C++ Indexer for the assisted source code navigation. The full indexer can be selected optionally. However, this takes more time to complete.
- c) Select C/C++ Make Project, and perform the following steps:
- In the Make Builder tab, deselect the Build on resource save (Auto Build) option and select the Stop on first build error option.
 - In the Environment tab, select Replace native environment with specific environment and add the environment variables listed in [Table 5](#) as shown in [Figure 4](#).

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. Environment Variables in Eclipse Make Builder

- In the Binary Parser tab, deselect Elf parser and select the GNU Elf Parser. Configure the addr2line and c++filt commands as listed in [Table 6](#).

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

- In the Discovery Options tab, deselect the Automate discovery of paths and symbols option.
- 7. Click OK to save and close the Properties window.
- 8. Click Project > Create Make Target to open a new window. For this U-Boot project, create the make targets listed in [Table 7](#) as shown in [Figure 5](#).

Table 7. Make Targets to Create

Target Name	Make Target	Description
Dist Clean	distclean	Full clean up of the source tree
i.MX35 3-Stack	mx35_3stack_config	Configures the U-Boot source tree to be built for the i.MX35 3-stack board
i.MX35 3-Stack MMC Boot	mx35_3stack_mmc_config	Configures the U-Boot source tree to be built for the i.MX35 3-stack board with the MMC boot enabled
i.MX35 Custom	mx35_custom_config	Configures the U-Boot source tree to be built for a custom i.MX35 based design
i.MX35 Custom MMC Boot	mx35_custom_mmc_config	Configures the U-Boot source tree to be built for a custom i.MX35 based design with the MMC boot enabled

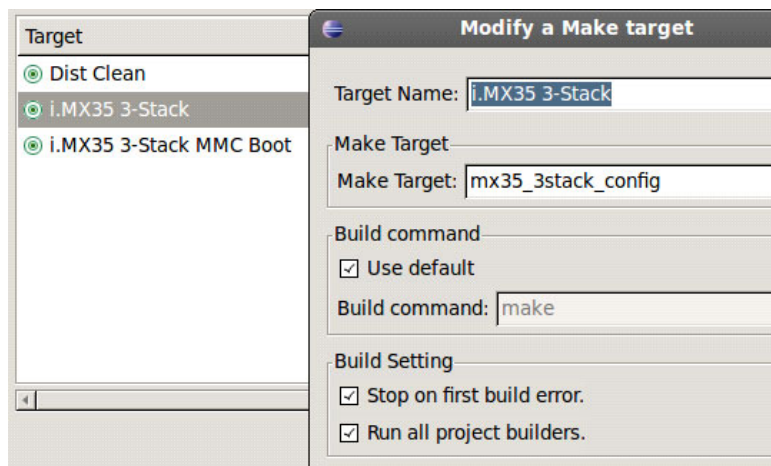


Figure 5. Make Targets in Eclipse Project

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 follows:

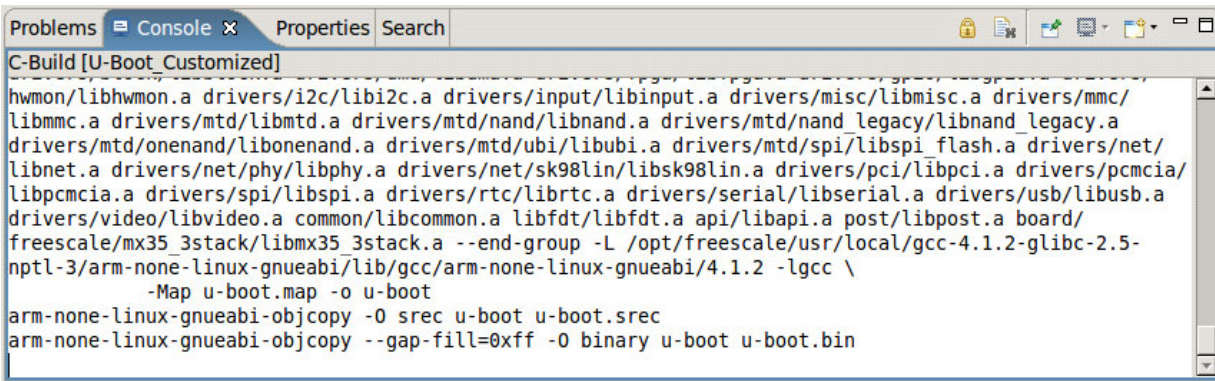
```
Make all
System not configured - see README
Make: *** [all] Error 1
```

Additionally, the Dist Clean target is used to perform a full clean up of the source tree. Therefore, the Dist Clean target removes all the files of the previous build.

After the successful configuration of Eclipse IDE, the steps to build the project are as follows:

1. Build the Dist Clean make target (optional).
2. Configure the system using the desired make target from the Make Target column in [Table 7](#).
3. Build the project.

The output files are placed in the U-Boot source code path as shown in [Figure 6](#).



```

C-Build [U-Boot_Customized]
-----
hwmon/libhwmon.a drivers/i2c/libi2c.a drivers/input/libinput.a drivers/misc/libmisc.a drivers/mmc/
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/mx35_3stack/libmx35_3stack.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

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

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 which 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, the Freescale logo, CodeWarrior, ColdFire, PowerQUICC, StarCore, and Symphony are trademarks of Freescale Semiconductor, Inc. Reg. U.S. Pat. & Tm. Off. CoreNet, QorIQ, QUICC Engine, and VortiQa are trademarks of Freescale Semiconductor, Inc. All other product or service names are the property of their respective owners. ARM is the registered trademark of ARM Limited. ARM1136JF-S is the trademark of ARM Limited. © 2010 Freescale Semiconductor, Inc.

