

Booting from On-Chip ROM (eSDHC or eSPI)

by *Networking and Multimedia Group*
Freescale Semiconductor, Inc.
Austin, TX

1 Overview

Several Freescale devices, such as MPC8536E, P2020, and P1020, offer booting from an on-chip ROM in addition to many other booting options. The option of on-chip ROM booting consists of booting from an SD/MMC card or from a serial flash or EEPROM memory device with an SPI interface. The term ‘EEPROM’ is used in this document when referring to the memory device.

This document describes how to boot from an SD/MMC card or an EEPROM under Linux Operating System. [Section 2, “Building the Configuration File,”](#) describes how to build the configuration file required to boot from the on-chip ROM. [Section 3, “Building a RAM-Based U-Boot,”](#) describes how to build a RAM-based U-Boot that is used for booting. [Section 4, “Bootting Utility Application,”](#) covers what the boot_format utility tool is and how to put the configuration file and booting image on an SD/MMC card, as well as how to generate an eSPI image used for booting. [Section 5, “POR Configurations for Booting from an On-Chip ROM,”](#) describes the POR configurations for the on-chip ROM booting. [Section 6, “Bootting from On-Chip ROM on an MPC8536DS Board,”](#) uses the MPC8536DS board as an example for explainaining the booting process.

Contents

1. Overview	1
2. Building the Configuration File	2
3. Building a RAM-Based U-Boot	12
4. Bootting Utility Application	14
5. POR Configurations for Booting from an On-Chip ROM	16
6. Bootting from On-Chip ROM on an MPC8536DS Board	17
7. Revision History	21

To support booting from all cards as well as from different types of cards, only 1-bit mode is used for booting from an SD/MMC card.

2 Building the Configuration File

The on-chip ROM contains a basic eSDHC device driver and a simple eSPI driver. The driver code in the ROM performs block copies from either an SD/MMC card or an EEPROM memory with an SPI interface to any target memory connected to the silicon, such as DDR. The target memory must be configured before it can be used and the booting image is able to be copied to it. Therefore, a special data structure is defined to accomplish the configurations and other information related to the booting image. A configuration file must be created to cover everything in the data structure except the user code (image).

As mentioned in the device reference manuals, a two-stage boot process may be used to boot from the on-chip ROM in case faster booting is required. Currently, one-step booting from a fast SD/MMC card takes approximately 30 ms to copy the 512-Kbyte U-Boot image to a memory location.

CAUTION

Using this mechanism could potentially overwrite the content of the CCSRBAR. In this case, it would cause the boot process to hang.

2.1 Building the Configuration File for SD/MMC Booting

The SD/MMC card should contain a specific data structure with control words, device configuration information, and boot image, such as U-Boot. [Figure 1](#) and [Table 1](#) describe the data structure used for the SD boot. See the “eSDHC Boot,” section in the device reference manuals for more information. The data format of a configuration file is “offset: data”-based. The first data value is the offset address, and the second data is the actual data value. The “:” must be used between the offset address and the data value at every line. The value for the offset is a hex-based number and the configuration data must be put on the first 24 blocks of the SD/MMC card. The data is a 32-bit data in hexadecimal format. [Example 1](#) shows an example of a configuration file.

[Figure 1](#) shows the data structure.

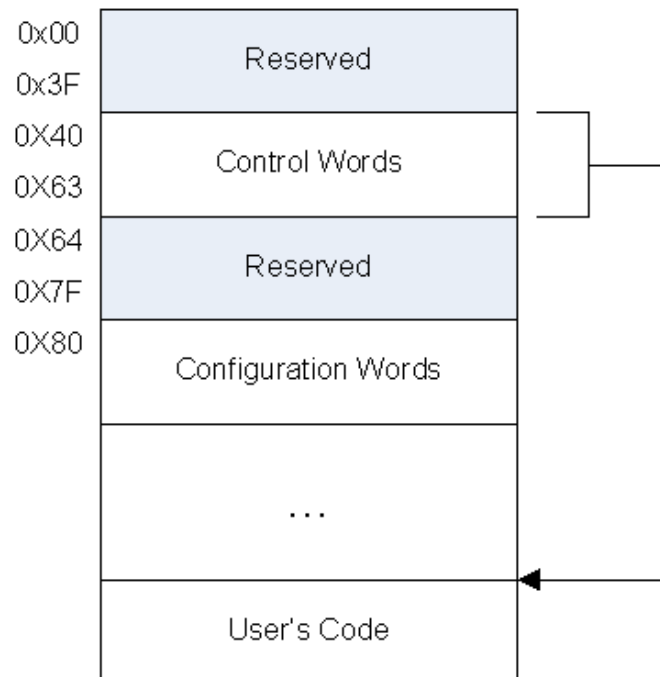


Figure 1. SD/MMC Card Data Structure

Table 1 describes the contents of the SD/MMC card for each section, as follows:

- Control Words
- Configuration Words
- User Code

The length of Control Words is fixed as shown in Figure 1. For the length of Configuration Words, the maximum is 40 pairs due to the FAT16/FAT32 file system support. The length of User Code section is limited by the length of the 32-bit address or the size of the SD/MMC card memory. Normally, the length of the user code is the size of the U-Boot, which is 512 Kbytes.

Table 1. SD/MMC Data Structure Definition

Address	Data Bits [0:31]
0x00–0x3F	Reserved
0x40–0x43	BOOT signature. This location should contain the value 0x424f_4f54, which is the ascii code for BOOT. The boot loader code searches for this signature. If the value in this location does not match the BOOT signature, the SD/MMC card does not contain a valid user code. The boot loader code disables the eSDHC and issues a hardware reset request of the SoC by setting RSTCR[HRESET_REQ].
0x44–0x47	Reserved
0x48–0x4B	User's code length. Number of bytes in the user's code to be copied. This must be a multiple of the SD/MMC card's block size (and the user's code zero-padded if necessary to achieve that length). User's code length <= 2 GBytes.
0x4C–0x4F	Reserved
0x50–0x53	Source Address. Contains the starting address of the user's code as an offset from the SD/MMC card starting address. In Standard Capacity SD/MMC Cards, the 32-bit Source Address specifies the memory address in byte address format. This must be a multiple of the SD/MMC card's block size. In High Capacity SD Cards (>2 GBytes), the 32-bit Source Address specifies the memory address in block address format. Block length is fixed to 512 bytes as per the SD High Capacity specification
0x54–0x57	Reserved
0x58–0x5B	Target Address. Contains the target address in the system's local memory address space in which the user's code is copied to. This is a 32-bit effective address. The core is configured in such a way that the 36-bit real address is equal to this (with 4 most significant bits zero).
0x5C–0x5F	Reserved
0x60–0x63	Execution Starting Address. Contains the jump address in the system's local memory address space into the user's code first instruction to be executed. This is a 32-bit effective address. The core is configured in such a way that the 36-bit real address is equal to this (with 4 most significant bits zero).
0x64–0x67	Reserved
0x68–0x6B	N. Number of Configuration Address/Data pairs. Must be $1 \leq N \leq 1024$ (but is recommended to be as small as possible).
0x6C–0x7F	Reserved
0x80–0x83	Configuration Address 1
0x84–0x87	Configuration Data 1
0x88–0x8B	Configuration Address 2

Table 1. SD/MMC Data Structure Definition (continued)

Address	Data Bits [0:31]
0x8C–0x8F	Configuration Data 2
	...
0x80 + 8*(N – 1)	Configuration Address N
0x80 + 8*(N– 1)+4	Configuration Data N
	...
	...
	...
—	User's Code

There are at least two sections in the configuration file; Control Words and Configuration Words are defined in this file. The first fixed six pairs of data in the file are Control Words. The Configuration Words can be varied depending on the system. However, it must be $1 \leq N \leq 1024$, but is recommended to be as small as possible because reset of the configurations can be accomplished by the U-Boot.

An example of a configuration file is shown in [Example 1](#), and the detailed explanations of the file are as follows:

1. The value of 424f4f54 is the booting signature. It should be the first data and has to be an offset of 0x40 from the start address of the first 24 blocks, whose size is 512 bytes. The offset value should be a value of 0x40, 0x240, 0x440, ---, 0x2E40.
2. The value at offset 0x48 is the booting image code length in bytes, which is the length of RAM-based special U-Boot image (0x00080000). This means the U-Boot has at most 524288 bytes. This must be a multiple of the SD/MMC card's block size, which is 512 bytes. It should be zero-padded if necessary to achieve that length.
3. The value at offset 0x50 is the Source Address. It indicates the starting address of the special U-Boot code as an offset from the SD/MMC card starting address. In Standard Capacity SD/MMC Cards, the 32-bit Source Address specifies the memory address in byte address format. This must be a multiple of the SD/MMC card's block size. In High Capacity SD Cards (>2 GBytes), the 32-bit Source Address specifies the memory address in block address format. Block length is fixed to 512 bytes as per the SD High Capacity specification. The example below, for a standard card, has a value of 0x00001000. Its value should be 0x00000008 for a high capacity SD card.
4. The value of 0x11000000 below at offset 0x58 is the address in DDR or SRAM memory in which a booting image and the RAM-based U-Boot code is copied to. This is a 32-bit effective address. Keep this value unchanged in the configuration file if the default Freescale LTIB or BSP package is used. This value should match the U-Boot configuration, and is the first data/instruction location of the U-Boot.
5. The value at offset 0x60 is the Execution Starting Address, which means that this is the first instruction of the U-Boot to be executed. This is a 32-bit effective address. Normally, this value is the same as the value at offset 0x58. But it is not necessary if the first location of the U-Boot image is not the starting instruction.

6. The value at offset 0x68 is Number of Configuration Address/Data pairs in the following section. A value of 0x10 is in our following example.
7–10 below are based on a sample of DDR configuration. However, at least (normally only LAW 0) one LAW should be configured using the configuration.
7. From offset 0x80 to 0xE4 except offset of 0xd8 and 0xdc, these are the DDR configuration parameters.
8. The value on offset of 0xd8 indicates a delay. The value on offset of 0xdc indicates 0x100 = 256 of 8 CCB clocks delay.
9. The values of the offset 0xe0 to 0xf4 are the configuration parameters of LAW 0.
10. The value on the offset 0xf8 indicates the end of the configuration file.

Example 1. Configuration File for SD/MMC or eSPI Booting

```
40:424f4f54
44:00000000
48:00080000
4c:00000000
50:00001000 <= Standard card, 00000008 <= High capacity card
54:00000000
58:11000000
5c:00000000
60:11000000
64:00000000
68:00000010

80:ff702110
84:42000000
88:ff702000
8c:0000001f
90:ff702080
94:80010202
98:ff702104
9c:00260802
a0:ff702108
a4:3935d322
```

a8:ff70210c
ac:05105408
b0:ff702114
b4:24401000
b8:ff702118
bc:00400432
c0:ff702124
c4:06db03e8
c8:ff702128
cc:deadbeef
d0:ff702130
d4:03800000
d8:40000001
dc:00000100
e0:ff702110
e4:c3008000
e8:ff700C08
ec:00000000
f0:ff700C10
f4:80F0001D
f8:efefefef

2.2 Building the Configuration File for eSPI Booting

Booting from eSPI is supported by the on-chip ROM booting. The on-chip ROM contains the basic eSPI device driver and the code to perform a block copy from an EEPROM. After the device has completed the reset sequence, if the ROM location selects the on-chip ROM eSPI Boot configuration, the e500 core starts to execute code from the internal on-chip ROM. The e500 core configures the eSPI controller, and enables it to communicate with the external EEPROM. The EEPROM should contain a specific data structure with control words, device configuration information, and a booting image. The data structure for the configuration file that is used for the eSPI boot is described in “eSDHC Boot” section of the applicable device reference manual. The data structure of a Configuration file is offset: data based.

Figure 2 shows the data structure.

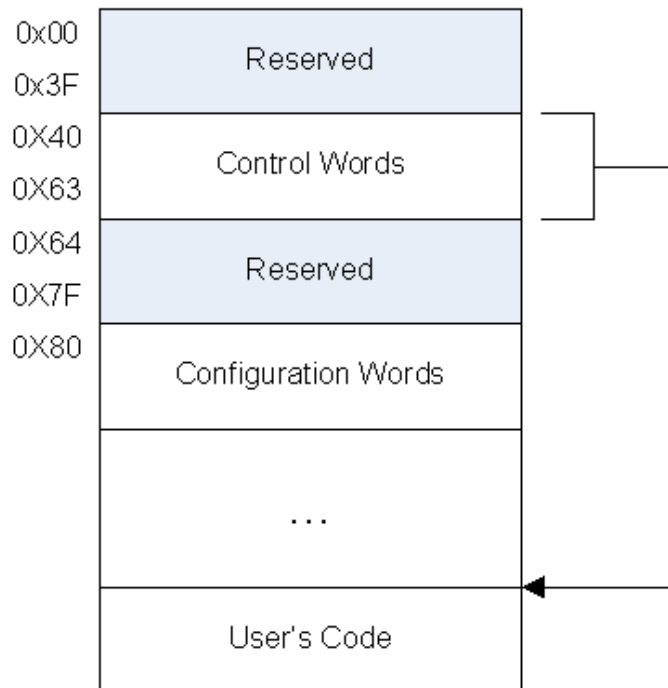


Figure 2. EEPROM Data Structure

Table 2 describes the contents of the EEPROM for each section, as follows:

- Control Words
- Configuration Words
- User Code.

The length of Control Words is fixed as shown in Figure 2. For the Configuration Words, its length is limited by the 16-bit or 24-bit address. The length for User Code sections is limited by the length of the 32-bit address or the size of EEPROM memory.

Table 2. eSPI EEPROM Data Structure Definition

Address	Data Bits [0:31]
0x00–0x3F	Reserved
0x40–0x43	BOOT signature. This location should contain the value 0x424f_4f54, which is the ascii code for BOOT. The eSPI loader code searches for this signature, initially in 24-bit addressable mode. If the value in this location doesn't match the BOOT signature, then the EEPROM is accessed again, but in 16-bit mode. If the value in this location still does not match the BOOT signature, it means that the eSPI device doesn't contain a valid user code. In such case the eSPI loader code disables the eSPI and issues a hardware reset request of the SoC by setting RSTCR[HRESET_REQ].
0x44–0x47	Reserved
0x48–0x4B	User's code length. Number of bytes in the user's code to be copied. Must be a multiple of 4. $4 \leq \text{User's code length} \leq 2\text{GBytes}$.
0x4C–0x4F	Reserved
0x50–0x53	Source Address. Contains the starting address of the user's code as an offset from the EEPROM starting address. In 24-bit addressing mode, the 8 most significant bits of this should be written to as zero, because the EEPROM is accessed with a 3-byte (24-bit) address. In 16-bit addressing mode, the 16 most significant bits of this should be written to as zero.
0x54–0x57	Reserved
0x58–0x5B	Target Address. Contains the target address in the system's local memory address space in which the user's code is copied to. This is a 32-bit effective address. The core is configured in such a way that the 36-bit real address is equal to this (with 4 most significant bits zero).
0x5C–0x5F	Reserved
0x60–0x63	Execution Starting Address. Contains the jump address in the system's local memory address space into the user's code first instruction to be executed. This is a 32-bit effective address. The core is configured in such a way that the 36-bit real address is equal to this (with 4 most significant bits zero).
0x64–0x67	Reserved
0x68–0x6B	N. Number of Config Address/Data pairs. Must be ≤ 1024 (but is recommended to be as small as possible).
0x6C–0x7F	Reserved
0x80–0x83	Configuration Address 1
0x84–0x87	Configuration Data 1
0x88–0x8B	Configuration Address 2
0x8C–0x8F	Configuration Data 2

Table 2. eSPI EEPROM Data Structure Definition (continued)

Address	Data Bits [0:31]
	...
0x80 + 8*(N-1)	Configuration Address N
0x80 + 8*(N-1)+4	Config Data N (Final Config Data N optional)
	...
	...
	...
	User's Code

The configuration words section is comprised of Configuration Address and Configuration Data pairs of adjacent 32-bit fields. These are typically used to configure the local access windows and the target memory controller's registers. They are system-dependent, because they need to be aware of the type and configuration of memory in a particular system.

The Configuration Address field has two modes that are selected by the least significant bit in the field (CNT). If the CNT bit is clear, then the 30 most significant bits are used to form the address pointer and the Configuration Data contains the data to be written to this address. If the CNT bit is set then the 30 most significant bits are used for control instruction. This flexible structure allows the user to configure any 4-byte-aligned memory mapped register, perform control instructions, and specify the end of the configuration stage. Refer to section "eSPI Boot" in the applicable device reference manual for more information.

The value of 424f4f54 is the booting signature. It should be the first data and has to be an offset of 0x40 from the start address. Unlike eSDHC, it must be at 0x40 offset.

The Configuration file example used for a standard SD/MMC card in [Example 1](#) also applies to eSPI. In [Example 2](#), the delay and the SPI interface frequency changes are shown by using the configuration words.

Example 2. Configuration File for eSPI Booting

```
40:424f4f54
44:00000000
48:00080000
4c:00000000
50:00000100
54:00000000
58:11000000
5c:00000000
60:11000000
```

64:00000000
68:00000012

80:ff702110
84:42000000
88:ff702000
8c:0000001f
90:ff702080
94:80010202
98:ff702104
9c:00260802
a0:ff702108
a4:3935d322
a8:ff70210c
ac:05105408
b0:ff702114
b4:24401000
b8:ff702118
bc:00400432
c0:ff702124
c4:06db03e8
c8:ff702128
cc:deadbeef
d0:ff702130
d4:03800000
d8:40000001
dc:00000100
e0:ff702110
e4:c3008000
e8:ff700C08
ec:00000000
f0:ff700C10

f4:80F0001D

f8:20000001 <= *Change the SPI interface frequency*

fc:21172210

100:40000001 <= *Delay*

104:00000001

108:efefefef

3 Building a RAM-Based U-Boot

The process used to build a RAM-based U-Boot can apply to both eSPI and eSDHC. The same RAM-based U-Boot image can be used for booting from either an SD/MMC card or an EEPROM.

On-chip boot ROM code uses the information from an SD/MMC card or content in the EEPROM to configure a LAW(s) as well as a target memory device, such as DDR, and to copy the boot image such as a U-Boot image to a target memory device through the eSDHC or the SPI interface. After all the image code is copied, the e500 core jumps to the address specified at the offset 0x60 in the configuration information, and starts to execute the code from the target memory device.

The on-chip boot ROM code has configured TLB 1 to cover from 0 to 2 Gbytes. It is very flexible to copy the image to any specified location of a target memory device based on the user's configuration. It may conflict with the U-Boot configuration in use, so the U-Boot code should consider this TBL1 at the start up.

The RAM-based U-Boot is different from a U-Boot based on a NOR flash. After a processor completes the reset (POR) sequence, if the ROM location selects the on-chip ROM boot configuration, the e500 core starts to execute code from the internal on-chip ROM, which is internally mapped to 0xFFFFE000. Therefore, a different configuration file is needed for the RAM-based U-Boot. This U-Boot should start from a value specified at the offset of 0x58 in the control words section. Normally, the U-Boot uses a Compile Time Header File to assign the starting location. For example, in MPC8536E BSP, *u-boot-sdboot.lds* file is used to make sure that the U-Boot starts from the location at 0x11000000. It only works with a value of 0x11000000 at the offset 0x58 in a U-Boot EEPROM or in an SD/MMC card.

The initialization code for the U-Boot should be changed to fit the different U-Boot options. Besides the TLB 1 configuration already in the Boot ROM code, the RAM-based U-Boot also should expect that at least one LAW configuration as well as the target memory configurations have already been in an SD/MMC card or in an EEPROM. However, they can be changed if necessary.

Normally, the U-Boot environment variables should be saved to an SD/MMC card or an EEPROM. The corresponding code dealing with the environment should be changed to do so. For Freescale BSP, *cmd_nvedit.c* and *env_common.c* have been changed and *env_sdcard.c* is added to deal with the environment variables.

In [Example 3](#), the MPC8536E BSP is used as an example to show how to build a RAM-based U-Boot.

Example 3. Building a RAM-Based U-Boot

It is assumed that the MPC8536E BSP package is already installed on a Linux system. The U-Boot system has been compiled, configured, and built as well as the Linux system for a PowerQUICC™ processor. The source code has been saved to a directory. The development tools have been installed and configured properly.

In the root directory, the following lines must be added to the Makefile:

```
MPC8536DS_SDCARD_config: unconfig
```

```
@echo "#define CFG_SDCARD_U_BOOT" >> $(obj)include/config.h ;
@$(MKCONFIG) -a MPC8536DS ppc mpc85xx mpc8536ds freescale ; \
    echo "TEXT_BASE = 0x11001000" > $(obj)board/freescale/mpc8536ds/config.tmp ; \
    echo "#define CFG_SDCARD_U_BOOT" >> $(obj)include/config.h ; \
    echo "CFG_SDCARD_U_BOOT = y" >> $(obj)include/config.mk ;
```

In the directory of $$(OBJTREE)/board/$(BOARDDIR)/$, *u-boot-sdboot.lds* file should be added. This file defines the boot page at 0x11000000. See the file in the BSP for details. In the same directory, the *config.mk* should include the following line at the top:

```
sinclude $(OBJTREE)/board/$(BOARDDIR)/config.tmp
```

An alias of the following is used to make the build process easier:

```
alias 85xxmake='make CROSS_COMPILE=powerpc-linux-gnuspe- ARCH=ppc'
```

The only difference between building a regular U-Boot image and a RAM-based image is a command for the configuration. To build a special RAM-based U-Boot, type the following command:

```
85xxmake distclean
85xxmake MPC8536DS_SDCARD_config
85xxmake
```

The U-Boot image will be in the current directory. This image can be used for booting from both an SD/MMC card or from an EEPROM.

4 Booting Utility Application

The booting utility application *boot_format* puts the configuration file and the RAM-based U-Boot image on an SD/MMC card. This application runs under a regular Linux machine and requires a super user mode to run. It can also generate a binary image that can be used for booting from the SPI interface by selecting a different option.

4.1 Booting Utility

The name of the Booting Utility Application is *boot_format*. It shows the user how to use it when typing *./boot_format*:

```
[root@b08938-02 new_tool]# ./boot_format
```

```
Usage: ./boot_format config_file image -sd dev|-spi out_image [-o out_config]
```

Where:

config_file: Boot signature, control words, configuration words, or user code.

image: optional user code.

dev: sd device file in SD mode

out_image: out image in SPI mode(exclusive with SD mode)

out_config: optional modified configuration file in SD mode(Only for FAT) format of the arguments:

config_file: data in hex. (Its length must be < 446 Bytes)

Address offset: data (The first two line of data must be:)

```
40: 424f4f54
```

```
44: 00000000
```

image (if any) has to be a binary file such as the uBoot image file.

4.1.1 Putting the Image on an SD/MMC Card Using Booting Utility

When using the booting utility to put all the configuration information as well as the U-Boot image on an SD/MMC card, it can be launched at the command line:

```
./boot_format config_file image -sd dev [-o out_config]
```

Where:

- *config_file*, contains boot signature, control words, configuration words, or user code
- *image*: optional U-Boot image
- *dev*: sd device file in SD mode, such as */dev/sdc*
- *out_config*: optional modified configuration file

NOTE

This utility may change the source address value at offset 0x50 because the FAT file system must remain untouched.

4.1.2 Generating an Image for EEPROM Using the Booting Utility

When using the booting utility to generate an image that can be used for EEPROM, from the configuration file and the U-Boot image, it can be launched at the command line:

```
./boot_format config_file image -spi out_image
```

Where:

- *config_file*, contains Boot signature, control words, configuration words, or user code.
- *image*: optional U-Boot image.
- *Out_image*: the U-Boot image that will be put into EEPROM for booting

4.2 SD/MMC Card File Format

An SD/MMC card is normally formatted to either the FAT16 or FAT32 file system format when it is purchased from a store. This boot utility application can keep this file system untouched. Based on the SD standard, it must be formatted to be a FAT 32 system if the capacity of a card is larger than 2 Gbytes. Otherwise, it should be formatted to a FAT16 system. The card can be reformatted by using the File Explore in Windows, if necessary. Currently, the FAT file system is the only format supported.

4.3 Putting a Boot Image on an SD/MMC Card

The booting utility runs on a regular Linux machine. It requires that the Linux machine has either an SD/MMC card interface, or a USB interface with an SD/MMC to USB converter. The steps for putting the built U-Boot image on an SD/MMC card are as follows:

1. Insert an SD/MMC card to a Linux machine.
2. Check whether it is in */dev/sdx*. *x* should be a char of *a, b, c,...*
3. Copy the application *boot_format* to a directory on the Linux machine.
4. Copy the SD booting configuration file and the U-Boot image to the same directory in step 3.
5. Switch to a super user mode using *su* if not logged in as a super user.
6. Type *./boot_format config_file image -sd /dev/sdx*, where *x* is the same as in step 2.

4.4 Putting a Booting Image on an EEPROM

As mentioned in [Section 4.3, “Putting a Boot Image on an SD/MMC Card,”](#) the booting utility runs on a regular Linux machine. It does not require that the Linux machine has either an SD/MMC card interface or a USB interface for the SPI interface boot. The steps for combining the configuration file and the U-Boot to a booting image are as follows:

1. Copy the application *boot_format* to a directory on the Linux machine.
2. Copy the eSPI booting configuration file and the U-Boot image to the same directory in step 1.
3. Switch to a super user mode using *su* if not logged in as a super user.
4. Type *./boot_format config_file image -spi out_image*.

The *out_image* is the booting image that can be put on an EEPROM. The ways to put the generated booting image on an EEPROM are as follows:

- Use the EEPROM writer, which is supplied from EEPROM manufacture or a tool supplier.
- Write the booting image to an EEPROM under the Linux environment after booting from another method first.

Section 6.2, “eSPI Booting from an 8536 MDS Board,” gives an example of how to enable the eSPI Linux driver. The property for the node of the eSPI EEPROM needs to be changed so that it can be writeable. The steps to put a booting image on an EEPROM are as follows:

1. Configure the Linux image with eSPI driver enabled.
2. Build device tree including a node for EEPROM device (MTD device).
3. Boot to Linux prompt (through other ways).
4. Check mtd device to see the EEPROM device.
5. Mount the EEPROM device.
6. Copy the booting image to a directory.
7. Erase the beginning part of the EEPROM.
8. Copy the booting image to the EEPROM.

5 POR Configurations for Booting from an On-Chip ROM

The on-chip ROM code does not set up any local access windows (LAWs). Access to CCSR address space does not require a LAW. It is the user’s responsibility to set up a local access window through a Control Word address/data pair for the desired Target Address and Execution Starting Address (which is typically in either DDR or Local Bus memory space). As shown in Example 1 or Example 2, at least one LAW must be configured to make booting work.

Note that any such local access window configured at this time must have the 4 Mbits of the address.

5.1 Configurations for SD Booting

There are several POR configuration settings needed to boot from an SD/MMC card, as follows:

1. Boot_Rom_Loc; (cog_rom_loc[0:3]). A value of 0b0111 starts the booting from an SD/MMC.
2. Only one core can be in booting mode while other cores must be in a boot holdoff mode if a silicon has multiple cores. The CPU boot configuration input, *cfg_cpux_boot*, should be 0, where x is from 1 to n, which is the number of cores in this silicon.

NOTE

The polarity of the SDHC_CD signal should be active low. Although it does not affect the booting from eSDHC, it may be useful information.

5.2 Configurations for Booting from EEPROM

There are several configuration settings needed to boot from an SD/MMC card, as follows:

1. `Boot_Rom_Loc`; (`cog_rom_loc[0:3]`). A value of `0b0110` starts the booting from an EEPROM with an SPI interface.
2. Only one core can be in booting mode while other cores have to be in a boot holdoff mode if a silicon has multiple cores. The CPU boot configuration input, `cfg_cpux_boot`, should be 0, where `x` is from 1 to `n`, which is the number of cores in this core.

During booting, the eSPI controller is configured to operate in master mode. The eSPI chip select 0 (`SPI_CS[0]`) must be connected to the EEPROM CS and selectively enables the EEPROM. No other chip select can be used for booting.

6 Booting from On-Chip ROM on an MPC8536DS Board

This section contains the following two sections:

- [Section 6.1, “SD Booting from an MPC8536 MDS Board”](#)
- [Section 6.2, “eSPI Booting from an 8536 MDS Board”](#)

6.1 SD Booting from an MPC8536 MDS Board

The steps for booting from an SD card on an 8536 MDS board are as follows:

1. Plug in the SD card in slot 0 (external slot) of the SD slot.
2. Change bit 5678 of SW2 to `0xB0111`.
3. Change bit 1 of SW3 to 0.
4. Change the bit 7 of SW8 to 1.
5. Keep the reset of the SW setting default.
6. Turn on the power; the U-Boot should come up.

6.2 eSPI Booting from an 8536 MDS Board

This section contains the following two sections:

- [Section 6.2.1, “Putting a Booting Image on an EEPROM on an 8536 MDS Board”](#)
- [Section 6.2.2, “eSPI Booting from EEPROM on an 8536 MDS Board”](#)

6.2.1 Putting a Booting Image on an EEPROM on an 8536 MDS Board

The MPC8536 DS supports EEPROM over the SPI interface. The steps to put a booting image on an EEPROM on the board are as follows:

1. Configure the Linux kernel to turn on the eSPI driver:

- SPI support
 - `CONFIG_SPI` Y
 - `CONFIG_SPI_BITBANG` Y

- CONFIG_FSL_ESPI Y
- Memory Technology Device (MTD) support:
 - CONFIG_MTD Y
 - CONFIG_MTD_PARTITIONS Y
 - CONFIG_MTD_CHAR Y
 - CONFIG_MTD_BLOCK Y
- Self-contained MTD device drivers:
 - CONFIG_MTD_FSL_M25P80 Y
- CONFIG_M25PXX_USE_FAST_READ Y

2. Device Tree binding is shown in [Table 3](#).

Table 3. Device Tree Binding

Property	Type	Status	Description
compatible	string	Required	Should be "fsl,mpc8536-espi"
mode (note: spi node)	string	Required	Should be "cpu"
mode (note: fsl_m25p80 node)	integer	Required	Should be "0"
modal	string	Required	Should be "s25s1128b"
clock-frequency	integer	Required	Should not beyond <80000000>

Default node:

```
spi@7000 {
    cell-index = <0>;
    #address-cells = <1>;
    #size-cells = <0>;
    compatible = "fsl,mpc8536-espi", "fsl,espi";
    reg = <0x7000 0x1000>;
    interrupts = <59 0x2>;
    interrupt-parent = <&mpic>;
    mode = "cpu";

    fsl_m25p80@0 {
        cell-index = <0>;
        compatible = "fsl,espi-flash";
        reg = <0x7000>;
        modal = "s25s1128b";
        clock-frequency = <50000000>
        mode = <0>;
        flash@0 {
```

```

compatible = "fsl,mpc8536-eeeprom";
#address-cells = <1>;
#size-cells = <1>;
u-boot@0 {
    label = "u-boot";
    reg = <0x00000000 0x00100000>;
};
kernel@100000 {
    label = "kernel";
    reg = <0x00100000 0x00500000>;
};
fs@600000 {
    label = "fs";
    reg = <0x00600000 0x00a00000>;
};
};
fsl_m25p80@1 {
    cell-index = <1>;
    compatible = "fsl,espi-flash";
    reg = <0x7000>;
    modal = "s25sl128b";
    clock-frequency = <50000000>;
    mode = <0>;
};
fsl_m25p80@2 {
    cell-index = <2>;
    compatible = "fsl,espi-flash";
    reg = <0x7000>;
    modal = "s25sl128b";
    clock-frequency = <50000000>;
    mode = <0>;
};
fsl_m25p80@3 {
    cell-index = <3>;
    compatible = "fsl,espi-flash";
    reg = <0x7000>;
    modal = "s25sl128b";
    clock-frequency = <50000000>;
};

```

```

        mode = <0>;
    };
};

```

The partition information for mtd0 needs to remove the “read-only” property in the mpc8536ds.dts

3. Boot the MPC8536DS system until login.
4. Check mtd device:

```

/root # cat /proc/mtd
dev: size erasesize name
mtd0: 00100000 00010000 "u-boot"
mtd1: 00500000 00010000 "kernel"
mtd2: 00a00000 00010000 "fs"
mtd3: 01000000 00010000 "SPIFLASH1"
mtd4: 01000000 00010000 "SPIFLASH2"
mtd5: 01000000 00010000 "SPIFLASH3"
/root #

```

5. Put the boot image to a directory:

- Set up the Ethernet port and gateway:

```

/boot # ifconfig eth0 down
/boot # ifconfig eth0 xx.xxx.xxx.xxx

```

- Start the TFTP server on a PC, and put the image file to the TFTP root directory.
- Start the TFTP client in the MPC8536DS system:

```

/jzhao # tftp yyy.yyy.yyy.yyy
/tftp> get outimage
/tftp> quit

```

6. Mount and write boot image to EEPROM:

```

/root # mkfs.ext2 /dev/mtdblock0
/root # mount /dev/mtdblock1 /mnt/tmp/
/root # cd /mnt/tmp/

/root # flash_eraseall /dev/mtd0
/root # cat outimage > /dev/mtd0

```

The Linux command **dd** can be used to check whether the image has been written to the EEPROM: **dd if=/dev/mtd0 of=a1 bs=256 count=2; od -x a1**

6.2.2 eSPI Booting from EEPROM on an 8536 MDS Board

The steps for booting from an SD card on an 8536 MDS board are as follows:

1. Change bit 5678 of SW2 to 0xB0110.
2. Keep the reset of the SW setting default.
3. Turn on the power; the U-Boot should come up.

7 Revision History

Table 4 provides a revision history for this application note.

Table 4. Document Revision History

Rev. Number	Date	Substantive Change(s)
0	12/2008	Initial release.

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
P.O. Box 5405
Denver, Colorado 80217
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 and the Freescale logo are trademarks or registered trademarks of Freescale Semiconductor, Inc. in the U.S. and other countries. All other product or service names are the property of their respective owners. The Power Architecture and Power.org word marks and the Power and Power.org logos and related marks are trademarks and service marks licensed by Power.org.

© Freescale Semiconductor, Inc., 2008. All rights reserved.

Document Number: AN3659

Rev. 0
12/2008

