Freescale Semiconductor
Application Note

Document Number: AN3659
Rev. 2, 06/2012

Booting from On-Chip ROM (eSDHC or eSPI)

This document describes on-chip ROM booting from an SD card/MMC or from an EEPROM under a Linux operating system on the following devices:

- MPC8536E
- MPC8569E
- P2020
- P1011
- P1012
- P1013
- P1020
- P1021
- P1022

NOTE

The term ‘EEPROM’ refers to a serial flash or an EEPROM memory device with an SPI interface in this document.

Contents

1 What does the on-chip ROM do?

The on-chip ROM includes both an eSDHC device driver and an eSPI driver. The driver code copies data from either an SD card/MMC or from an EEPROM with an SPI interface to a temporary memory location (see Section 2.4, “Choosing the temporary memory location”).

The on-chip ROM is internally mapped to 0xFFFF_E000 when booting from either an SD card/MMC or from an EEPROM.The on-chip boot ROM code uses the information from the SD card/MMC or the EEPROM to configure a temporary memory, such as the L2 cache or a DDR, before it copies a U-Boot image to this temporary memory. After SD card/MMC- or EEPROM-specific configurations are set up and all the image code is copied, the e500 core jumps to the address specified at offset 0x60 in the configurations and starts to execute the code from the temporary memory.

1.1 Avoiding on-chip ROM configuration issues using TLB1

The on-chip ROM code configures the first entry of the table lookaside buffer 1 (TLB1) to access up to 4 Gbytes starting from address 0x0000. Although the user configuration easily copies the image to any specified temporary memory location, it may conflict with the U-Boot configuration. This table shows how the MAS0–3 registers are set.

Table 1. TLB1 MAS0–3 register values

Register	Value
MAS0	0x1000_0000
MAS1	0xC000_0B00
MAS2	0x0000_000E
MAS3	0x0000_0015

Note: MAS4–7 and TLB1CFG are at their reset values.

2 Building a Cconfiguration file

2.1 Boot location-specific data structures

A special data structure specific to each booting location provides the configurations and other information related to the booting image (see Section 2.5.1, “SD Card/MMC data structure,” and Section 2.6.1, “EEPROM data structure”). A configuration file must be created to implement everything in the data structure except the user code (which is most often a U-Boot image under Linux).

2.2 Requirements for configuration files

CAUTION

Improperly using configuration files may overwrite the content of the configuration, control, and status base address register (CCSRBAR), causing the boot process to hang.

The data format of a configuration file is offset/address: data-based, which means that the first data value is the offset/address, and the second data is the actual data value. Note the following requirements for creating configuration files:

- The colon must be used between the offset address and the data value at every line.
- The value for the offset is a hex-based number.
- The 32-bit data must be in hexadecimal format.
- The configuration data must be put on the first 24 blocks of an SD card/MMC.
- The configuration data must be put on the first block of an EEPROM.

NOTE

For the eSDHC interface, the address may be an offset.

2.3 Definition of an address/data pair

An address/data pair consists of a configuration offset/address and configuration data. Table 3, “SD Card/MMC data structure definition and address/data pairs,” and Table 5, “eSPI EEPROM data structure definition,” group the data into pairs using shading.

The configuration words section consists of address/data pairs of adjacent 32-bit fields. These address/data pairs are typically used to configure the local access windows (LAWs) and the temporary memory’s configuration registers.

NOTE

For a DDR memory, these register values may be system-dependent, because a different DDR memory requires a different set of configuration parameters in a particular system.

2.4 Choosing the temporary memory location

Use either DDR or L2 cache as the temporary memory location. Using L2 cache is ideal because it is more reliable and easier to configure and to debug than DDR.

However, if the size of the U-Boot exceeds the size of the L2 cache, DDR must be used. Use this table to determine whether DDR or L2 cache should be used as the temporary memory for your device.

Table 2. Temporary memory location

Device’s L2 Cache Size	U-Boot Size	Temporary Memory	Devices
256 Kbytes	512 Kbytes	DDR	P1xxx
512 Kbytes	512 Kbytes	L2 cache	MPC8536E, MPC8569E, P2020

2.5 Building an SD Card/MMC configuration file

NOTE

Keep in mind that only 1-bit mode is used for booting from an SD card/MMC (to support booting from different types of cards).

2.5.1 SD Card/MMC data structure

An SD card/MMC used for booting contains a specific data structure that consists of control words, device configuration information, and a boot loader, such as a U-Boot image. This figure shows the SD card/MMC boot data structure's sections.

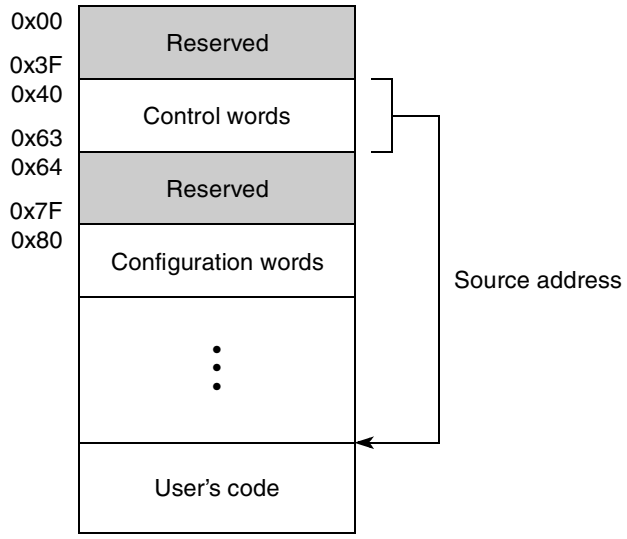

Figure 1. SD Card/MMC data structure

Figure 1 NOTES:

1. The length of the control words is fixed.
2. The maximum length of configuration words is 40 pairs due to the FAT16/FAT32 file system support if the data is copied to the first block of an SD card/MMC.
3. The length of the user code is limited by the length of the 32-bit address or the size of the SD card/MMC memory. Normally, the length of the user code is the size of the U-Boot (512 Kbytes).

This table describes the SD card/MMC data structure. Note that address/data pairs are delineated by shading.

Table 3. SD Card/MMC data structure definition and address/data pairs

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 card/MMC 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].
—	Reserved
0x48–0x4B	User's code length <= 2 Gbytes Number of bytes in the user's code to be copied, which must be a multiple of the SD card/MMC's block size (and the user's code zero-padded if necessary to achieve that length).
0x4C–0x4F	Reserved

Table 3. SD Card/MMC data structure definition and address/data pairs (continued)

Address	Data bits [0:31]
0x50–0x53	Source address Contains the starting address of the user's code as an offset from the SD card/MMC starting address. • In standard capacity (SD) cards/MMCs, the 32-bit source address specifies the memory address in byte address format, which must be a multiple of the SD card/MMC's block size. • In high capacity SD (SDHC) cards (>2 Gbytes), the 32-bit source address specifies the memory address in byte address format. However, it must be a multiple of block length, which is fixed to 512 bytes as per the SDHC 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.[1]
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.[1]
0x64–0x67	Reserved
0x68–0x6B	N Number of configuration data pairs Must be 1<=N<=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
…	
0x80 + 8*(N – 1)	Configuration address N
0x80 + 8*(N– 1)+4	Configuration Data N
… … …	
—	User code

Note:

[1] This is a 32-bit effective address. The e500 core is configured in such a way that the 36-bit real address is equal to this (with the 4 msbs = 0).

See Section "eSDHC Boot," in the applicable chip reference manual for more information.

2.5.2 Building an SD card/MMC configuration file

Control words and configuration words must be included in the SD card/MMC configuration file, but other sections of the data structure can also be included.

The initial six address/data pairs in the configuration file are control words of a fixed length. The number of configuration words can be varied depending on the system. However, it must be 1<=N<=1024, and is recommended to be as small as possible, because rest of the configurations can be accomplished by a boot loader such as the U-Boot.

This example shows an annotated SD card/MMC configuration file using DDR as the temporary memory, and Example 2 shows the configuration file using L2 cache.

Example 1. Configuration file for SD card/MMC using DDR

```
40:424f4f54
44:00000000
48:00080000
4c:00000000
50:00001000
54:00000000
58:11000000
5c:00000000
60:1107f000
64:00000000
68:00000010   16 address/data pairs of configuration words
80:ff702110   DDR configuration parameters
84:42000000   DDR configuration parameters
88:ff702000   DDR configuration parameters
8c:0000001f   DDR configuration parameters
90:ff702080   DDR configuration parameters
94:80010202   DDR configuration parameters
98:ff702104   DDR configuration parameters
9c:00260802   DDR configuration parameters
a0:ff702108   DDR configuration parameters
a4:3935d322   DDR configuration parameters
a8:ff70210c   DDR configuration parameters
ac:05105408   DDR configuration parameters
```

b0:ff702114 DDR configuration parameters
b4:24401000 DDR configuration parameters
b8:ff702118 DDR configuration parameters
bc:00400432 DDR configuration parameters
c0:ff702124 DDR configuration parameters
c4:06db03e8 DDR configuration parameters
c8:ff702128 DDR configuration parameters
cc:deadbeef DDR configuration parameters
d0:ff702130 DDR configuration parameters
d4:03800000 DDR configuration parameters
d8:40000001 Delay
dc:00000100 0x100 = 256 of 8 CCB clocks delay
e0:ff702110 DDR configuration parameters
e4:c3008000 DDR configuration parameters
e8:ff700C08 Configuration parameters of LAW 0
ec:00000000 Configuration parameters of LAW 0
f0:ff700C10 Configuration parameters of LAW 0
f4:80F0001D Configuration parameters of LAW 0
f8:efefefef End of configuration words

This example shows an SD card/MMC configuration file using L2 cache.

Example 2. Configuration file for SD card/MMC using L2 cache

```
40:424f4f54
44:00000000
48:00080000
4c:00000000
50:00001000
54:00000000
58:f8f80000
5c:00000000
60:f8fff000
64:00000000
68:00000006     6 address/data pairs of configuration words
80:ff720100     L2/SRAM configuration parameters
84:f8f80000     L2/SRAM configuration parameters
88:ff720e44     L2/SRAM configuration parameters
8c:0000000c     L2/SRAM configuration parameters
90:ff720000     L2/SRAM configuration parameters
94:80010000     L2/SRAM configuration parameters
98:ff72e40c     eSDHC configuration parameters
9c:00000040     eSDHC configuration parameters
a0:40000001     Delay
a4:00000100     0x100 = 256 of 8 CCB clocks delay
a8:80000001End of configuration words
```

This table shows additional details on how to define the configurations shown in Example 1 and Example 2.

Table 4. SD card/MMC configuration file details

Value at offset(s):	Description	Comments/Requirements
0x40 0x240 0x440 ... 0x2E40	Booting signature	This value should be the first data and must be an offset of 0x40 from the start address of the first 24 blocks (each block being 512 bytes).
0x48	Booting image code length in bytes	The length of RAM-based U-Boot image. A value of 0x0008_0000 means that the U-Boot has at most 524288 bytes. This value must be a multiple of the SD card/MMC's block's size (512 bytes). It should be zero-padded, if necessary.
0x50	Source address	This value indicates the starting address of the special U-Boot code as an offset from the SD card/MMC starting address. For all SD card/MMCs, the 32-bit source address specifies the memory address in byte address format. This value must be a multiple of the SD card/MMC's block's size (512 bytes).
0x58	Address in DDR memory where a booting image and the RAM-based U-Boot code are copied to.	If using the default Freescale LTIB or BSP package, keep this value unchanged in the configuration file. This value should match the U-Boot configuration, and is the first data/instruction location of the U-Boot.
0x60	Execution starting address	This value is the first instruction of the U-Boot to be executed.
0x68	Number of configuration data pairs in the subsequent data structure section	—

Note:

2.6 Building an EEPROM configuration file

2.6.1 EEPROM data structure

The basic eSPI driver code on the on-chip ROM performs reads from an EEPROM. An EEPROM used for booting contains a specific data structure that consists of control words, device configuration information, and a boot image. This figure shows the EEPROM boot data structure's sections.

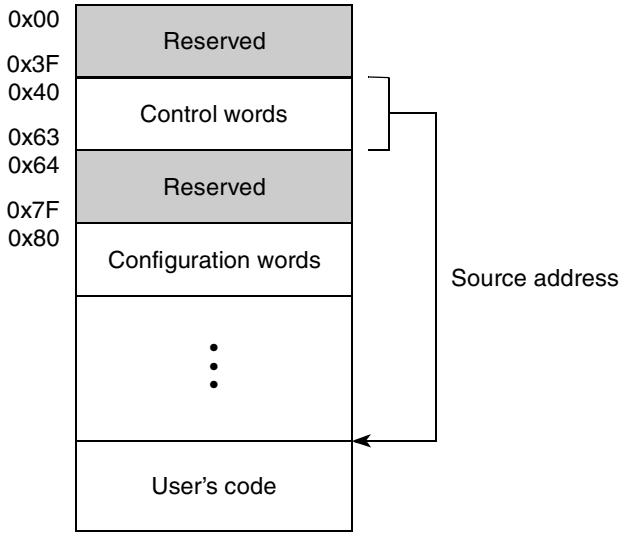

Figure 2. eSPI EEPROM data structure

Figure 2 NOTES:

1 The length of the control words is fixed.

2 The maximum length of configuration words is limited by the 16- or 24-bit address.

3 The length of the user code is limited by the length of the 32-bit address or the size of the EEPROM.

This table describes the EEPROM data structure's bits [0:31]. Note that address/data pairs are delineated by shading.

Table 5. 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

Table 5. eSPI EEPROM data structure definition (continued)

Address	Data Bits [0:31]
0x48–0x4B	User's code length Number of bytes in the user's code to be copied. Must be a multiple of 4. (4 <= 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 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 the 4 msbs = zero).
0x64–0x67	Reserved
0x68–0x6B	N Number of configuration 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
…	
0x80 + 8*(N–1)	Configuration address N
0x80 + 8*(N–1)+4	Config data N (final configuration data N optional)
… … …	
—	User code

See Section "eSPI Boot," in the applicable device reference manual for more information.

2.6.2 Building an EEPROM eSPI configuration file

The configuration address field has two modes that are selected by the lsb 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.

The value of 424f4f54 is the booting signature, which must be the first data at the offset 0x40 from the start address.

This example shows an EEPROM configuration file using DDR, and Example 4 shows the configuration file using L2 cache. Note that Example 1 is identical to Example 3 except for the differences shown in red.

Example 3. Configuration file for EEPROM using DDR

```
40:424f4f54
44:00000000
48:00080000
4c:00000000
50:00000400     <= Source address is 0x400 instead of 0x1000 for SD/MMC
54:00000000
58:11000000
5c:00000000
60:1107f000
64:00000000
68:00000012     <= Total of 18 pairs of configuration words
80:ff702110     DDR configuration parameters
84:42000000     DDR configuration parameters
88:ff702000     DDR configuration parameters
8c:0000001f     DDR configuration parameters
90:ff702080     DDR configuration parameters
94:80010202     DDR configuration parameters
98:ff702104     DDR configuration parameters
9c:00260802     DDR configuration parameters
a0:ff702108     DDR configuration parameters
a4:3935d322     DDR configuration parameters
a8:ff70210c     DDR configuration parameters
```

ac:05105408 *DDR configuration parameters*
b0:ff702114 *DDR configuration parameters*
b4:24401000 *DDR configuration parameters*
b8:ff702118 *DDR configuration parameters*
bc:00400432 *DDR configuration parameters*
c0:ff702124 *DDR configuration parameters*
c4:06db03e8 *DDR configuration parameters*
c8:ff702128 *DDR configuration parameters*
cc:deadbeef *DDR configuration parameters*
d0:ff702130 *DDR configuration parameters*
d4:03800000 *DDR configuration parameters*
d8:40000001 *Delay*
dc:00000100 *0x100 = 256 of 8 CCB clocks delay*
e0:ff702110 *DDR configuration parameters*
e4:c3008000 *DDR configuration parameters*
e8:ff700C08 *Configuration parameters of LAW 0*
ec:00000000 *Configuration parameters of LAW 0*
f0:ff700C10 *Configuration parameters of LAW 0*
f4:80F0001D *Configuration parameters of LAW 0*
f8:20000001 *<= Change the SPI interface frequency*
fc:21172210
100:40000001 *<= Delay*
104:00000001
108:efefefef

This example shows an EEPROM configuration file using L2 cache. Note that Example 2 is identical to Example 4 except for the differences shown in red.

Example 4. Configuration file for booting using the L2 cache

40:424f4f54
44:00000000
48:00080000
4c:00000000
50:00000400 *<= Source address is 0x400 instead of 0x1000 for SD/MMC*

54:00000000
58:f8f80000
5c:00000000
60:f8fff000
64:00000000
68:00000008 <= *Total of 8 pairs of configuration words*
80:ff720100 *L2/SRAM configuration parameters*
84:f8f80000 *L2/SRAM configuration parameters*
88:ff720e44 *L2/SRAM configuration parameters*
8c:0000000c *L2/SRAM configuration parameters*
90:ff720000 *L2/SRAM configuration parameters*
94:80010000 *L2/SRAM configuration parameters*
98:ff72e40c *eSDHC configuration parameters*
9c:00000040 *eSDHC configuration parameters*
a0:40000001 *Delay*
a4:00000100 *0x100 = 256 of 8 CCB clocks delay*
a8:20000001 <= *Change the SPI interface frequency*
ac:21172210
b0:40000001 <= *Delay*
b4:00000001
b8:80000001

This table shows additional details on how to define the configurations shown in Example 3 and Example 4.

Table 6. EEPROM configuration file details

Value at address:	Description	Comments/Requirements
0x40	Booting signature	This value should be the first data and must be an offset of 0x40 from the start address of an EEPROM.
0x48	Booting image code length in bytes	The length of RAM-based special U-Boot image. A value of (0x0008_0000) means that the U-Boot has at most 524288 bytes. This value must be a multiple of 4 bytes.
0x50	Source address	This value indicates the starting address of the special U-Boot code as an offset from the EEPROM starting address. This value must be a multiple of 4 bytes.

Table 6. EEPROM configuration file details (continued)

Value at address:	Description	Comments/Requirements
0x58	Address in DDR memory where a booting image and the RAM-based U-Boot code are copied to.	If using the default Freescale LTIB or BSP package, keep this value unchanged in the configuration file. This value should match the U-Boot configuration, and is the first data/instruction location of the U-Boot.
0x60	Execution starting address	This value is the first instruction of the U-Boot to be executed.
0x68	Number of configuration data pairs in the subsequent data structure section	—

Note:

3 Building a RAM-based U-Boot under Linux

Both SD card/MMC and EEPROM booting use the same RAM-based U-Boot image. A RAM-based U-Boot is different from a NOR Flash-based U-Boot in the following ways:

- A NOR Flash can perform random accesses, but an SD card/MMC or EEPROM cannot be accessed directly.
- The starting address of the RAM-based U-Boot is different than a NOR Flash-based U-Boot.After copying the U-Boot image to offset 0x58 in the control words section of the data structure, the on-chip ROM jumps to the location specified at offset 0x60.

The requirements for building a RAM-based U-Boot under Linux are as follows:

- Use a compile time header file to assign the starting location for a U-Boot. For example, in the MPC8536E BSP, file *u-boot.lds* ensures that the U-Boot starts running from the location at 0xF8FF_F000; in this case, the value 0xF8FF_F000 must therefore be stored at offset 0x60 in an SD card/MMC or EEPROM.
- The initialization code for the U-Boot must be changed to fit the different U-Boot options. Due to the first entry of TLB1 configuration already in the boot ROM code, the RAM-based U-Boot may need to manage different TLBs or need to change the first entry of the TLB1, if necessary.
- The U-Boot environment variables must be saved to an SD card/MMC or an EEPROM, and the corresponding code dealing with the environment must be changed to save the variables Note that in the Freescale BSP, *cmd_nvedit.c* and *env_common.c* are changed, and *env_sdcard.c* is added to handle the environment variables.

This example shows the procedure for building a RAM-based U-Boot using the MPC8536E BSP already installed on a Linux system.

Example 5. Building a RAM-based U-Boot on the MPC8536E BSP

1. Go to the *ltib* directory.
2. Type *./ltib -c* to bring up the “Freescale MPC8536DS PowerPC Developement Board Configuration” window.
3. Select the “u-boot target board type” menu (shown in Figure 3).

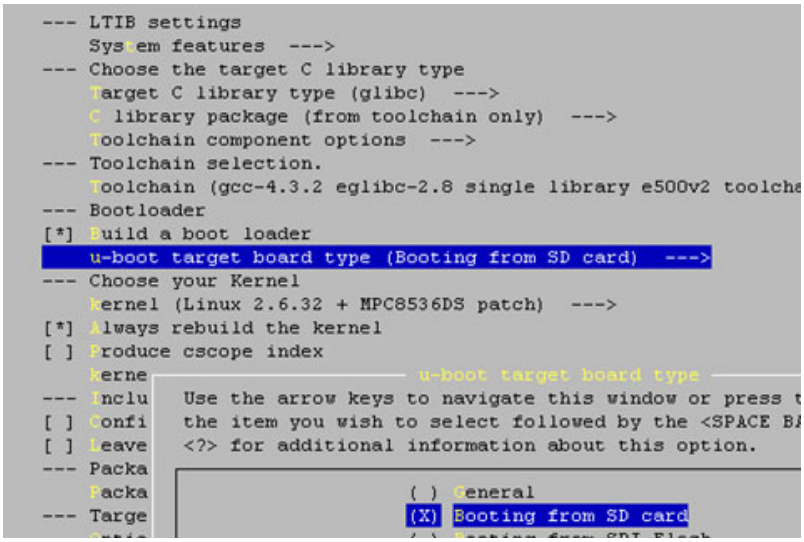

Figure 3. Finding “u-boot target board type”

4. Select “Booting from SD card” or “Booting from SPI Flash.”
5. Exit the window.
6. The U-Boot image file (*u-boot.bin*) is under the directory *./rootfs/boot* after completing the build process properly.

Perform the following sequence of tasks to generate a RAM-based U-Boot to boot from an SD card without a BSP:

Example 6. Building a RAM-based U-Boot from an SD card

1. Add `MPC8536DS_SDCARD_config \` after the `MPC8536DS_NAND_config \` to the *Makefile*.
2. Include `#define CONFIG_MK_SDCARD 1` before `#include <configs/MPC8536DS.h>` in file *./include/config.h*.
3. Add the following to file *./include/configs/mpc8536ds.h*:

```
#ifdef CONFIG_MK_SDCARD
#define CONFIG_RAMBOOT_SDCARD 1
#define CONFIG_RAMBOOT_TEXT_BASE 0xf8f8000
#endif
```

4. Append the following to the line `#if defined (CONFIG_sysy_spl) || defined (CONFIG_RAMBOOT_NAND)` in file *./include/configs/mpc8536ds.h*:

```
|| defined (CONFIG_RAMBOOT_SDCARD)
```

and before line `#define CONFIG_SYS_RAMBOOT`

5. Add the following to file *./board/freescale/mpc8536ds/config.mk* before `ifndef TEXT_BASE`

```
ifeq ($(CONFIG_MK_SDCARD), y)
TEXT_BASE = $(CONFIG_RAMBOOT_TEXT_BASE)
RESET_VECTOR_ADDRESS = 0xf8ffffc
endif
```

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

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

7. Type the following commands:

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

8. Use the U-Boot image in the current directory to boot from an SD card/MMC.

Example 7. Manually building a RAM-based U-Boot from an EEPROM

To manually build a RAM-base U-Boot to boot from an EEPROM, use the procedure in Example 6, but replace “SDCARD” with “SPIFLASH”.

TIP

Check the *mkconfig* file under the *ltib* directory to find out more about the make process.

Example 8. Building a regular U-Boot

To build a regular U-Boot, use the procedure in Example 6, but replace “`MPC8536DS_SDCARD_config`” with “`85xxmake MPC8536DS_config`”.

4 Preparing the image using *boot_format*

boot_format is a booting utility application.

- When booting from an SD card/MMC, *boot_format* puts the configuration file and the RAM-based U-Boot image on the card (Section 4.3, “Putting a boot image on an SD card/MMC”).
- When booting from an EEPROM, *boot_format* generates a binary image that is used to boot from this EEPROM(Section 4.4, “Generating a binary file for eSPI booting,” and Section 4.5, “Putting a boot image on an EEPROM”).

boot_format runs under a regular Linux machine and requires a super user mode to run. After typing *boot_format*, the following information displays:

```
[root@b08938-02 new_tool]# ./boot_format
 Usage: ./boot_format config_file image -sd dev [-o out_config] | -spi out_image
      Where:
          config_file: includes boot signature and configuration words
          image: the U-Boot image for booting from eSDHC/eSPI
          dev: SDCard’s device node (e.g. /dev/sdb, /dev/mmcblk0)
          out_image: boot image in SPI mode
          out_config: modified configure file for SD mode
```

There are two available revisions of *boot_format*:

- *boot_format* Rev. 1.0 (see Section 4.1, “boot_format rev. 1.0 considerations”)
- *boot_format* Rev. 1.1 (see Section 4.2, “boot_format rev. 1.1 considerations”)

NOTE

The source address value is changed by *boot_format*. Its programmed value on an SD card/MMC is usually not the same value as in the configuration file. *boot_format* Rev. 1.1 may give a different source address value from Rev 1.0.

NOTE

boot_format adjusts the starting address of the space based on the size of the boot loader image and the size of the first partition. As such, *boot_format* changes the source address value to be larger than the first partition size.

4.1 *boot_format* rev. 1.0 considerations

TROUBLE

boot_format Rev 1.0 has a known bug that usually prevents the generated binary image from working when booting from the EEPROM.

The requirements for using *boot_format* Rev. 1.0 when booting from an SD card/MMC are as follows:

- Two partitions should be created for an SD card/MMC before using *boot_format* Rev. 1.0 to put a boot image on the card. Of these partitions, the first must be a FAT16 or FAT32 file system with a size less than 2 Gbytes. The most common use case is to have two partitions, as follows:
  - One FAT16 or FAT32 file system with a size of approximately 300 Mbytes
  - One Ext2 or Ext3 file system of a much larger size than the FAT file system's size

The different images are usually partitioned as follows:

- The boot loader image (which includes the configuration information) is stored in the address space between the first partition and the second partition, or appended right after the first partition.
- The Linux kernel image and the flat device tree file are on the first partition.
- The Linux root file system image is on the second partition.

4.2 *boot_format* rev. 1.1 considerations

The requirements for using *boot_format* Rev. 1.1 when booting from an SD card/MMC are as follows:

- Two partitions must be created for an extended capacity SD card (SDXC), because *boot_format* Rev. 1.1 does not work with the exFAT file system.
- The first partition must be a FAT16 or a FAT32 file system.
- If the size of the first partition is smaller than 2 Gbytes, partition the different images the same way as Rev. 10.
- If the size of the first partition is larger than 2 Gbytes and less than 32 Gbytes, *boot_format* changes the source address value to 4608. This is a reserved area based on *SD Specifications Part 2, File System Version 3.0*.

4.3 Putting a boot image on an SD card/MMC

NOTE

boot_format can only run on a regular Linux machine or a Linux based board that has either an SD card/MMC interface or a USB interface with an SD card/MMC-to-USB converter.

Perform the following sequence of tasks to put a boot image on an SD card/MMC using *boot_format*:

1. Insert an SD card/MMC to the Linux machine.
2. Check whether it is in */dev/sdx (x* should be a character of *a, b, c,...*) or */dev/mmcblk0* if using a Freescale BSP.
3. Copy the application *boot_format* to a directory on the Linux machine.
4. Copy the SD card/MMC configuration file and the U-Boot image to the same directory as *boot_format*.
5. If not logged in as a super user, switch to super user mode using *su*.
6. Type *./boot_format config_file image -sd /dev/sdx*
   or
   /dev/mmcblk0 (depending on where the card found in step 2).

NOTE

This utility may change the source address value at offset 0x50.

4.4 Generating a binary file for eSPI booting

Perform the following sequence of tasks to generate a binary file using *boot_format*:

1. Copy the application *boot_format* to a directory on a Linux machine.
2. Copy the EEPROM configuration file and the U-Boot image to the same directory as *boot_format*.
3. If not logged in as a super user, switch to super user mode using *su*.
4. Type *./boot_format config_file image -spi out_image*.
5. Generate the booting image "*out_image*" that is put on the EEPROM in one of the following ways:
   - 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.

4.5 Putting a boot image on an EEPROM

Section 6.2, "EEPROM booting on an MPC8536DS," gives an example of how to enable the eSPI Linux driver. The property for the node of the eSPI EEPROM must be changed so that it can be writeable.

Perform the following sequence of tasks to put the booting image on an EEPROM:

1. Configure the Linux image with the eSPI driver enabled.
2. Build a device tree including a node for EEPROM (an mtd device).
3. Boot to Linux prompt.
4. Check the mtd device to see the EEPROM.
5. Mount the EEPROM.
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 Required POR configurations for booting from on-chip ROM

The on-chip ROM code does not set up any local access windows (LAWs). Access to the CCSR address space or the L2 cache does not require a LAW. It is the user's responsibility to set up a LAW 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 and Example 3, at least one LAW must be configured for successful booting using DDR as the temporary memory.

Note that any such LAW configured must have the 4 Mbits of the address due to the 512-Mbyte U-Boot.

5.1 Required configurations for SD card/MMC booting

The configuration settings required to boot from an SD card/MMC are as follows:

- Ensure that cfg_rom_loc[0:3] (Boot_Rom_Loc) are driven with a value of 0b0111.
- Only one core can be in booting mode. If your device has multiple cores, all other cores must be in a boot hold-off mode. The CPU boot configuration input, cfg_cpux_boot, should be 0, where *x* is from 1 to *n* (*n* = the number of cores).
- Booting from the eSDHC interface can occur from different SD card slots if multiple SD card slots are designed on the board. In this case, ensure the appropriate SD card/MMC is selected.
  For example, on the MPC8536DS board, bit 7 of the SW8 is used to select which SD/MMC slot is used. If SW8[7] = 1, an SD card/MMC must be put to the external SD card/MMC slot (J1).

TIP

The polarity of the SDHC_CD signal should be active-low.

5.2 Required configurations for EEPROM booting

The configuration settings required to boot from an EEPROM are as follows:

- Ensure that cfg_rom_loc[0:3] (Boot_Rom_Loc) are driven with a value of 0b0110.
- Only one core can be in booting mode. If your device has multiple cores, all other cores must be in a boot hold-off mode. The CPU boot configuration input, cfg_cpux_boot, should be 0, where *x* is from 1 to *n* (*n* = the number of cores).
- The eSPI chip select 0 (SPI_CS[0]) must be connected to the EEPROM that is used for booting. No other chip select can be used for booting. This is because during booting, the eSPI controller is configured to operate in master mode. Booting from the eSPI interface only works with SPI_CS[0].

6 Booting from on-chip ROM on an MPC8536DS

6.1 SD card/MMC booting on an MPC8536DS

Perform the following sequence of tasks to boot from an SD card/MMC on an MPC8536DS:

1. Plug the SD card/MMC in the external SD slot (slot 0).
2. Change bit 5678 of SW2 to 0xB0111.
3. Change bit 1 of SW3 to 0.
4. Change bit 7 of SW8 to 1.
5. Keep the default reset of the software setting.
6. Turn on the power; the U-Boot comes up if everything is done properly.

6.2 EEPROM booting on an MPC8536DS

6.2.1 Putting the booting image on the EEPROM

Perform the following sequence of tasks to put a booting image on an EEPROM on an MPC8536DS:

1. Configure the Linux kernel to turn on the eSPI driver as follows:
   - 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_OF_PARTS 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

Use this table to ensure the required properties have the proper descriptions to achieve device tree binding.

Table 7. Required device tree binding settings

Property	Type	Required description
Compatible	String	fsl,espi
Mode (spi node)	String	cpu
Mode (fsl_m25p80 node)	Integer	0
Modal	String	s25s1128b
Clock-frequency	Integer	Not beyond <80000000>

The default node is as follows:

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

```
espi,num-ss-bits = <4>;
mode = "cpu";

fsl_m25p80@0 {
    #address-cells = <1>;
    #size-cells = <1>;
    compatible = "fsl,espi-flash";
    reg = <0>;
    linux, modalias = “fsl_m25p80”;
    spi-max-frequency = <40000000>; /* input clock */
    partition@u-boot-spi {
        label = “u-boot-spi”;
        reg = <0x00000000 0x00100000>;
    };
    partition@kernel {
        label = "kernel-spi";
        reg = <0x00100000 0x00500000>;
        read-only;
    };
    partition@dtb {
        label = "dtb-spi";
        reg = <0x00600000 0x00100000>;
        read-only;
    };
};
fsl_m25p80@1 {
    compatible = "fsl,espi-flash";
    reg = <1>;
    linux, modalias = “fsl_m25p80”;
    spi-max-frequency = <40000000>;
};
fsl_m25p80@2 {
    compatible = "fsl,espi-flash";
    reg = <2>;
    linux, modalias = “fsl_m25p80”;
    spi-max-frequency = <40000000>;
};

fsl_m25p80@3 {
    compatible = "fsl,espi-flash";
```

```
            reg = <3>;
            linux, modalias = “fsl_m25p80”;
            spi-max-frequency = <40000000>;
        };
};
```

2. Remove the read-only property on the mpc8536ds.dts.
3. Boot the MPC8536DS system until login.
4. Check the mtd device:

```
/root # cat /proc/mtd
dev: size erasesize name
mtd0: 00100000 00010000 "u-boot-spi"
mtd1: 00500000 00010000 "kernel-spi"
mtd2: 00100000 00010000 "dtb-spi"
mtd3: 01000000 00010000 "spi32766.1"
mtd4: 01000000 00010000 "spi32766.2"
mtd5: 01000000 00010000 "spi32766.3"
/root #
```

5. Put the boot image to a directory:
   a) Set up the Ethernet port and gateway:

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

   b) Start the TFTP server on a PC and put the image file on the TFTP root directory.
   c) Start the TFTP client in the MPC8536DS system:

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

6. Mount and write the boot image to the EEPROM:

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

Use the Linux command *dd* to check whether the image has been written to the EEPROM, if desired:

```
dd if=/dev/mtd0 of=a1 bs=256 count=2; od –x a1
```

6.2.2 Booting from the EEPROM on an MPC8536DS

Perform the following sequence of tasks to boot from an EEPROM on an MPC8536DS:

1. Change bit 5678 of SW2 to 0xB0110.
2. Keep the default reset of the software setting.
3. Turn on the power; the U-Boot comes up if everything is done properly.

7 Booting to Linux from an SD card/MMC

To boot to Linux from an SD card/MMC, it is assumed that all following configuration files for booting are in the same directory under a Linux machine:

- RAM-based U-Boot image (*u-boot.bin*)
- Kernel image (*uImage)*
- Flat device tree file (*mpc8536ds.dtb*)
- Root file syste (*rootfs.ext2.gz.uboot*)
- Latest *boot-format*

Perform the following sequence of tasks to boot to Linux from an SD card/MMC; note that the MPC8536DS system is used as an example:

1. Plug an empty SD card/MMC into the Linux machine.
2. Use Linux command *fdisk* to create two partitions: one 512-Mbyte FAT16 and one ext2/ext3 with remainder of the available disk size.
3. Use Linux command *mkfs* to create the FAT file system for the first partition.
4. Use *mkfs* to create the ext2/ext3 file system for the second partitions
5. Follow the procedure in Section 4.3, "Putting a boot image on an SD card/MMC."
6. Use *boot_format* to put the boot image on the card.
7. Put the root file system *(rootfs.ext2.gz.uboot*) on the second partition using the following commands:
   — *dd if=rootfs.ext2.gz.uboot of=rootfs.gz bs=64 skip=1*
   — *gunzip rootfs.gz*
   — *dd if=rootfs of=/dev/sdc2*
8. Mount the FAT system (*mount /dev/sdc1 /mnt/tmp*).
9. Copy the kernel file (*cp uImage /mnt/tmp*) and flat device tree file (*cp mpc8536ds.dtb /mnt/tmp*) to the root directory of the FAT system.
10. Unmount the FAT system (*umount /mnt/tmp*).

TIP

After step 9 is performed properly, all the required files and information are on the SD card/MMC.

11. If a Linux desk PC is used:
    a) Unplug the SD card/MMC from this PC.
    b) Plug the SD card/MMC into a system that is going to boot from this card.
12. Configure the system to boot from an SD card/MMC (see Section 6.1, "SD card/MMC booting on an MPC8536DS")
13. Stop the U-Boot before it loads the Linux kernel by typing any key.
14. Change the *bootcmd* parameter by typing the following:

setenv sdboot 'setenv bootargs root=/dev/mmcblk0p2 rw rootfstype=ext2 rootdelay=5 console=ttyS0,115200;mmcinfo; fatload mmc 0:1 1000000 /uImage; fatload mmc 0:1 c00000 /mpc8536ds.dtb; bootm 1000000 - c00000'

15. Save the *bootcmd* parameter by typing *save*.
16. Continue to boot the system to the Linux prompt by entering *run sdboot*.

If the system boots properly, the login screen shows the following information. Note that this is only some of the information displayed:

```
Device: FSL_ESDHC
Manufacturer ID: 3
OEM: 5344
Name: SD02G
Tran Speed: 25000000
Rd Block Len: 512
SD version 2.0
High Capacity: No
Capacity: 2032664576
Bus Width: 4-bit
reading /uImage.8536

3173024 bytes read
reading /mpc8536ds.dtb

12433 bytes read
WARNING: adjusting available memory to 30000000
## Booting kernel from Legacy Image at 01000000 ...
   Image Name:   Linux-2.6.32-rc5
   Image Type:   PowerPC Linux Kernel Image (gzip compressed)
   Data Size:    3172960 Bytes =  3 MB
   Load Address: 00000000
   Entry Point:  00000000
   Verifying Checksum ... OK
## Flattened Device Tree blob at 00c00000
   Booting using the fdt blob at 0xc00000
   Uncompressing Kernel Image ... OK
Using MPC8536 DS machine description
Memory CAM mapping: 256/256/256 Mb, residual: 256Mb
```

```
Linux version 2.6.32-rc5 (jzhao@xeon2) (gcc version 4.3.2 (GCC) ) #8 Fri Oct 22 08:13:08
CDT 2010

bootconsole [udbg0] enabled

setup_arch: bootmem

mpc8536_ds_setup_arch()

...

sdhci: Secure Digital Host Controller Interface driver

sdhci: Copyright(c) Pierre Ossman

mmc0: SDHCI controller on ffe2e000.sdhci [ffe2e000.sdhci] using DMA

...

/bin/ntpclient: option requires an argument -- 'h'

Usage: /bin/ntpclient [-c count] [-d] [-g goodness] -h hostname [-i interval]

        [-l] [-p port] [-r] [-s]

rebuilding rpm database

PHY: mdio@ffe24520:01 - Link is Up - 1000/Full

ADDRCONF(NETDEV_CHANGE): eth0: link becomes ready


        Welcome to the LTIB Embedded Linux Environment


!!!!! WARNING !!!!!!!


The default password for the root account is: root

please change this password using the 'passwd' command

and then edit this message (/etc/issue) to remove this message


mpc8536ds login: root

Password:
```

8 Revision history

This table provides a revision history for this document.

Table 8. Document revision history

Rev. number	Date	Substantive change(s)
2	06/2012	In step 7 in Section 7, "Booting to Linux from an SD card/MMC," changed the first command to: "*dd if=rootfs.ext2.gz.uboot of=rootfs.gz bs=64 skip=1*".
1	11/2010	• Editorial changes throughout • Updated Freescale BSP information throughout. • Added additional devices that this document supports. • Added Section 1, "What does the on-chip ROM do?." • Added Section 2, "Building a Cconfiguration file." • Added Section 2.4, "Choosing the temporary memory location." • In Table 3, "SD Card/MMC data structure definition and address/data pairs," updated Source Address description, and added a footnote. • Modified Example 1, "Configuration file for SD card/MMC using DDR." • Added Example 2, "Configuration file for SD card/MMC using L2 cache." • Modified Example 3, "Configuration file for EEPROM using DDR." • Added Example 4, "Configuration file for booting using the L2 cache." • Updated Section 2.6, "Building an EEPROM configuration file." • Added Section 7, "Booting to Linux from an SD card/MMC." • Added Section 4.1, "boot_format rev. 1.0 considerations." • Added Section 4.2, "boot_format rev. 1.1 considerations."
0	12/2008	Initial release

How to Reach Us:

Home Page:
freescale.com

Web Support:
freescale.com/support

Document Number: AN3659
Rev. 2
06/2012

freescale™