Freescale Semiconductor
Application Note

Document Number: AN3201
Rev. 0, 06/2007

Using U-boot to Boot From a NAND Flash Memory Device for MPC8313E

by *Nick Spence*
Network Computing Systems Group,
Freescale Semiconductor, Inc.

NOR-based Flash memory devices have traditionally been used for non-volatile storage for a bootloader. The advantages of this type of memory include support for execute-in-place code, random access to memory, and zero error rate.

Because of their higher density and lower cost per byte, NAND Flash technologies are becoming more common. Unfortunately, the technology comes with a major disadvantage, which is the early failure of bits during memory writes. After some number of erase cycles, some bits lose the ability to be programmed to a zero state. Block-based error correction solves this problem, but it prevents random memory accesses for execute-in-place code without a specialized NAND Flash memory controller.

The local bus controller in the PowerQUICC™ MPC8313E processor is enhanced with a specialized NAND Flash control machine (FCM) to provide hardware support for small- and large-page NAND Flash-based memories, including the ability to boot from NAND Flash memory.

This application note provides a brief overview of the FCM operation and describes the implementation of a bootloader using u-boot [1] for the MPC8313 device with a small-page NAND device (Samsung K9F5608U0D and Micron MT29F2G08AACWP).

Contents

freescale™ semiconductor

1 MPC8313 Flash Control Machine

The FCM in the enhanced local bus controller (eLBC) is dedicated to supporting NAND Flash memory devices. It supports 8-bit small-page (512 byte) and 2048-byte large-page devices, offloading all NAND memory access and optionally error checking/correction (ECC) from the main processor to the FCM, including the hardware interfacing.

The FCM works in conjunction with the boot sequencer to enable the NAND Flash memory to store the hardware reset configuration word (HRCW) and stage 0 bootloader in the first 4 Kbytes of the first good block of memory in the NAND Flash memory device. Only the main features of the FCM are described here. For a complete description, refer to the MPC8313E reference manual [2].

1.1 Registers

The FCM is controlled and monitored by a set of new registers in the eLBC block, as well as some existing eLBC registers. These registers are listed in Table 1 and Table 2, respectively.

Table 1. New eLBC Registers Used by the FCM

Register	Description	Offsets
Flash mode register (FMR)	Sets the mode of FCM operation, including the command wait time-out, boot mode, ECC mode, and address length.	0x50E0
Flash instruction register (FIR)	Contains a sequence of up to eight FCM instructions.	0x50E4
Flash command register (FCR)	Contains Flash memory command opcode bytes. It can hold up to a maximum of four such commands.	0x50E8
Flash block address register (FBAR)	Contains the block index.	0x50EC
Flash page address register (FPAR)	Contains the page and column indices.	0x50F0
Flash byte count register (FBCR)	Contains the count of bytes written into or read from the Flash memory device. When this register contains a value of zero, the entire page is read from the Flash memory device. The hardware ECC can be generated/checked only when the entire page is transferred by setting the byte count to 0.	0x50F4

Table 2. Existing eLBC Registers Reused by the FCM

Register	Description	Offsets
Special operation initiation register (LSOR)	Used by software to start a NAND Flash access.	0x5090
Transfer error status register (LTESR)	Indicates the cause of an error or event.	0x50B0
Transfer error check disable register (LTEDR)	Allows event or error checks to be disabled	0x50B4
Transfer interrupt enable register (LTEIR)	Selects events or errors that can generate an eLBC interrupt.	0x50B8
Transfer error attributes register (LTEATR)	Provides information about the error transaction and must be cleared before the next event or error can be captured	0x50BC

1.2 Buffers

The CPU does not directly communicate with the NAND. Instead, it reads and writes to an internal buffer RAM in the FCM. The buffer RAM can hold 4 Kbytes of data and 128 bytes of spare/out-of-band data. The buffer RAM is split into 8 pages when small-page devices (512 byte) are accessed or 2 pages when large-page devices (2048 byte) are accessed. Figure 1 shows how the buffer RAM is organized during normal operation when a small-page Flash memory device is accessed. For small-page devices, the three LSBs of FPAR determine which buffer is used for the data transfer, as the following examples illustrate:

- FPAR[17–21] = 10001. Page 1 is used.
- FPAR[17-21] = 10101. Page 5 is used.

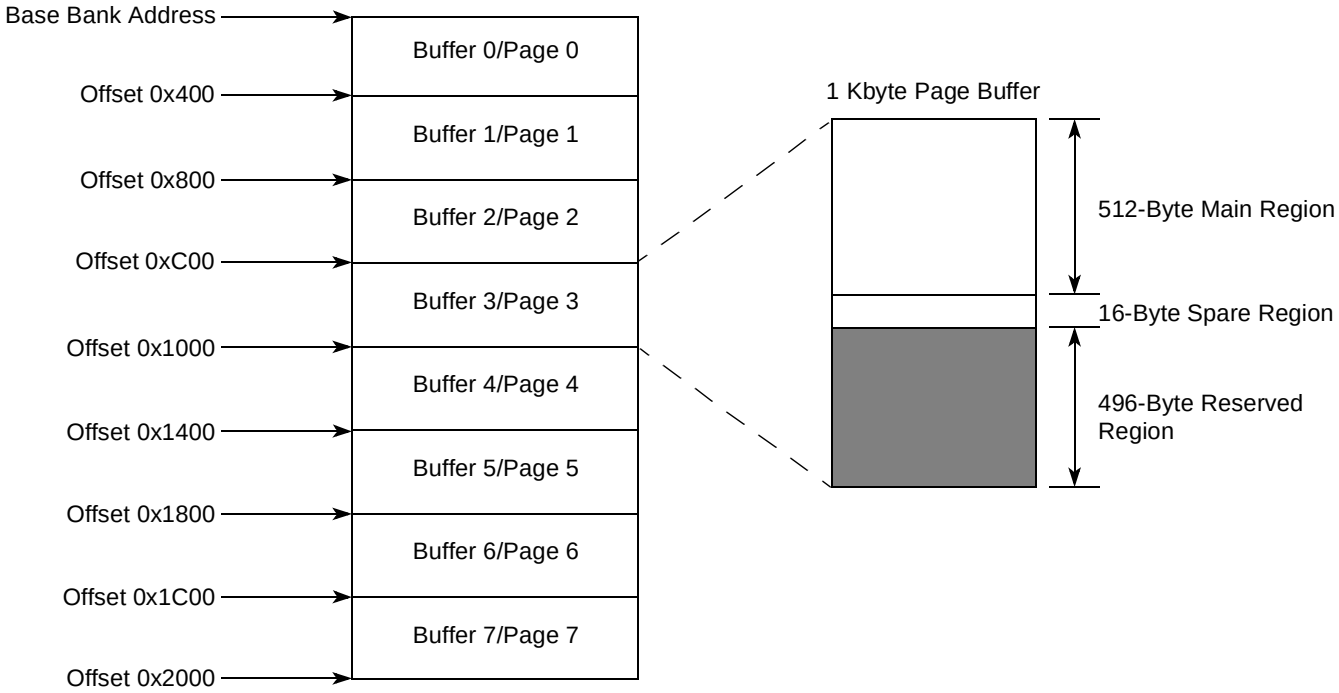

Figure 1. Normal Mode FCM Buffer Layout for Small-Page NAND Flash Devices

The starting address of Buffer 0/Page 0 is set by the base address of the base register for the bank that is programmed to use the NAND FCM.

During boot-block loading, FMR[BOOT] is automatically set, which ensures that the entire buffer appears as a single contiguous 4096 (4 Kbyte) region as shown in Figure 2. To enable normal operation of the FCM buffer, FMR[BOOT] must be cleared. For details on FCM operation and interfacing with the NAND devices, consult the section on the Flash control machine in the chapter on the enhanced local bus controller in [2]

When the NAND Flash memory is used as the boot source, the base address of the 4 Kbyte buffer region is set according to the reset configuration word high Boot Memory Space (BMS) bit. If the BMS bit is cleared, the buffer RAM starts at 0x0000_0000; otherwise, it starts at 0xFFF0_0000. This enables the stage 0 bootloader code to execute in place within the buffer RAM

.

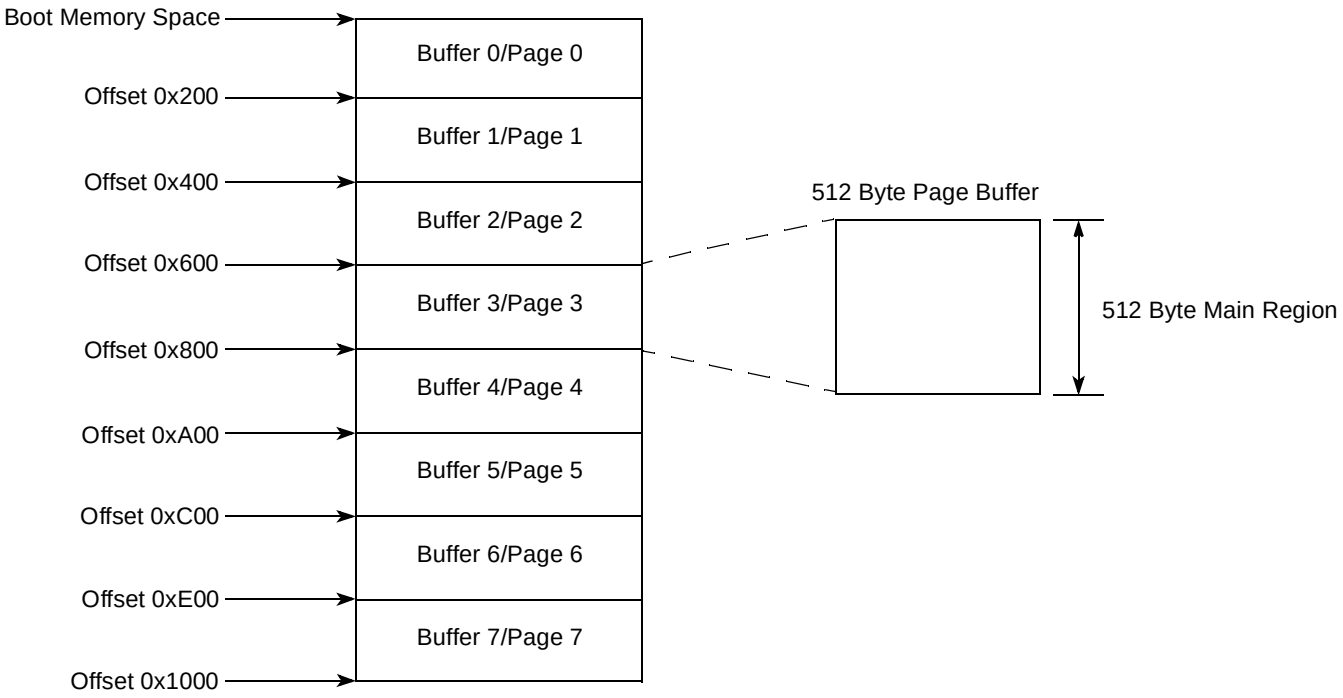

Figure 2. Boot Mode FCM Buffer Layout for Small-Page NAND Flash Memory Devices

1.3 Error Checking and Correction

The technology underlying NAND Flash memory has a natural error rate that is too high for normal use, so it is used with block error checking and correction (ECC) to improve its reliability to a usable level. The block ECC divides each page into one or more sections. When a page is written into NAND Flash memory, each section is processed to generate a 3-byte signature that is written into the spare or out-of-band region of the page. When the page is read back, each section is again processed to generate a 3-byte signature, and the new signature is compared to the original signature for the section read back from the out-of-band region of the page. If the signatures match, the section has no detectable errors. If the signatures differ, either a single-bit error may be corrected or a multiple-bit error is detected but cannot be corrected.

Software can perform ECC operation, usually with a section size of 256 bytes, so for small-page NAND Flash memory devices with 512 data bytes per page, there are two section signatures for each page, using 6 bytes in the out-of-band region of each page. There are two disadvantages to using software ECC:

- The CPU must use processing cycles to calculate the ECC signatures
- There is no software present when the stage 0 bootloader is loaded, so the bootloader page must be error free.

The MPC8313 FCM includes a hardware ECC block to perform ECC generation and correction. This hardware ECC uses a section size of 512 bytes, so for small-page NAND Flash memory devices with 512 data bytes per page, there is 1 section signature for each page, using 3 bytes in the out-of-band region of each page. The hardware ECC is used when the 4 Kbyte stage 0 bootloader is loaded, but it can be enabled or disabled during normal operation.

ECC can correct a maximum of only one bit in each section. If there is more than one error bit, the section is uncorrectable and becomes unusable. When a section of a page becomes unusable, the entire block containing the bad page it is marked as bad to prevent it from being used further. The block is marked bad by setting the bad block marker byte in the out-of-band region of the first 2 pages in the block to a non-0xFF value. Some pages are found to be unusable immediately after manufacture and are marked bad by the manufacturer. Others may become bad after a number of erase/write cycles and may be marked as bad by the NAND Flash control software.

1.4 NAND Boot Sequencer

To select the NAND FCM as the boot code source, configure the reset configuration word high register (RCWHR) RLEXT bit and the ROMLOC field to a value of 1 for a small-page NAND device or 5 for a large-page device. The RCW can also be loaded from a small-page NAND device by configuring the CFG_RESET_SOURCE to a value of 1 for a small-page device or 5 for a large-page device.

The boot sequencer performs the following steps:

1. Set the FMR[BOOT] bit to configure the FCM buffer RAM into a contiguous 4 Kbyte region.
2. Clear the block index to 0.
3. If the current block is marked as bad, increment the block index to the next block and repeat this step.
4. Load the first 4 Kbytes from the current block, correcting any 1-bit errors in each 512 byte block using the hardware ECC. If a block cannot be corrected, assert the HRESET_REQ signal and halt.
5. Start executing instructions at offset 0x100.

Note that the block index in the FBAR register contains the index of the block from which the stage 0 bootloader is loaded. Normally, this is 0, but if the first block is marked as bad, it is set to the index of the first block.

2 U-boot

U-boot [1] is a widely used bootloader that supports a number of processor architectures, including PowerPC™. The code is available under GPL from `www.denx.de` and can be used and distributed without charge. U-boot contains all the code required to initialize the processor and board-specific files that accommodate differences between different boards. Version 1.1.6 adds support for NAND Flash bootloaders in the `nand_spl` directory. This enables both the stage 0 bootloader and the main u-boot images to be built and combined into a single binary file that is written into the NAND Flash memory device.

2.1 Stage 0 Bootloader

The stage 0 bootloader is loaded from the NAND Flash memory into the FCM buffer as described in Section 1.4, “NAND Boot Sequencer,” and then executes from offset 0x100. It must perform the following procedure:

1. Reset and initialize the MPC8313 processor registers and cache and memory access windows.
2. Preserve the reset status register contents.
3. Set up the data cache as a data memory region to store the stack so that C code can be run.

4. Configure the chip select 0 BR and OR registers to set the timing and ECC modes.
5. Initialize DDR memory and set up the DDR access window.
6. Copy the stage 0 bootloader from the FCM buffer into DDR memory so that the FCM buffers can be used to read the rest of u-boot.
7. Clear the FMC[BOOT] bit to return the FCM buffers to normal operation.
8. Read the u-boot image from the NAND Flash memory (skipping bad pages and fixing single-bit errors) and copy it into the DDR memory.
9. Jump to the start address of the u-boot image.

The u-boot image differs slightly from the normal image used in a NOR Flash memory in that the u-boot image is in DDR RAM, which is already initialized by the stage 0 bootloader. It must not be initialized again because this may damage the u-boot image. The u-boot build is configured to skip the DDR initialization step by setting the CFG_RAMBOOT define. The stage 0 bootloader file has two additional functions, console support and software ECC support, which are described in the following sections.

2.1.1 Console Support

U-boot supports a serial console using one of the internal UARTs. The stage 0 bootloader uses a subset of this so that status and error messages can be displayed during the stage 0 bootloader operation. This facilitates debugging, if required.

The stage 0 bootloader prints out the version name of the u-boot image from which it was built, the status of each NAND Flash block read to copy the u-boot image, and any fatal errors. An example of the console transcript is shown here:

```
NAND SPL - U-Boot 1.1.6 (Nov 20 2006 - 09:55:26) MPC83XX
Loading from NAND:.........................
```

When the blocks are read from the NAND Flash memory, a period (.) is normally printed if the block is successfully read. If the block is already marked bad and therefore skipped, a `B` is printed. If the FCM does not get a response from the NAND device, a timeout occurs and `T` is printed and a fatal error signalled. If the page has an uncorrectable error, a `U` is printed and a fatal error signalled. If a fatal error occurs, the processor reboots after a short pause and attempts to run the bootloader.

2.1.2 Software ECC Support

The u-boot NAND Flash drivers include support for software ECC in the file `drivers/nand/nand_ecc.c`, which includes ECC generation, checking, and correction. This file is copied and simplified (to reduce the memory footprint) to provide support for software ECC. The stage 0 bootloader is always loaded using hardware ECC, but this can be used to read and correct the rest of the u-boot image. Software ECC enables us to determine the number of correctable ECC errors that occur while the block is read. If the block is still correctable, the software prints a digit indicating the number of bits corrected.

2.2 Board Configuration Settings for NAND boot

u-boot is configured on a per board basis using the board configuration file (`include/configs/MPC8313RDB.h`). This file defines C preprocessor macros to control the build and

operation of the rest of the C files in u-boot. Table 3 shows the C preprocessor macros used in the NAND boot.

Table 3. Defines for Normal and NAND u-boot

Define	Description
CFG_HRCW_LOW	The clock multiplier configurations
CFG_HRCW_HIGH	Must include HRCWH_FROM_0XFFF00100, HRCWH_ROM_LOC_NAND_SP_8BIT and HRCWH_RL_EXT_NAND
CFG_INIT_RAM_ADDR	Specifies the unmapped location to lock the cache as a temporary RAM
CFG_INIT_RAM_END	Specifies the size of the temporary RAM created from the D cache
CFG_IMMR	The address to be used for the IMMR space
CFG_IBATxx and CFG_DBATxx	Instruction and data BAT values
CFG_LCRR	Local bus configuration
CFG_NAND_BR0_PRELIM	CS0 base register. This must set the desired error correction mode in the DECC field.
CFG_NAND_OR0_PRELIM	CS0 option register. This selects the NAND Flash page size.
CFG_NAND_LBLAWBAR0_PRELIM	Set to the same NAND base address as CFG_NAND_BR0_PRELIM
CFG_NAND_LBLAWAR0_PRELIM	Set to the smallest supported window size (32 KBytes)
CONFIG_83XX_CLKIN	The CLK_IN rate in Hz for standalone or PCI host configurations
CONFIG_83XX_PCICLK	The PCI CLK rate in Hz, used for PCI agent configuration
CONFIG_CONS_INDEX	Index of the UART used for console messages (1 or 2)
CFG_NS16550_COMn	Enables the configuration of UART n = 1 or 2
CONFIG_BAUDRATE	Console baudrate
CFG_NAND_BASE	Set to the same NAND base address as CFG_NAND_BR0_PRELIM

Table 4 shows the new C preprocessor macros that are used only in the NAND boot code.

Table 4. Defines for NAND u-boot

Define	Description
CFG_NAND_BOOT_QUIET	Suppress stage 0 bootloader console messages, except fatal errors
CFG_NAND_RELOC	Address to copy the stage 0 bootloader to in DRAM (normally 0x10000)
CFG_NAND_BOOT_SHOW_ECC_NONE	Ignore ECC status when printing the block status character
CFG_NAND_BOOT_SHOW_ECC_NUM	Report number of errors fixed by SW ECC in each block
CFG_NAND_BLOCK_SIZE	Number of data bytes in each NAND Flash block
CFG_NAND_PAGE_SIZE	Number of data bytes in each NAND Flash page
CFG_NAND_BAD_BLOCK_POS	Byte position of the bad block marker in the out-of-band region (5 for small page devices)
CFG_NAND_U_BOOT_DST	Location in DDR to copy the main u-boot image

Table 4. Defines for NAND u-boot (continued)

Define	Description
CFG_NAND_U_BOOT_SIZE	Size of the main u-boot image in bytes
CFG_NAND_U_BOOT_START	Start address to jump to in main u-boot

2.3 u-boot files for NAND Boot

The stage 0 bootloader uses a number of files that already exist in u-boot. The board configuration file is `include/configs/MPC8313ERDB.h` and the DDR SDRAM initialization file is `board/mpc8313erdb/sdram.c`. These files make it easier to maintain consistency between the regular u-boot build and the NAND bootloader build. See Table 5.

Table 5. u-boot Files for NAND Boot

File	Description
`cpu/mpc83xx/start.S`	Processor initialization and configuration. Cache and BAT setup and code relocation (shared with normal u-boot)
`board/mpc8313erdb/nand_boot.c`	Reset, console, NAND read, chip select initialization, and control functions (specific to NAND u-boot).
`board/mpc8313erdb/nand_ecc.c`	Software ECC functions optimized from drivers/nand/nand_ecc.c (specific to NAND u-boot)
`board/mpc8313erdb/sdram.c`	SDRAM initialization and configuration (shared with normal u-boot)
`include/configs/MPC8313ERDB.h`	Board configuration settings (shared with normal u-boot)
`nand_spl/board/mpc8313erdb/config.mk`	Build base address configuration (specific to NAND u-boot)
`nand_spl/board/mpc8313erdb/Makefile`	Build control (specific to NAND u-boot)
`nand_spl/board/mpc8313erdb/u-boot.lds`	Linker control (specific to NAND u-boot)

2.4 NAND u-boot Execution Sequence

Table 6 lists and describes the functions executed by the NAND u-boot.

Table 6. NAND u-boot Execution Sequence

File	Description
`cpu/mpc83xx/start.S/HRCW`	Hardware reset configuration words, CFG_HRCW_LOW and CFG_HRCW_HIGH
`cpu/mpc83xx/start.S/_start`	Execution starts at offset 0x100 and stores the BOOTFLAG_COLD value. The IMMR value is set.
`cpu/mpc83xx/start.S/init_e300_core`	Enables the machine check interrupt, configures the watchdog timer, sets the HID0 and HID2 registers, clears BATS 0 to 3, and invalidates the TLBs
`cpu/mpc83xx/start.S/setup_bats`	Loads instruction and data BATs with configured values for BATS 0 to 7. This specifies the cacheability of blocks.
`cpu/mpc83xx/start.S/enable_addr_trans`	Enables address translation for instruction and data spaces by setting the IR and DR bits in the MSR so that the BATs are used.

Table 6. NAND u-boot Execution Sequence (continued)

File	Description
cpu/mpc83xx/start.S/dcache_enable	Invalidates the data cache contents and enables the cache.
cpu/mpc83xx/start.S/lock_ram_in_cache	Maps the data cache to a non-existent memory space and locks it so that it can be used as a RAM to hold the stack and global data.
board/mpc8313erdb/nand_boot.c/cpu_init_f	Stores the reset mode register value. Configures the local bus clock, chip select 0 BR and OR values, and the local address windows. Determines the CSB clock rate.
board/mpc8313erdb/nand_boot.c/NS16550a_init	If a console port is identified, configures it to the specified baud rate.
board/mpc8313erdb/sdram.c/initdram	Initializes the DDR memory. This code is shared with the normal u-boot code.
cpu/mpc83xx/start.S/relocate_code	Copies the stage 0 bootloader to DDR, fixes the global offset table pointers that point to the strings used by puts, resets the stack into DDR, and clears the caches.
board/mpc8313erdb/nand_boot.c/cpu_init_r	Controls the loading of the u-boot image from NAND Flash memory into DDR memory.
cpu/mpc83xx/start.S/icache_enable	Enables the instruction cache to speed up the stage 0 bootloader.
board/mpc8313erdb/nand_boot.c/nand_read_next_block	Uses the FCM to read the next block and copy it to DDR. If the block is marked as bad, it skips the block. If the block has uncorrectable errors, it returns a fatal error code. If console messages are enabled, it prints a status character for the page. Note this starts at loading the image from the block following the block that contains the stage 0 bootloader.
board/mpc8313erdb/nand_ecc.c/nand_correct_data	If hardware ECC is not selected in the BR0 setting, this function is called to perform software ECC of the page read from NAND Flash memory.
board/mpc8313erdb/nand_ecc.c/reset	If a fatal error occurs during bootloading, an error message is printed. This function waits for a short period so that the message can be read, then generates a hardware reset.
cpu/mpc83xx/start.S/boot_warm	If the stage 0 bootloader completes successfully, it jumps to the boot_warm location of the normal u-boot image and runs u-boot.

3 Building and Loading NAND u-boot

Because the bootloader is integrated into u-boot, it is quite easy to build. Start by making the changes to the configuration file and the board-specific directory to support your board. To test these changes, build a regular u-boot image, load it into DRAM, and execute it. The NAND Flash image can be built using the following instructions:

```
make CROSS_COMPILE=powerpc-linux- MPC8313ERDB_NAND_config
make
```

This produces the `u-boot-nand.bin` file that can be loaded into the NAND Flash memory. You can program the Flash memory using a normal u-boot image that is loaded into DDR on the board, with a toolchain such as CodeWarrior™ over a COP debugger or with a NAND Flash programmer. For the code to execute correctly, it must be written to Flash memory with the correct ECC signatures. The first page, which contains the stage 0 bootloader, must be written with hardware ECC enabled. The rest of the image must

be written with hardware ECC if the BR0[DECC] value is non-zero. If the value is zero, it must be written with software ECC.

4 NAND Auto Boot Settings on RDB Board

Set DIP switch S4 as OFF-OFF-OFF-ON (1110) and set DIP switch S3 as ON-ON-ON-OFF (0001). Note that there is no boot image on NAND Flash memory with the default shipment.

5 Conclusion

This application note describes the basic operation of the MPC8313 NAND Flash control machine (FCM) and its use with u-boot to support automatic booting from a NAND Flash memory device. The source code is available as part of the u-boot source code which is freely available and supported on the Freescale MPC8313 RDB board.

6 References

1. Wolfgang Denk et al. *Das U-boot: A Universal Boot Loader*, available at `http://u-boot.sourceforge.net`.
2. MPC8313E *PowerQUICC™ II Pro Integrated Host Processor Reference Manual*, available at the Freescale web site.

Appendix A NAND u-boot Macros

The following macros are included in include/configs/MPC8313ERDB.h to configure the NAND boot operation:

```
#ifdef CONFIG_NAND_SPL
#define CFG_NAND_BASE0xFFF00000
#else
#define CFG_NAND_BASE0xE2800000
#endif

#define CFG_NAND_BR_PRELIM(CFG_NAND_BASE \
                                    | (2<<BR_DECC_SHIFT) /* Use HW ECC */ \
                                    | BR_PS_8            /* Port Size = 8 bit */ \
                                    | BR_MS_FCM          /* MSEL = FCM */ \
                                    | BR_V)             /* valid */

#define CFG_NAND_OR_PRELIM( 0xFFFF8000/* length 32K */ \
                                    | OR_FCM_CSCT \
                                    | OR_FCM_CST \
                                    | OR_FCM_CHT \
                                    | OR_FCM_SCY_1 \
                                    | OR_FCM_TRLX \
                                    | OR_FCM_EHTR )
                                   /* 0xFFFF8396 */

#define CFG_NAND_BR0_PRELIMCFG_NAND_BR_PRELIM
#define CFG_NAND_OR0_PRELIMCFG_NAND_OR_PRELIM
#define CFG_NAND_LBLAWBAR0_PRELIMCFG_NAND_BASE
#define CFG_NAND_LBLAWAR0_PRELIM0x8000000E   /* 32KB  */

#undef  CFG_NAND_BOOT_QUIET                 /* Enable NAND boot status messages */
#define CFG_NAND_BOOT_SHOW_ECC_NUM          /* Show corrected ECC errors */
#define CFG_NAND_RELOC(0x10000)             /* Stage 1 load address */
#define CFG_NAND_PAGE_SIZE(512)             /* NAND chip page size */
#define CFG_NAND_BLOCK_SIZE(16 << 10)       /* NAND chip block size = 16 KBytes */
#define CFG_NAND_BAD_BLOCK_POS(5)           /* Bad block marker location */
#define CFG_NAND_FMR((14 << FMR_CWTO_SHIFT) | (0 << FMR_AL_SHIFT))

#define CFG_NAND_U_BOOT_SIZE(384 << 10)     /* Size of RAM U-Boot image */
#define CFG_NAND_U_BOOT_DST(0x01000000)     /* Load NUB to this addr */
#define CFG_NAND_U_BOOT_START(CFG_NAND_U_BOOT_DST + 0x120) /* NUB start */
```

Appendix B NAND u-boot Code

The following code is copied from the board/mpc8313erdb/nand_boot.c file, which manages the boot cycle.

```
#include <common.h>
#include <mpc83xx.h>
#include <ns16550.h>
#include <nand.h>
#include <asm/processor.h>

/* NAND ECC checking method - 0 = no hardware ECC check */
#define NAND_HARD_ECC ((CFG_NAND_BR0_PRELIM >> BR_DECC_SHIFT) & 3)

/* NAND Page Size : 0 = small page (512 bytes ), 1 = large page (2048 bytes) */
#define NAND_PGS ((CFG_NAND_OR0_PRELIM >> OR_FCM_PGS_SHIFT) & 1)

/* Timeout in case FCM does not complete */
#define NAND_TIMEOUT (1000000)

/* Delay before restarting after a fatal u-boot error */
#define RESTART_DELAY (0x4000000)

/* Error codes returned from nand_read_next_block()                */
#define NAND_OK          (1)/* read block okay                 */
#define NAND_BAD_BLOCK   (0)/* block marked bad - skip block   */
#define NAND_ERR_TIMEOUT (-1)/* timeout error - fatal error     */
#define NAND_ERR_ECC     (-2)/* uncorrectable ecc - fatal error */

/* Macros to control selected serial port */
#if CONFIG_CONS_INDEX == 1 && defined(CFG_NS16550_COM1)
#define NS16550_COM ((NS16550_t)CFG_NS16550_COM1)
#elif CONFIG_CONS_INDEX == 2 && defined(CFG_NS16550_COM2)
#define NS16550_COM ((NS16550_t)CFG_NS16550_COM2)
#else
#warning  "****************************"
#warning  "** No console port defined **"
#warning  "****************************"
#define NS16550_COM ((NS16550_t)0)
#define CFG_NAND_BOOT_QUIET
#endif /* CONFIG_CONS_INDEX */

/* Quiet Boot - only prints fatal error messages */
#if defined(CFG_NAND_BOOT_QUIET)
#define status_putc(c) { while (0); }
#define status_puts(s) { while (0); }
#else
#define status_putc(c) { putc(c); }
#define status_puts(s) { puts(s); }
#endif /* CFG_NAND_BOOT_QUIET */

#if !(NAND_HARD_ECC)
const u_char ecc_pos[] = {
#if (NAND_PGS)
        40, 41, 42, 43, 44, 45, 46, 47,
        48, 49, 50, 51, 52, 53, 54, 55,
```

```
        56, 57, 58, 59, 60, 61, 62, 63
#else
        0, 1, 2, 3, 6, 7
#endif /* NAND_PGS */
};
#endif /* !(NAND_HARD_ECC) */

/* u-boot version string from start.S */
extern char version_string[];

/* nand_ecc.c */
extern int nand_correct_data (u_char * dat, const u_char * ecc_pos, int blocks);

/* reset board */
static
void reset (void)
{
        int ctr;
        ulong msr;
#ifndef MPC83xx_RESET
        ulong addr;
#endif

        volatile immap_t *immap = (immap_t *) CFG_IMMR;

        /* add delay before reseting */
        ctr = RESTART_DELAY;
        while (ctr--) ;

#ifdef MPC83xx_RESET
        /* Interrupts and MMU off */
        __asm__ __volatile__ ("mfmsr    %0":"=r" (msr):);

        msr &= ~( MSR_EE | MSR_IR | MSR_DR);
        __asm__ __volatile__ ("mtmsr    %0"::"r" (msr));

        /* enable Reset Control Reg */
        immap->reset.rpr = 0x52535445;
        __asm__ __volatile__ ("sync");
        __asm__ __volatile__ ("isync");

        /* confirm Reset Control Reg is enabled */
        while(!((immap->reset.rcer) & RCER_CRE));

        /* perform reset, only one bit */
        immap->reset.rcr = RCR_SWHR;

#else   /* ! MPC83xx_RESET */

        immap->reset.rmr = RMR_CSRE;    /* Checkstop Reset enable */

        /* Interrupts and MMU off */
        __asm__ __volatile__ ("mfmsr    %0":"=r" (msr):);

        msr &= ~(MSR_ME | MSR_EE | MSR_IR | MSR_DR);
        __asm__ __volatile__ ("mtmsr    %0"::"r" (msr));
```

```
        /*
         * Trying to execute the next instruction at a non-existing address
         * should cause a machine check, resulting in reset
         */
        addr = CFG_RESET_ADDRESS;
        ((void (*)(void)) addr) ();
#endif  /* MPC83xx_RESET */
}

#define LCRVAL LCR_8N1              /* 8 data, 1 stop, no parity */
#define MCRVAL (MCR_DTR | MCR_RTS)/* RTS/DTR */
#define FCRVAL (FCR_FIFO_EN | FCR_RXSR | FCR_TXSR) /* Clear & enable FIFOs */

static
void NS16550a_init (int baud_divisor)
{
        if (NS16550_COM) {
                NS16550_COM->ier = 0x00;
                NS16550_COM->lcr = LCR_BKSE | LCRVAL;
                NS16550_COM->dll = baud_divisor & 0xff;
                NS16550_COM->dlm = (baud_divisor >> 8) & 0xff;
                NS16550_COM->lcr = LCRVAL;
                NS16550_COM->mcr = MCRVAL;
                NS16550_COM->fcr = FCRVAL;
        }
}

/* print a single character, with an extra line feed for return characters */
void putc (const char c)
{
        if (NS16550_COM) {
                if (c == '\n') {
                        while ((NS16550_COM->lsr & LSR_THRE) == 0);
                        NS16550_COM->thr = '\r';
                }
                while ((NS16550_COM->lsr & LSR_THRE) == 0);
                NS16550_COM->thr = c;
        }
}

/* print an entire null terminated string */
void puts (const char *s)
{
        while (*s) {
                putc (*s++);
        }
}

/* read the next block from NAND flash and store it at the supplied address
 *
 * return values:
 *  NAND_OK          - block was successfully read and copied to the destination
 *  NAND_BAD_BLOCK   - block was marked bad so should be skipped
 *  NAND_ERR_TIMEOUT - page read did not complete (fatal error)
 *  NAND_ERR_ECC     - uncorrectable ECC (fatal error)
 */
static
```

```
int nand_read_next_block (unsigned int *dst)
{
        volatile immap_t *im = (immap_t *) CFG_IMMR;
        volatile lbus83xx_t *lbc = &im->lbus;
        int buf_num;
        int page;
        unsigned char *srcc;
        unsigned int *src;
        int ecc_err;
        int ctr;
        unsigned int status;
#if !(NAND_HARD_ECC)
        int ecc_cnt;
        char ecc_char;

        ecc_cnt = 0;
#endif /* !(NAND_HARD_ECC) */

        ecc_err = 0;
        srcc = 0;

        /* Enable FCM detection of timeouts, ECC errors and completion */
        lbc->ltedr = 0;

        lbc->fbcr = 0;    /* read entire page to enable ECC */
        lbc->fbar++;      /* read next block, follows boot loaded block */
#if (NAND_PGS)
        lbc->fir = (FIR_OP_CW0 << FIR_OP0_SHIFT) |
            (FIR_OP_CA << FIR_OP1_SHIFT) |
            (FIR_OP_PA << FIR_OP2_SHIFT) |
            (FIR_OP_CW1 << FIR_OP3_SHIFT) |
            (FIR_OP_RBW << FIR_OP4_SHIFT);
#else
        lbc->fir = (FIR_OP_CW0 << FIR_OP0_SHIFT) |
            (FIR_OP_CA << FIR_OP1_SHIFT) |
            (FIR_OP_PA << FIR_OP2_SHIFT) |
            (FIR_OP_RBW << FIR_OP3_SHIFT);
#endif /* NAND_PGS */
        lbc->fcr = (NAND_CMD_READ0 << FCR_CMD0_SHIFT) |
            (NAND_CMD_READSTART << FCR_CMD1_SHIFT);

        /* read in each page of the block */
        for (page = 0; page < (CFG_NAND_BLOCK_SIZE / CFG_NAND_PAGE_SIZE);
             page++) {
                if (NAND_PGS) {
                        lbc->fpar = ((page << FPAR_LP_PI_SHIFT) & FPAR_LP_PI);
                        buf_num = (page & 1) << 2;
                } else {
                        lbc->fpar = ((page << FPAR_SP_PI_SHIFT) & FPAR_SP_PI);
                        buf_num = page & 7;
                }
                lbc->fmr = CFG_NAND_FMR | 2;

                /* clear event registers */
                lbc->ltesr = lbc->ltesr;
                lbc->lteatr = 0;
```

```
                /* execute special operation on bank 0 */
                lbc->lsor = 0;
                asm ("sync");

                /* copy previous page to RAM */
                if (srcc) {
#if !(NAND_HARD_ECC)
                        status =
                            nand_correct_data (srcc, ecc_pos,
                                               sizeof (ecc_pos) / 3);
                        ecc_cnt += status;
                        if (status < 0)
                                ecc_err = 1;
#endif /* !(NAND_HARD_ECC) */
                        src = (unsigned int *)srcc;
                        for (ctr = CFG_NAND_PAGE_SIZE / sizeof (unsigned int);
                             ctr; ctr--) {
                                *(dst++) = *(src++);
                        }
                }

                /* store the source address for the next page */
                srcc = (unsigned char *)((CFG_NAND_BR0_PRELIM & BR_BA) +
                                         (buf_num * 1024));

                /* wait for FCM complete flag or timeout */
                status = 0;
                for (ctr = NAND_TIMEOUT; ctr; ctr--) {
                        status = lbc->ltesr;
                        if (status) {
                                break;
                        }
                }

                /* check for timeout or hardware ECC errors */
                if (status != LTESR_CC) {
#if (NAND_HARD_ECC)
                        if (status & LTESR_PAR) {
                                ecc_err = 1;
                        } else
#endif /* NAND_HARD_ECC */
                        {
                                status_putc ('T');
                                return NAND_ERR_TIMEOUT;
                        }
                }

                /* Check if the block is marked as bad */
                if (page < 2) {
                        if (srcc[CFG_NAND_PAGE_SIZE + CFG_NAND_BAD_BLOCK_POS] !=
                            0xFF) {
                                status_putc ('B');
                                return NAND_BAD_BLOCK;
                        }
                }
        }
```

```
        /* copy last page to RAM */
#if !(NAND_HARD_ECC)
        status = nand_correct_data (srcc, ecc_pos, sizeof (ecc_pos) / 3);
        ecc_cnt += status;
        if (status < 0)
                ecc_err = 1;
#endif /* !(NAND_HARD_ECC) */
        src = (unsigned int *)srcc;
        for (ctr = CFG_NAND_PAGE_SIZE / sizeof (unsigned int); ctr; ctr--) {
                *(dst++) = *(src++);
        }

        /* abort if any of the pages had uncorrectable errors */
        if (ecc_err && (page > 1)) {
                status_putc ('U');
                return NAND_ERR_ECC;
        }
#if (NAND_HARD_ECC)
        status_putc ('.');
#else
#ifdef CFG_NAND_BOOT_SHOW_ECC_NONE
        ecc_char = '.';
#else
        if (ecc_cnt <= 0) {
                ecc_char = '.';
#ifdef CFG_NAND_BOOT_SHOW_ECC_NUM
        } else if (ecc_cnt <= 9) {
                ecc_char = '0' + ecc_cnt;
        } else {
                ecc_char = 'a' + ecc_cnt - 10;
#else
        } else {
                ecc_char = 'c';
#endif /* CFG_NAND_BOOT_SHOW_ECC_NUM */
        }
#endif /* CFG_NAND_BOOT_SHOW_ECC_NONE */
        status_putc (ecc_char);
#endif /* NAND_HARD_ECC */

        return NAND_OK;   /* block read completed ok */
}

/* initial C code called from start.S prior to relocating code to DDR
 *
 * This performs minimal CPU initailization, DDR initialization, a few
 * print statements and the calls relocate_code() to copy the code from
 * the NAND flash buffer to DDR.
 */
void cpu_init_f (volatile immap_t * im)
{
        u8 spmf;
        u8 clkin_div;
        u32 csb_clk;

        /* RMR - Reset Mode Register - enable checkstop reset enable */
        im->reset.rmr = (RMR_CSRE & (1 << RMR_CSRE_SHIFT));
```

```
        /* LCRR - Clock Ratio Register - set up local bus timing */
        im->lbus.lcrr = CFG_LCRR;

#if defined(CFG_NAND_BR0_PRELIM)  \
        && defined(CFG_NAND_OR0_PRELIM) \
        && defined(CFG_NAND_LBLAWBAR0_PRELIM) \
        && defined(CFG_NAND_LBLAWAR0_PRELIM)
        im->lbus.bank[0].br = CFG_NAND_BR0_PRELIM;
        im->lbus.bank[0].or = CFG_NAND_OR0_PRELIM;
        im->sysconf.lblaw[0].bar = CFG_NAND_LBLAWBAR0_PRELIM;
        im->sysconf.lblaw[0].ar = CFG_NAND_LBLAWAR0_PRELIM;
#else
#error  CFG_NAND_BR0_PRELIM, CFG_NAND_OR0_PRELIM, CFG_NAND_LBLAWBAR0_PRELIM &
CFG_NAND_LBLAWAR0_PRELIM must be defined
#endif
        clkin_div = ((im->clk.spmr & SPMR_CKID) >> SPMR_CKID_SHIFT);
        spmf = ((im->reset.rcwl & RCWL_SPMF) >> RCWL_SPMF_SHIFT);

        if (im->reset.rcwh & HRCWH_PCI_HOST) {
#if defined(CONFIG_83XX_CLKIN)
                csb_clk = CONFIG_83XX_CLKIN * spmf;
#else
                csb_clk = 0;
#endif /* CONFIG_83XX_CLKIN */
        } else {
#if defined(CONFIG_83XX_PCICLK)
                csb_clk = CONFIG_83XX_PCICLK * spmf * (1 + clkin_div);
#else
                csb_clk = 0;
#endif /* CONFIG_83XX_PCICLK */
        }

        /* initialize selected port with appropriate baud rate */
        NS16550a_init (csb_clk / 16 / CONFIG_BAUDRATE);

        status_puts ("\nNAND SPL - ");
        status_puts ((char *)(&version_string));

        /* board specific DDR initialization */
        initdram (0);

        /* copy code to DDR and jump to it - this should not return */
        /* NOTE - code has to be copied out of NAND buffer before
         * other blocks can be read.
         */
        relocate_code (CFG_NAND_RELOC + 0x10000, 0, CFG_NAND_RELOC);

        /* should never get here */
        puts ("\nRelocate failed\n");

        /* delay then restart */
        reset ();
}

/* called after code is moved to DDR, to complete boot loading */
void board_init_r (gd_t * id, ulong dest_addr)
{
```

```
        int blockcopy_count;
        unsigned char *dst;
        void (*uboot) (void* dummy, void* immr);
        int ret;

        icache_enable ();/* faster execution */

        status_puts ("\nLoading from NAND : ");

        /*
         * Load U-Boot image from NAND into RAM
         */
        dst = (unsigned char *)CFG_NAND_U_BOOT_DST;
        blockcopy_count = ((CFG_NAND_U_BOOT_SIZE + CFG_NAND_BLOCK_SIZE - 1)
                           / CFG_NAND_BLOCK_SIZE);

        while (blockcopy_count) {
                ret = nand_read_next_block ((unsigned int *)dst);
                switch (ret) {
                case NAND_OK:
                        /* advance to the next block */
                        dst += CFG_NAND_BLOCK_SIZE;
                        blockcopy_count--;
                        break;
                case NAND_BAD_BLOCK:
                        /* skip bad block */
                        break;
                default: /* fatal error */
#if defined(CFG_NAND_BOOT_QUIET)
                        puts ("\nNAND SPL - ");
#else
                        putc ('\n');
#endif /* CFG_NAND_BOOT_QUIET */
                        if (ret == NAND_ERR_TIMEOUT)
                                puts ("**FATAL** : NAND Flash operation timeout\n");
                        else
                                puts ("**FATAL** : uncorrectable ECC Error\n");

                        /* delay then restart */
                        reset ();
                        break;
                }
        }

        /*
         * Jump to U-Boot image
         */
        uboot = (void (*)(void* dummy, void* immr))CFG_NAND_U_BOOT_START;
        (*uboot) (NULL, (void*) CFG_IMMR);
}
```

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 Hong Kong Ltd.
Technical Information Center
2 Dai King Street
Tai Po Industrial Estate
Tai Po, N.T., Hong Kong
+800 2666 8080
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

Document Number: AN3201
Rev. 0
06/2007