

# Application note

---

## TI DM644x DaVinci NAND FLASH Programming with PEEDI

Version 1.0

---

# 1. Preface

NAND FLASH memory chips are considerably cheaper than their elder brothers – the NOR FLASH chips. So NAND chips started to push away NOR chips in embedded devices and now it is possible to build a device without a NOR FLASH – only using a NAND flash which is used to hold the bootloader and all other executable code that is copied and executed from RAM.

The DM644x devices are based on TI's TMS320C64x+™ DSP core and an ARM926 processor. Because of its on-chip NAND Flash controller and the ability to boot using it, many hardware designers decided to have a system only with a NAND FLASH chip, without a NOR- FLASH.

This application note will explain how to program a complete Linux system (UBL, U-BOOT, Linux kernel, root file system) via the JTAG interface of the ARM926 CPU using the PEEDI JTAG Emulator and Flash programmer.

It will be explained also how to prepare PEEDI for an automatic programming.

## 2. ARM ROM Boot Modes

(Quote from TI's SPRUE14A-March 2007)

The DM644x DMSoC ARM ROM boot loader (RBL) executes when the BOOTSEL[1:0] pins indicate a condition other than the normal ARM EMIF boot (BTSEL[1:0] ≠ 01). In this case, control is passed to the ROM boot loader (RBL). If the value in BTSEL[1:0] from the BOOTCFG register is 00b, the NAND mode executes. The NAND boot mode assumes the NAND is located on the EM\_CS2 interface, whose bus width is controlled by the external EM\_WIDTH pin at reset. The RBL uses the state of the EM\_WIDTH pin from the BOOTCFG register to determine the access size to be used when reading data from the NAND.

First, the device ID of the NAND device is read from the device. Any needed access information (such as the block and page sizes, etc.) are obtained from the device information table in the RBL. Then, the RBL searches for the UBL descriptor in page 0 of the block after CIS/IDI block (block 1).

If a valid UBL is not found here, as determined by reading a proper UBL signature, the next block is searched. Searching continues for up to 5 blocks. This provision for additional searching is made in case the first few consecutive blocks have been marked as bad (that is, they have errors). Searching 5 blocks is sufficient to handle the errors found in virtually all NAND devices. If no valid UBL signature is found in the search, the RBL reverts to the UART boot mode.

If a valid UBL is found, the UBL descriptor is read and processed. The descriptor gives the information required for loading and control transfer to the UBL. The UBL is then read and processed. The RBL may enable any combination of faster EMIF and I-Cache operations based on information in the UBL descriptor first. Additionally, the descriptor provides information on whether or not DMA should be used during UBL copying. Once the user-specified start-up conditions are set, the RBL copies the UBL into ARM internal RAM, starting at address 0000:0020h.

**Note:** The actual copying of the UBL is performed on the lower 14 KB of the TCM data area: 8020 to B81Fh. The first 32-bytes of the ARM internal RAM (AIM) are the ARM's system interrupt vector table (IVT) (8 vectors, 4-bytes each). The UBL copy starts after the 32-byte IVT.

---

The NAND RBL attempts to verify a correct read by checking the ECC values when reading the UBL in the ARM IRAM. If a read error occurs, the UBL copy immediately halts for that instance, but the RBL continues to search the block following that in which the magic number was found for another instance of a magic number. When a magic number is found, the process repeats. Using this retry process, the magic number and UBL can be duplicated up to 5 times, giving significant redundancy and error resilience to NAND read errors.

The NAND user boot loader descriptor format is described in the following table:

**Table 1:**

<b>Offset</b>	<b>32-Bits</b>	<b>Description</b>
0	0xA1ACEDxx	Magic number
4	Entry Point Address of UBL	Entry point address for the user boot-loader (absolute address)
8	Number of pages in UBL	Number of pages (size of user boot-loader in number of pages)
12	Starting Block # of UBL	Block number where user boot-loader is present
16	Starting Page # of UBL	Page number where user boot-loader is present
20	Load Address	Application load address, usually 0x20

---

## 3. PEEDI configuration

PEEDI is able to program all NAND chips with 8 and 16 bits bus and up to 8Gbits of size. The first thing to do is to configure PEEDI to work with the target. This includes making the target initialization section of the PEEDI configuration file and setting the FLASH sections.

See davinci.cfg from our example configuration:

[http://download.ronetix.at/peedi/peedi\\_config\\_files.zip](http://download.ronetix.at/peedi/peedi_config_files.zip)

### 3.1 The INIT section

The INIT section of the configuration file must include the initialization for the PLL, DDR and chip selects, because the FLASH Programmer doesn't make any initialization.

Here is an example of an INIT section for a custom board with TMS320DM6446:

```
[INIT_DAVINCI]

; mask all IRQs by setting all bits in the EINT default
mem write 0x01c48018 0x00000000
mem write 0x01c4801c 0x00000000

; Put the GEM in reset
mem and 0x01c41a9c      0xffffffff

; Enable the Power Domain Transition Command
mem or 0x01c41120 0x00000002
wait 1

; Enable L1 & L2 Memories in Fast mode
mem write 0x01c4004c 0x00000001
mem write 0x01c42010 0x00444400

; Select the Clock Mode Depending on the Value written in the Boot Table by the
run script
mem and 0x01c40d00 0xffffffff      ; PLL2_CTL &= PLL_CLKSRC_MASK
mem and 0x01c40d00 0xfffffddf      ; Select the PLEN source
mem and 0x01c40d00 0xfffffefe      ; Bypass the PLL
wait 1

mem and 0x01c40d00 0xffffffff7     ; Reset the PLL
mem and 0x01c40d00 0xfffffefd      ; Power up the PLL
mem and 0x01c40d00 0xfffffefe      ; Enable the PLL from Disable

mem write 0x01c40d10 0x00000013    ; PLL2_PLLM = 0x13
wait 1

mem or      0x01c40d1c 0x00000001    ; PLL2_DIV1
wait 1

mem write 0x01c40d18 0x00000006    ; PLL2_DIV2
wait 1

mem or      0x01c40d1c 0x00080000    ; PLL2_DIV1 divider enable
wait 1

mem or      0x01c40d38 0x00000001    ; PLL2_PLLCMD, GOSET=1
wait 1

mem or      0x01c40d18 0x00000001    ; PLL2_DIV2 divider enable
```

```

mem or      0x01c40d38 0x00000001      ; PLL2_PLLCMD, GOSET=1
wait 1

mem or      0x01c40d00 0x00000008      ; Bring PLL out of Reset
wait 1

mem or      0x01c40d00 0x00000001      ; Enable the PLL
wait 1

; Shut down the DDR2 LPSC Module
mem and 0x01c41a34 0xffffffffe0      ; MDCTL_DDR2_0
mem or 0x01c41a34 0x00000003      ; MDCTL_DDR2_0

; Enable the Power Domain Transition Command
mem or      0x01c41120 0x000000001    ; PTCMD_0
wait 1

; Program DDR2 MMRs for 135MHz Setting
;-----

; Program PHY Control Register
mem write 0x200000e4 0x14001905      ; DDRCTL

; Program SDRAM Bank Config Register
mem write 0x20000008 0x00008622      ; SDCFG

; Program SDRAM TIM-0 Config Register
mem write 0x20000010 0x229229c9      ; SDTIM0

; Program SDRAM TIM-1 Config Register
mem write 0x20000014 0x0012c722      ; SDTIM1

; Program the SDRAM Bang Config Control Register
mem write 0x20000008 0x00000622      ; SDCFG

; Program SDRAM TIM-1 Config Register
mem write 0x2000000c 0x0000041d      ; SDREF

; dummy DDR2 write/read
mem write 0x80000000 0xa55aa55a
mem read 0x80000000

; Shut down the DDR2 LPSC Module
mem and 0x01c41a34 0xffffffffe0      ; MDCTL_DDR2_0
mem or 0x01c41a34 0x00000001      ; MDCTL_DDR2_0

; Enable the Power Domain Transition Command
mem or      0x01c41120 0x000000001    ; PTCMD_0
wait 1

; Turn DDR2 Controller Clocks On

; Enable the DDR2 LPSC Module
mem or 0x01c41a34 0x00000003      ; MDCTL_DDR2_0

; Enable the Power Domain Transition Command
mem or      0x01c41120 0x000000001    ; PTCMD_0
wait 1

mem write 0x80010000 0x00000001      ; CFGTEST

```

---

## 3.2 The FLASH section

A FLASH section contains information about:

1. The addresses of the DATA, ALE and CLE commands
2. Default file image to be programmed, if a short program command is used. PEEDI supports various file sources:
  - TFTP server
  - FTP server
  - HTTP server
  - MMC/SD card
3. The kind of OOB info to be used. PEEDI supports several methods to calculate the ECC:

**Table2:**

Type	Erased marker	DaVinci usage	Description
JFFS2	Yes	kernel	The standard ECC used in the Linux JFFS2 driver
JFFS2_NO_EM	No		The standard ECC used in the Linux JFFS2 driver
RAW	No		Data and spare bytes will be loaded from the image file
YAFFS	No		Like RAW, but bad blocks are skipped
FF	No		Only data bytes will be read from the image file, spare bytes will be set to 0xFF
DAVINCI_ECC	No	UBL, U-BOOT, U-BOOT environment	DaVinci Hardware ECC, 4 bytes per 512 bytes data
DAVINCI_ECC_HW6_512	Yes	root file system	DaVinci Hardware ECC, 6 bytes per 512 bytes data
DAVINCI_ECC_HW6_512_2610	Yes	root file system	DaVinci Hardware ECC, 6 bytes per 512 bytes data with workaround for a JFFS2 bug in Linux kernel 2.6.10

4. Descriptor magic – the first 32-bit value in the UBL descriptor (*see table 1*)

PEEDI is capable of generating automatically the UBL descriptor. If the parameter DAVINCI\_UBL\_DESCRIPTOR\_MAGIC is set to non-zero value then programming of the file image is relocated with one NAND Flash page (512 or 2048 bytes). The skipped page is used for the UBL descriptor and it is filled by PEEDI.

If the parameter DAVINCI\_UBL\_DESCRIPTOR\_MAGIC is missing or set to zero, then no UBL descriptor will be programmed and the file image will not be relocated.

5. Descriptor entry point – will be programmed at offset 0x4 in the UBL descriptor (*see table 1*)
6. UBL maximal file size – used from PEEDI to print a warning if the programmed file size exceeds this limit.

---

For easy programming 5 flash profiles are defined:

```
CORE0_FLASH0 = NAND_UBL
CORE0_FLASH1 = NAND_UBOOT
CORE0_FLASH2 = NAND_UBOOT_ENV
CORE0_FLASH3 = NAND_KERNEL
CORE0_FLASH4 = NAND_ROOTFS
```

The **first flash profile** (CORE0\_FLASH0) will program the UBL (user boot loader). The UBL must contain a descriptor and the DAVINCI\_ECC should be used.

The **second flash profile** (CORE0\_FLASH1) will program the U-BOOT. Depends on the UBL the U-BOOT may contain or may not contain a descriptor, the DAVINCI\_ECC should be used.

The **third flash profile** (CORE0\_FLASH2) will program the U-BOOT environment, the DAVINCI\_ECC should be used.

The **fourth flash profile** (CORE0\_FLASH3) will the Linux kernel, the standard software JFFS2 ECC should be used.

The **fifth flash profile** (CORE0\_FLASH4) will program the JFFS2 root file system, the DAVINCI\_ECC\_HW6\_512 or DAVINCI\_ECC\_HW6\_512\_2610 should be used.

If DAVINCI\_ECC\_HW6\_512 is used, then PEEDI will calculate 6 bytes ECC per 512 bytes data. Actually only 3 bytes contain ECC info, the other 3 bytes are not used.

DAVINCI\_ECC\_HW6\_512\_2610 is the same as DAVINCI\_ECC\_HW6\_512, but implements a workaround because of a bug in the JFFS2 driver in Linux 2.6.10. The workaround is that PEEDI doesn't write the ECC for an empty page (filled only with 0xFF).

*Flash Profile 0:*

```
;
; UBL should be programmed with hardware ECC without clean markers
;
[NAND_UBL]
CHIP      = NAND_FLASH
DATA_BASE = 0x02000000      ; data
CMD_BASE  = 0x02000010      ; commands (CLE)
ADDR_BASE = 0x0200000A      ; addresses (ALE)
FILE = "tftp://192.168.3.1/ubl.bin", BIN, 2048*64 ; page 64

; list with bad blocks to be marked as bad
;=====
;BAD_BLOCKS=1633,1881
;ERASE_BAD_BLOCKS = YES
OOB_INFO = DAVINCI_ECC
DAVINCI_UBL_DESCRIPTOR_MAGIC = 0xA1ACED00
DAVINCI_UBL_DESCRIPTOR_ENTRY_POINT = 0x20
DAVINCI_UBL_MAX_IMAGE_SIZE = 14*1024
```

---

Flash Profile 1:

```
;
; U-BOOT at page 384
;
;
[NAND_UBOOT]
CHIP      = NAND_FLASH
DATA_BASE = 0x02000000
CMD_BASE  = 0x02000010
ADDR_BASE = 0x0200000A
FILE = "tftp://192.168.3.1/uboot.bin", BIN, 2048*384 ; page 384

; list with bad blocks to be marked as bad
;=====
;BAD_BLOCKS=10,1633,1881
ERASE_BAD_BLOCKS = NO
OOB_INFO = DAVINCI_ECC
DAVINCI_UBL_DESCRIPTOR_MAGIC      = 0xA1ACED11
DAVINCI_UBL_DESCRIPTOR_ENTRY_POINT = 0x81080000
DAVINCI_UBL_DESCRIPTOR_LOAD_ADDR  = 0x81080000
DAVINCI_UBL_MAX_IMAGE_SIZE        = 512*1024
```

Flash Profile 2:

```
;
; U-BOOT environment at page 256
;
;
[NAND_UBOOT_ENV]
CHIP      = NAND_FLASH
DATA_BASE = 0x02000000
CMD_BASE  = 0x02000010
ADDR_BASE = 0x0200000A
FILE = "tftp://192.168.3.1/uboot_env.bin", BIN, 2048*256
ERASE_BAD_BLOCKS = NO
OOB_INFO = DAVINCI_ECC
```

Flash Profile 3:

```
;
; Linux kernel at page 512
;
;
[NAND_KERNEL]
CHIP      = NAND_FLASH
DATA_BASE = 0x02000000
CMD_BASE  = 0x02000010
ADDR_BASE = 0x0200000A
FILE = "tftp://192.168.3.1/uimage.bin", BIN, 2048*512
AUTO_ERASE = YES
ERASE_BAD_BLOCKS = NO
OOB_INFO = JFFS2
```

Flash Profile 4:

```
;
; JFFS2 root file system at page 44608
;
;
[NAND_ROOTFS]
CHIP      = NAND_FLASH
DATA_BASE = 0x02000000
CMD_BASE  = 0x02000010
ADDR_BASE = 0x0200000A
FILE = "ftp://user:password@192.168.3.1/rootfs.jffs2", BIN, 2048*44608
```

```
;OOB_INFO = DAVINCI_ECC_HW6_512
; Linux kernel 2.6.10 has a bug - to workaround it use:
OOB_INFO = DAVINCI_ECC_HW6_512_2610
AUTO_ERASE = YES
```

***WARNING:***



***The PEEDI TFTP client uses fixed 512 bytes transfer block size, which limits the size of the image file to 32MB. If your file is bigger, use HTTP or FTP file server or use MMC/SD card to store the file and put it on PEEDI.***

### 3.3 The ACTIONS section

Declares what scripts can be executed using front panel buttons, each declaration must be on a new line. The declaration consists of a number associated with the specified script name. A section with the same name must exist somewhere in the target configuration file. If AUTORUN=N parameter is specified, where N is number of a script, the given script will be executed every time a target is connected to PEEDI.

Example:

```
[ACTIONS]                ; user defined scripts
;AUTORUN                  = 2          ; executed on every target connect
1 = erase
2 = prog

;
; erase the NAND Flash
;
[erase]                   ; erase flash
flash set 0
flash erase

;
; program UBL, U-BOOT, Kernel and ROOTFS at once
;
[prog]
flash set 0                ; select flash profile 0
flash erase                 ; erase the flash
flash prog                 ; program UBL

flash set 1                ; select flash profile 1
flash program              ; program U-BOOT

flash set 2                ; select flash profile 2
flash program              ; program U-BOOT environment

flash set 3                ; select flash profile 3
flash program              ; program Linux kernel

flash set 4                ; select flash profile 4
flash program              ; program JFFS2 root file system
```

---

## 3.4 Bad block management

The NAND Flash devices may have blocks that are invalid when they are shipped.

An invalid block is one that contains one or more bad bits. Additional bad blocks may develop with use.

The factory identifies invalid blocks before shipping by programming data other than FFh (x8) or FFFFh (x16) into the first spare location of the first or second page of each bad block.

PEEDI automatically detects the bad blocks and reports them:

```
peedi> flash info
FLASH configuration for core #0:
Nand Flash: ID = 0x75, 8-bit, 32 MB
  page size           = 512 + 16 bytes
  pages per block     = 32
  number of blocks    = 2048
  number of bad blocks = 2 (use "flash query" to get the list)
  erase bad blocks    = true

peedi> flash query
Total blocks = 2048
Bad blocks = 2
Bad blocks list:
1289
1291

peedi>
```

Once detected, the bad blocks are protected against erasing and programming.

On demand, PEEDI can be forced to try to erase the existing bad blocks.

It is also possible to force blocks as bad.

To erase all blocks including the bad blocks the following parameter must be changed:

```
ERASE_BAD_BLOCKS = YES
```

After PEEDI restart, the command "flash erase" will erase all blocks.



**WARNING:**

***If you erase blocks factory marked as bad, there is now way to detect which were the bad blocks.***

Make sure you have saved the output of the "flash query" command so you can mark again the bad blocks as bad.

To force marking of blocks 4, 27 and 1002 as bad:

```
BAD_BLOCKS = 4, 27, 1002
```

After PEEDI restart the "flash info" command will mark the given blocks as bad.

Once marked as bad, the blocks are not marked anymore.

---

## 4. Programming

Now everything is configured and you should be ready to program your NAND chip.

The flash programming using the flash profiles is very easy:

```
peedi> flash set          ; show all flash profiles
*Flash #0 -> NAND_UBL (current)
Flash #1 -> NAND_UBOOT
Flash #2 -> NAND_UBOOT_ENV
Flash #3 -> NAND_KERNEL
Flash #4 -> NAND_ROOTFS

peedi> flash program      ; program UBL or just
peedi> f p

If you want to program a second UBL:

peedi> flash program tftp://192.168.3.1/ubl.bin bin 2048*64*2

peedi> flash set 1       ; select flash profile 1
peedi> flash program     ; program U-BOOT

peedi> flash set 2       ; select flash profile 2
peedi> flash program     ; program U-BOOT environment

peedi> flash set 3       ; select flash profile 3
peedi> flash program     ; program Linux kernel

peedi> flash set 4       ; select flash profile 4
peedi> flash program     ; program JFFS2 root file system
```

The programming of the complete Linux system at once:

```
peedi> run $prog          ; run the script "prog" defined in [ACTIONS]
```