

Technical Note

Boot-from-NAND Using Micron® MT29F1G08ABB NAND Flash with the Texas Instruments™ (TI) OMAP2420 Processor

Overview

NAND Flash memory devices are designed for applications requiring nonvolatile, high-density, solid-state storage media. Historically, NAND Flash has been used extensively in applications such as mass storage cards and digital cameras. With its low cost and high performance, NAND Flash is beginning to make its way into more complex embedded systems where NOR Flash has dominated in the past—for example, in mobile phones.

In complex embedded systems, one of the challenges associated with a change from NOR Flash to NAND Flash has been the boot process. NOR Flash has been a popular choice for these systems because it contains a traditional memory interface (including both address and data buses), making it capable of execute-in-place (XIP) operation. XIP memory enables the system CPU to execute code directly from the memory device because the boot code is stored on and executed from a single device.

NAND Flash is a page-oriented memory device that does not inherently support XIP, at least not in the same manner as a typical XIP memory device. Operating system and boot code can be stored in NAND Flash memory, but the code must be copied (or shadowed) to DRAM before it can be executed. This requires system designers to modify the boot process for their systems when using NAND Flash. The payoff for this modification is that the system benefits from the lower cost of NAND Flash as the storage solution and the higher performance of DRAM as the XIP memory.

Scope

This technical note discusses a boot-from-NAND solution for applications using the Texas Instruments™ (TI) OMAP2420 processor and the Micron® MT29F1G08ABB NAND Flash device. The technical note provides a four-stage boot sequence. Stages 1 and 2 are independent of the operating system (OS); however, they are highly dependent on system hardware. Thus, the primary focus of this technical note is on stage 1 and stage 2 boot processes.

Additional system details:

- TI OMAP2420 H4 with processor daughter card “Menelaus ES 2.0”;
S/N: 750-0006, Rev C.
- The boot-from-NAND concepts discussed are OS independent; however, the Linux OS is used as an example in some explanations.

Note that secure booting via the OMAP™ high-security (HS) device is not in the scope of this document. However, NAND Flash booting for HS and GP differs only in the generation of the X-Loader. Thus, the solution discussed here is generally applicable to both the HS and GP processors.

Terms used in the technical note are provided in Table 1.

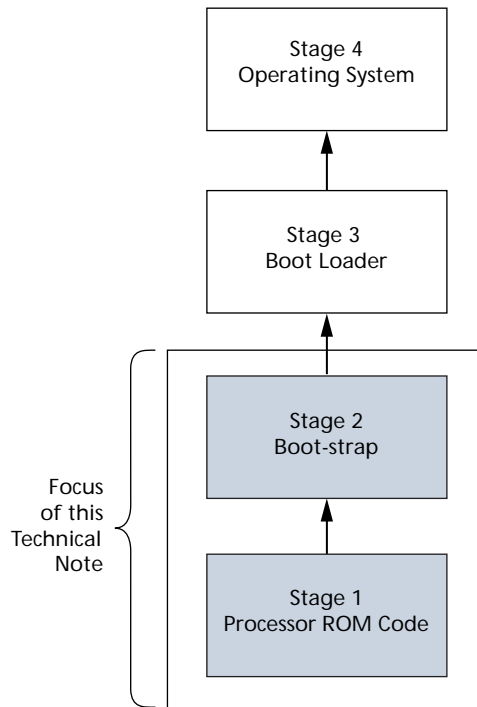
Table 1: Glossary of Terms and Abbreviations

Abbreviation	Description
ECC	Error correction code
GP	General purpose OMAP device
GPMC	General purpose memory controller
HS	High-security OMAP device
ICE	TRACE32® in-circuit emulator by Lauterbach, Inc.
JTAG	JTAG standard (from the Joint Test Action Group)
OMAP2420	TI processor
OS	Operating system
OST tools	TI OMAP software tools
TI	Texas Instruments™
U-boot	Linux OS boot loader
XIP	Execute in place
X-Loader	Pre-OS bootstrap code

Boot-from-NAND Overview

Figure 1 provides an overview of the boot sequence for an OMAP2420-based system using boot-from-NAND.

Figure 1: Boot-from-NAND Stages



Boot Stages

Stage 1: Processor ROM Code

During power-on, or after a RESET operation, the OMAP2420 processor runs its internal ROM code. Note that only NAND Flash devices supported by the OMAP2420 processor ROM can be used for the boot process (see Table 2 on page 4).

Stage 2: Bootstrap

X-Loader is an example of stage 2 bootstrap code. The X-Loader code is stored in the NAND Flash, and the ROM code copies it to the OMAP processor SRAM for execution.

Stage 3: Boot Loader

Stage 3 is the boot loader, which is used to copy the operating system code from the NAND Flash to the DRAM; in this case U-Boot is the boot loader code. The U-Boot code is stored in NAND Flash, and the stage 2 code copies it to the DRAM for execution.

Stage 4: Operating System

The OS code, such as the Linux kernel, is stored in the NAND Flash, and the stage 3 code copies it to the DRAM, where it is executed. The boot process is complete after this stage as the OS takes control of the system.

Table 2: Micron NAND Flash Devices Supported by the OMAP2420 Processor

Micron Part Number	Density	Device ID	Bus Width	Page Size (bytes)
MT29F1G08ABB	1Gb	A1h	x8	2,112
MT29F1G16ABB	1Gb	B1h	x16	2,112
MT29F2G08AAD	2Gb	AAh	x8	2,112
MT29F2G16AAD	2Gb	BAh	x16	2,112

If the device ID of the NAND Flash is not supported by the ROM, an attempt is made to determine the configuration of the NAND Flash device by performing a READ ID2 operation. Contact your Micron representative for READ ID2 operation details. If both the READ ID and READ ID2 operations fail, the ROM code performs a software reset on the system.

Stage 1: Processor ROM Code

Stage 1 is the execution of the OMAP2420 processor ROM code. This ROM code cannot be modified by the system designer. Only NAND Flash devices supported by the ROM code can be used for boot-from-NAND with the OMAP2420 device. If the ROM code does not support a particular NAND Flash device, contact a Texas Instruments representative to determine if additional ROM code is available that will support the Micron NAND Flash device.

After a power-on-reset is initiated, the ROM code reads the SYS.BOOT register to determine the memory interface configuration and programs the general-purpose memory controller (GPMC) accordingly. Then the ROM code issues a RESET (FFh) command (see Figure 2) to the NAND Flash device, followed by a READ ID (90h) command (see Figure 3 on page 5). The READ ID operation enables the OMAP2420 processor to determine how the NAND Flash device is configured and whether this device is supported by the ROM code.

Table 3 shows the required SYS.BOOT register settings to support boot-from-NAND. Micron NAND Flash devices are available in both x8 and x16 configurations; the Micron NAND Flash device referenced in this technical note, MT29F1G08ABB, has a x8 interface.

Table 3: SYS.BOOT Register

SYS.BOOT[3:0]				Device Type
3	2	1	0	
1	1	0	0	x8 NAND Flash
1	1	0	1	x16 NAND Flash

Figure 2: NAND Flash Reset (FFh) Command

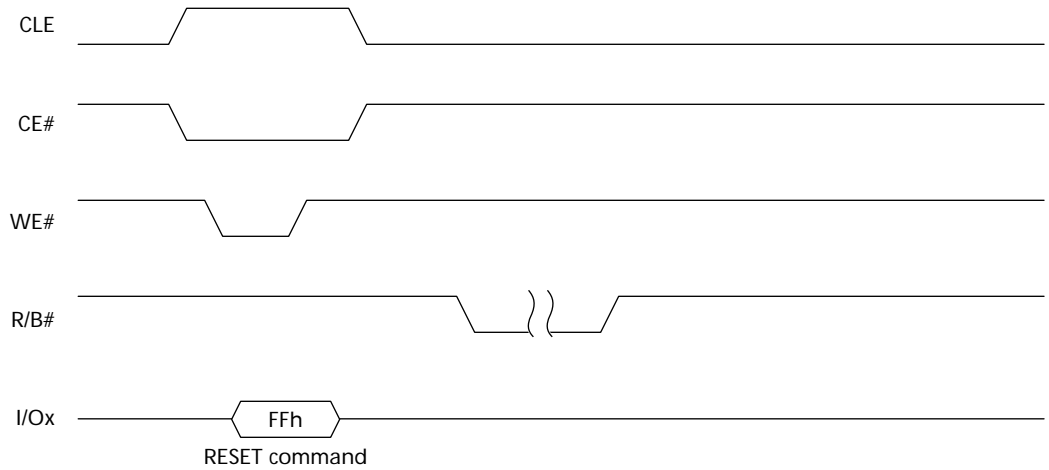
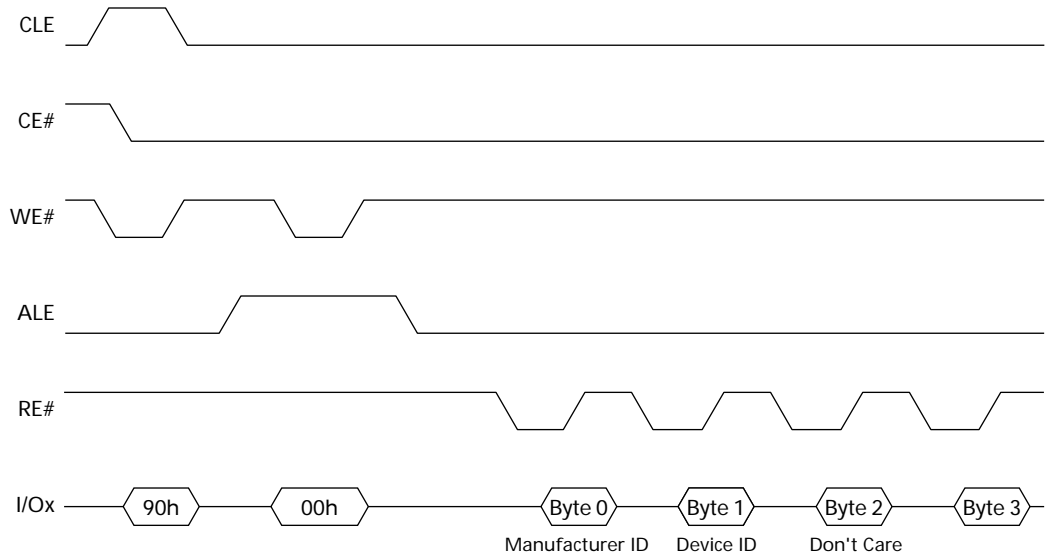


Figure 3: NAND Flash READ ID Command



Bad Blocks

The ROM code expects the X-Loader to be in block 0, 1, 2, or 3 of the NAND Flash device and can use any of these blocks in the boot process. After the NAND Flash device configuration has been determined, the OMAP2420 processor ROM code performs a bad-block check on these blocks.

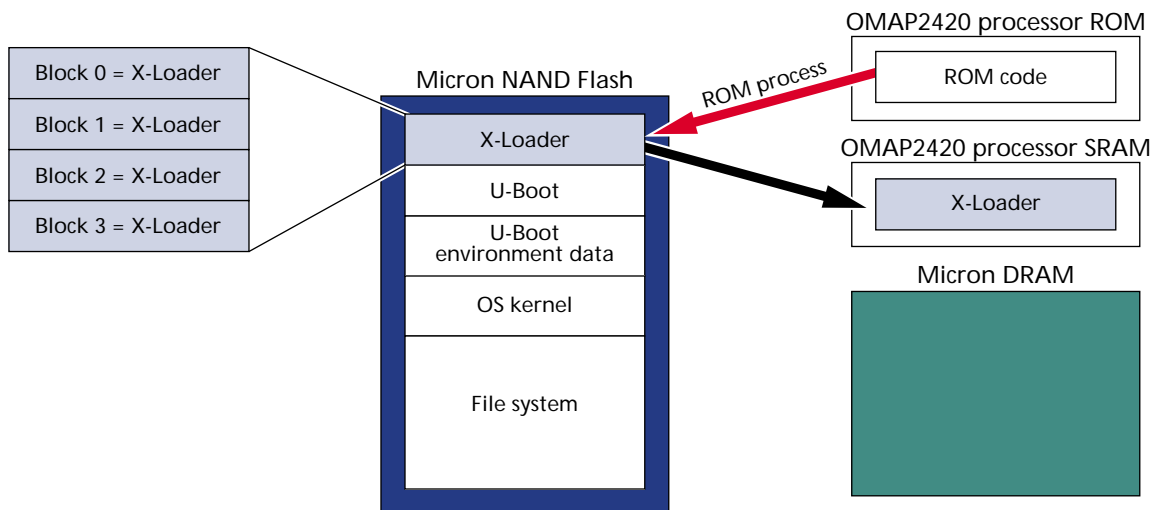
Block 0 of the MT29F1G08ABB device is guaranteed to be good for 1,000 PROGRAM/ERASE cycles, so in the majority of cases, the ROM code will not have to look beyond block 0 for a bootable image.

Code Shadowing to the OMAP Processor SRAM

After the NAND Flash device configuration has been verified and the bad blocks have been checked, the process of copying (shadowing) the X-Loader from the NAND Flash device to the internal SRAM of the OMAP2420 processor begins.

First, the ROM code reads bytes 1 through 4 of the X-Loader to determine the size of the file; then it reads bytes 5 through 8 of the X-Loader, which contain the destination address in SRAM where the X-Loader will be shadowed (see Figure 6 on page 8). The ROM code then shadows the X-Loader from the NAND Flash device to the OMAP2420 processor SRAM (see Figure 4), and finally, the system jumps to the SRAM address where the first byte of the X-Loader is stored.

Figure 4: Shadowing X-Loader Code from NAND Flash to SRAM



Error Correction Code

The ROM code contains error correction code and checks for errors in the X-Loader. The ECC scheme is a Hamming code capable of detecting 2-bit errors and correcting one 1-bit error per 512 bytes.

- When a 1-bit error is detected in a 512-byte sector, the ROM code will use the ECC to correct the error, and the boot process will continue from that block.
- When a 2-bit error is detected in a 512-byte sector, the ROM code will skip this block and attempt to boot from the next block.
- When an error of 3 bits or more is detected, effects on the system may vary and may include hanging.

Figure 7 on page 9 depicts how the X-Loader, bad block marking, and ECC are stored in a typical page of the MT29F1G08ABB NAND Flash device programmed for boot-from-NAND in an OMAP2420-based system.

Stage 2: Bootstrap

Stage 2 of the boot process is dependent on NAND Flash in the sense that the X-Loader is stored in the NAND Flash. It is important to remember that this X-Loader is executed in the OMAP processor SRAM, so the image must fit in the available SRAM.

In a Linux-based system, the stage 2 boot consists of an X-Loader that bootstraps U-Boot. This bootstrap includes:

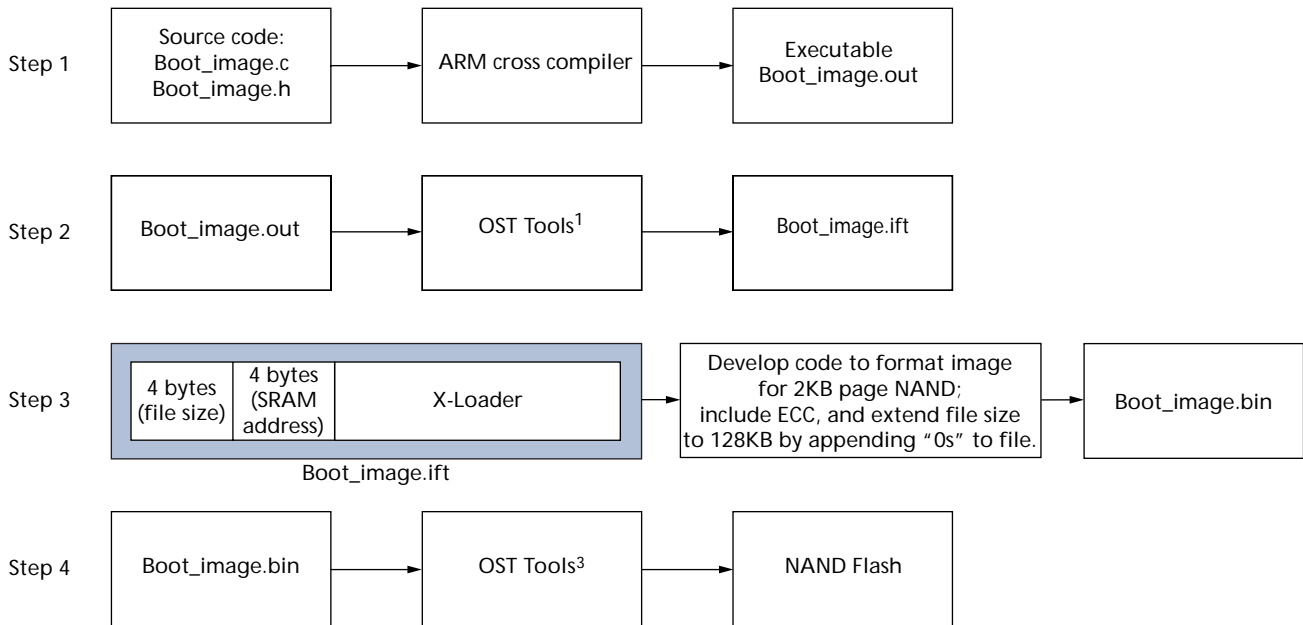
- Information for each supported CPU architecture
- A configuration file for each supported board
- NAND Flash driver code
- Serial driver code (to support debug and development efforts)

Building the X-Loader

Building the X-Loader is a critical step in developing a boot-from-NAND system. Figure 5 on page 8 shows the process for converting source code to a raw binary X-Loader that can be stored in the Micron NAND Flash device. The “Boot_image” prefix in the file names is provided only as an example; actual file names can be designated by the system designer. Figure 6 on page 8 illustrates how the Boot_image.bin code is laid out in the NAND Flash.

1. Compile the X-Loader source code into an executable format. The example in Figure 5 on page 8 shows how the source code comprises Boot_image.c and Boot_image.h; the executable is Boot_image.out.
2. Execute the OST Tools to sign the target file Boot_image.out. When this process is complete, an 8-byte header is included in the Boot_image.ift file. Bytes 1 through 4 of Boot_image.ift contain the file size. Bytes 5 through 8 contain the OMAP processor SRAM address where the X-Loader will be loaded and executed (see Figure 6 on page 8). OST Tools are available from TI. (See OST Tools documentation for additional details.)
3. Format the Boot_image.ift file resulting from step 2 for 2KB/page NAND Flash. Code must be developed for this purpose if it is not available from TI. To format the Boot_image.ift file, calculate the ECC for each 512-byte sector. Then store the result in the appropriate area of the file. See Figure 7 on page 9 for boot code, bad block marking, and ECC storage in a typical page of an MT29F1G08ABB NAND Flash device programmed for boot-from-NAND in an OMAP2420-based system.
4. Program a copy of the Boot_image.bin file to each of the first four good blocks of the NAND Flash device. The MT29F1G08ABB device identifies a good block as one that has 0xFF data at byte 0x800 in both page 0 and page 1.

Figure 5: Preparing the X-Loader



- Notes:
1. OST Tools is a TI program. Contact a TI representative to obtain a copy.
 2. Micron.exe converts ASCII to binary code. Other ASCII-to-binary programs can be used in its place.
 3. Some OST versions do not support Micron MT29F1G08ABB NAND Flash. See “Writing Binary Images to NAND Flash with Limited OST Tools Support” on page 12 for instructions regarding alternatives.

Figure 6: X-Loader Layout

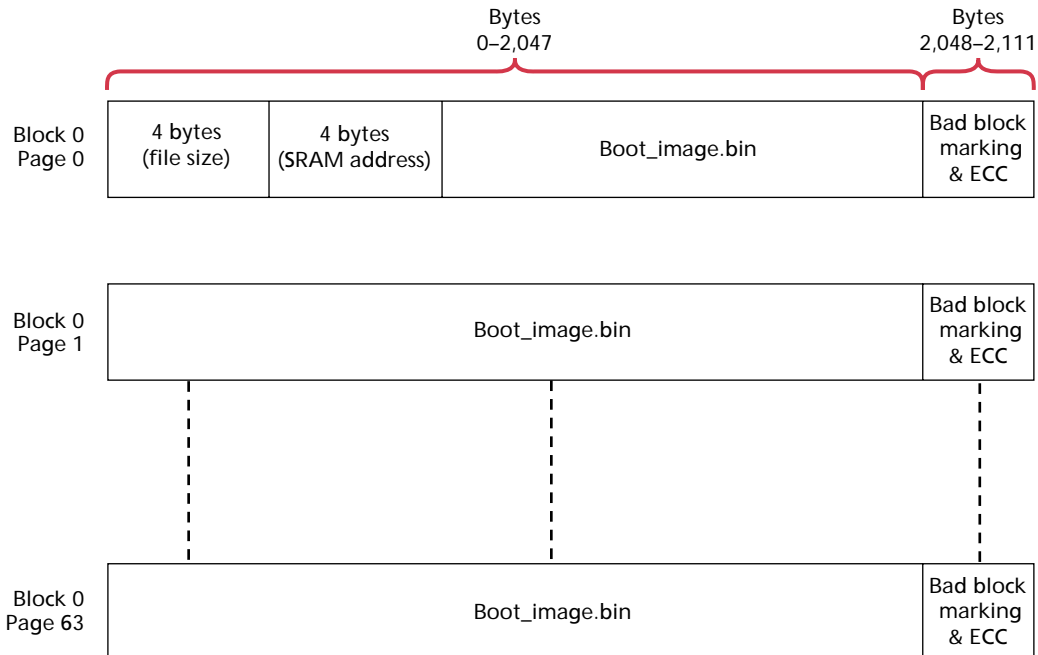
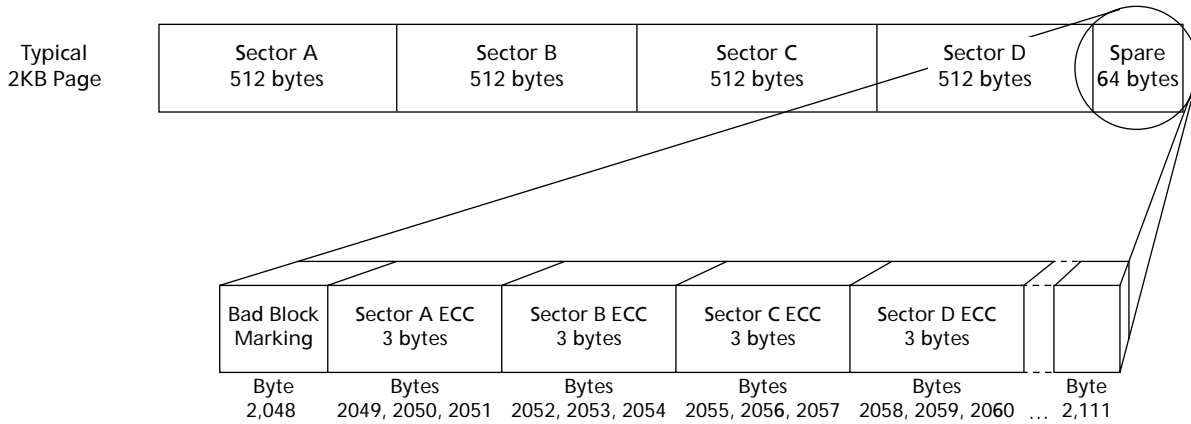


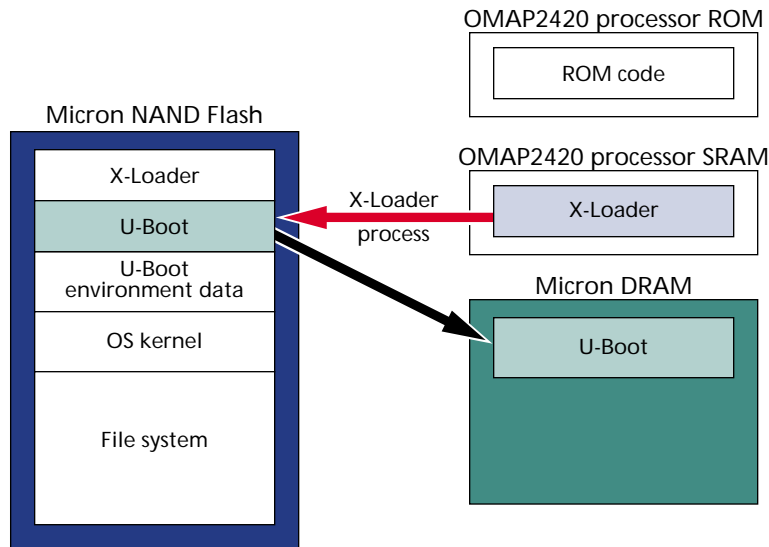
Figure 7: Page-Level Boot Code Storage



Shadowing U-Boot Code from NAND Flash to DRAM

In an OMAP2420 system, X-Loader shadows the U-Boot code from NAND Flash to DRAM (see Figure 8). Then the system jumps to the address in DRAM where the first byte of the U-Boot code resides.

Figure 8: Shadowing U-Boot Code from NAND Flash to DRAM



Stage 3: Boot Loader

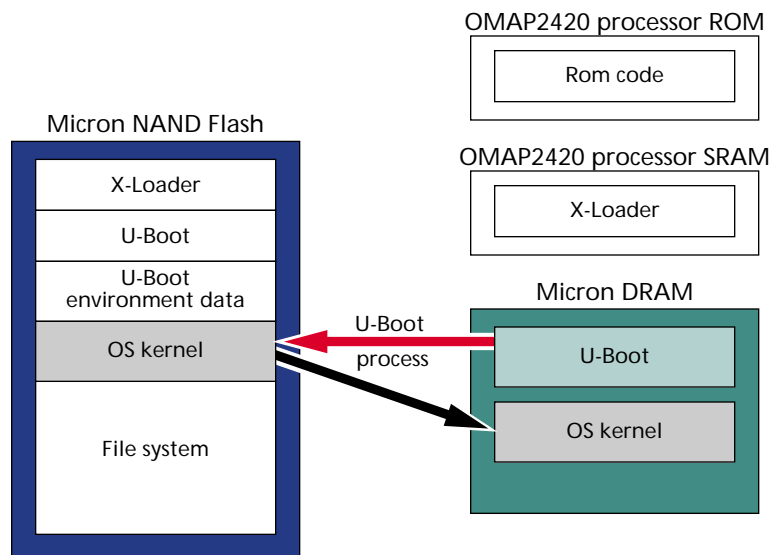
Stage 3 of the boot process is heavily dependent on the OS. In a Linux system, the stage 3 boot consists of U-Boot, the OS boot loader for Linux. U-Boot resides in the NAND Flash but is shadowed to DRAM for execution (as mentioned in the stage 2 boot description).

U-Boot Considerations

- The memory map must be configured to support boot-from-NAND.
- U-Boot must contain NAND Flash support such that it can read and write to the NAND Flash device.
- U-Boot environment data should be written such that it can be stored in a single block (128KB) of the Micron MT29F1G08ABB NAND Flash device.
- The CFG_NAND_BOOT configuration label is stored in a board configuration file and is used to differentiate NAND U-Boot from NOR U-Boot.

U-Boot shadows the OS kernel code from NAND Flash to DRAM (see Figure 9) and then jumps to the address in DRAM where the first byte of the OS code is stored.

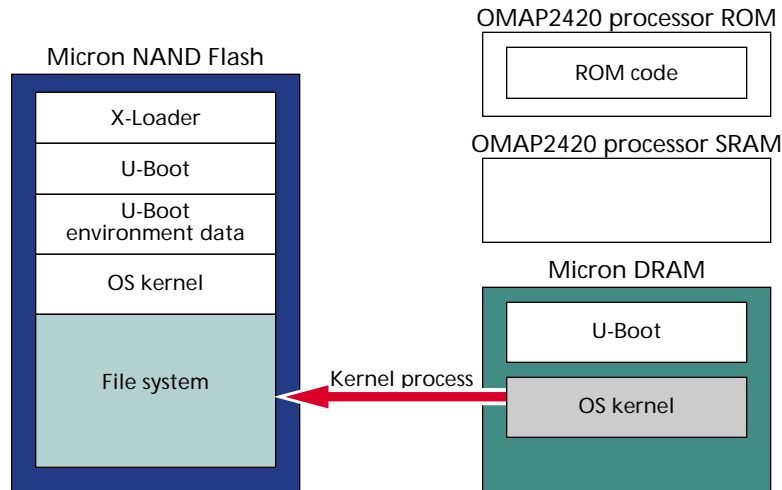
Figure 9: Shadowing OS Kernel Code from NAND Flash to DRAM



Stage 4: Operating System

The final stage of the boot process involves the initial execution of the OS. The operating system kernel is stored in NAND Flash and shadowed to DRAM for execution as described in the stage 3 boot description. When the system jumps to the beginning of the OS code (see Figures 10), the OS takes control of the system.

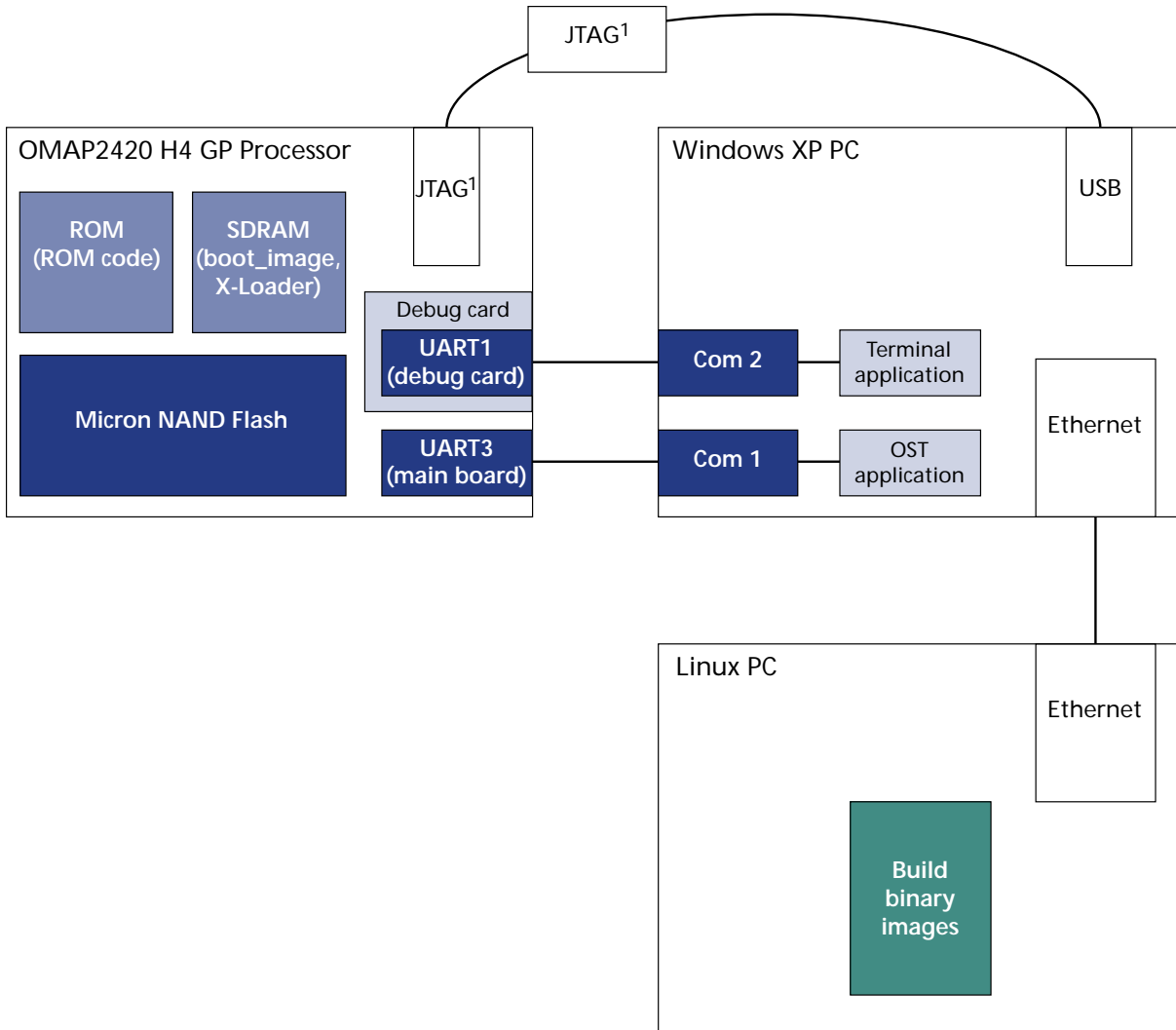
Figure 10: Kernel Process: OS Takes Control



Writing Binary Images to NAND Flash with Limited OST Tools Support

Some versions of the OST Tools do not fully support Micron NAND Flash devices. In these cases, it is necessary to develop an alternative method for loading the boot code into the NAND Flash device. A workstation similar to the one shown in Figure 11 is required. In addition, modified X-Loader and U-Boot software are required. Contact your Micron representative for the modified software.

Figure 11: Boot-from-NAND Workstation Configuration

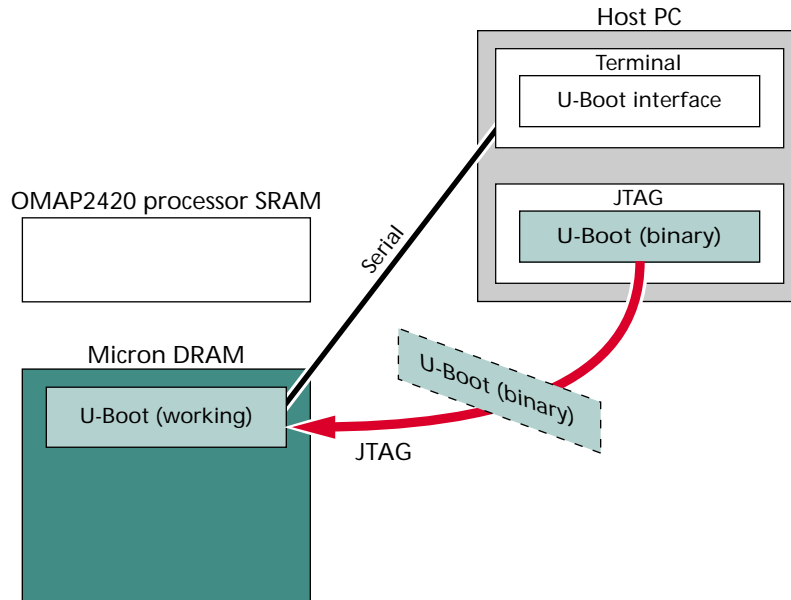


Notes: 1. TRACE32 in-circuit emulator, from Lauterbach, Inc.

Run the U-Boot Program

1. Use JTAG to load the U-Boot program into the Micron DRAM (see Figure 12). The U-Boot program can also be loaded with some versions of the OST Tools.
2. Run the U-Boot program.

Figure 12: Example of Running the U-Boot

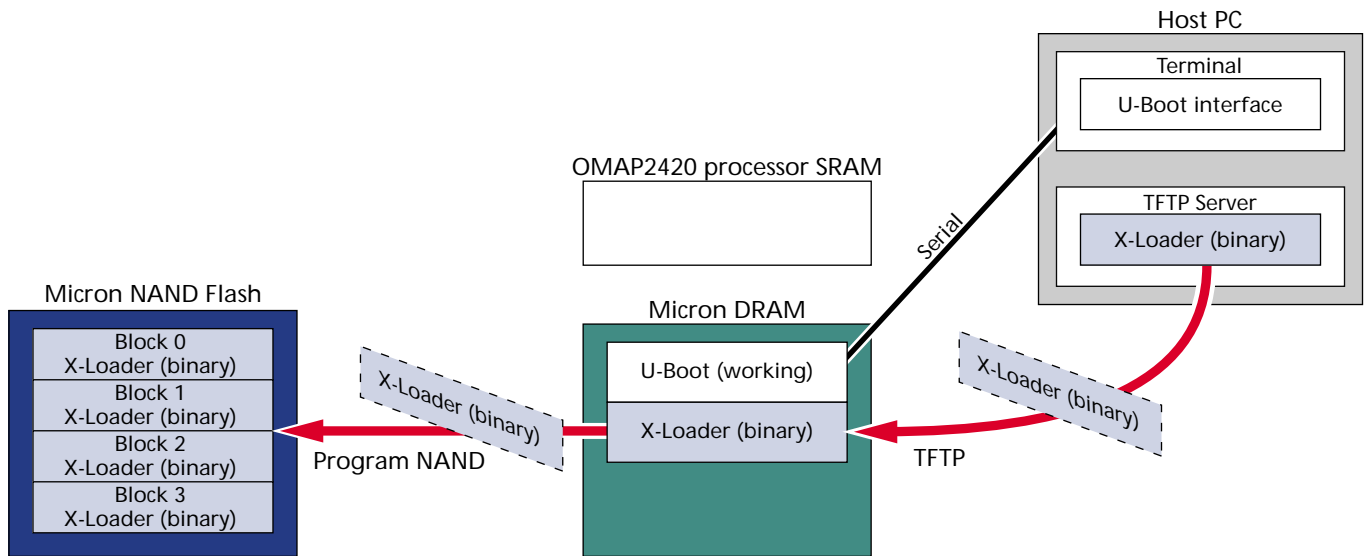


Write the X-Loader to the NAND Flash

1. Configure the U-Boot to communicate with the TFTP server in the terminal window.
2. Place the X-Loader image in the TFTP server.
3. Write a copy of the X-Loader file to the DRAM using the U-Boot program.
4. Erase the area in the NAND Flash where the X-Loader will reside.
5. Copy the X-Loader program from the DRAM to the NAND Flash.

This step is illustrated in Figure 13.

Figure 13: Example of Writing the X-Loader to the NAND Flash

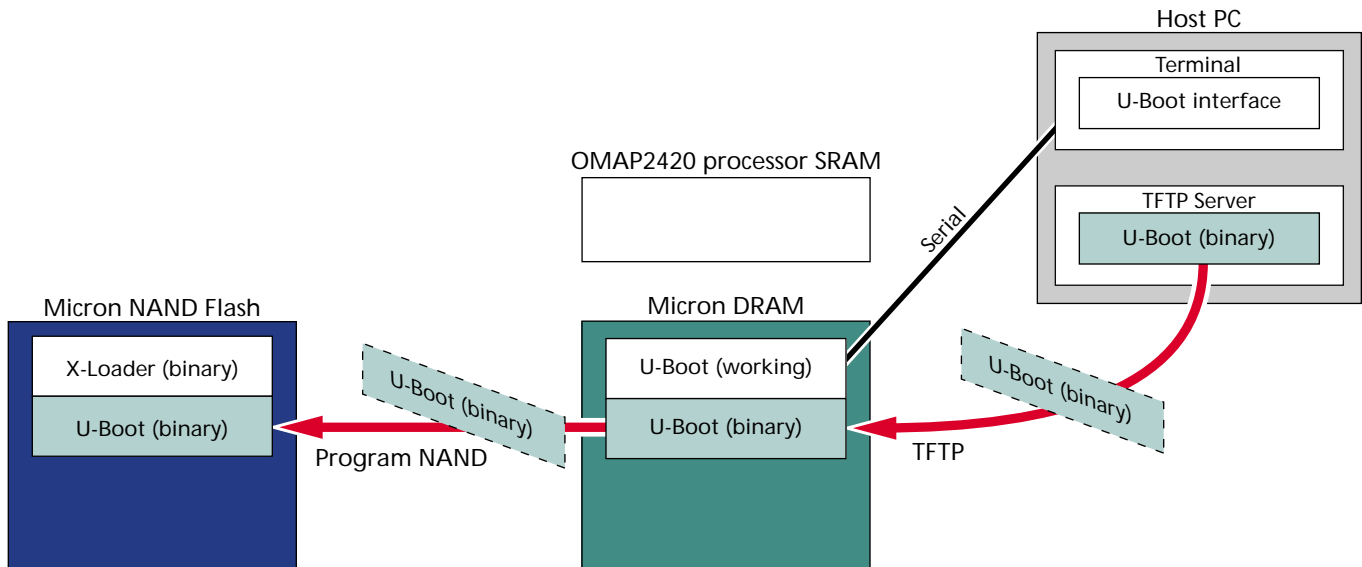


Write the U-Boot to the NAND Flash

1. Place the U-Boot file in the TFTP server.
2. Write a copy of the U-Boot file to the DRAM using the U-Boot program.
3. Erase the area in the NAND Flash where U-Boot will reside.
4. Copy the U-Boot program from the DRAM to the NAND Flash.

This step is illustrated in Figure 14.

Figure 14: Example of Writing the U-Boot to the NAND Flash

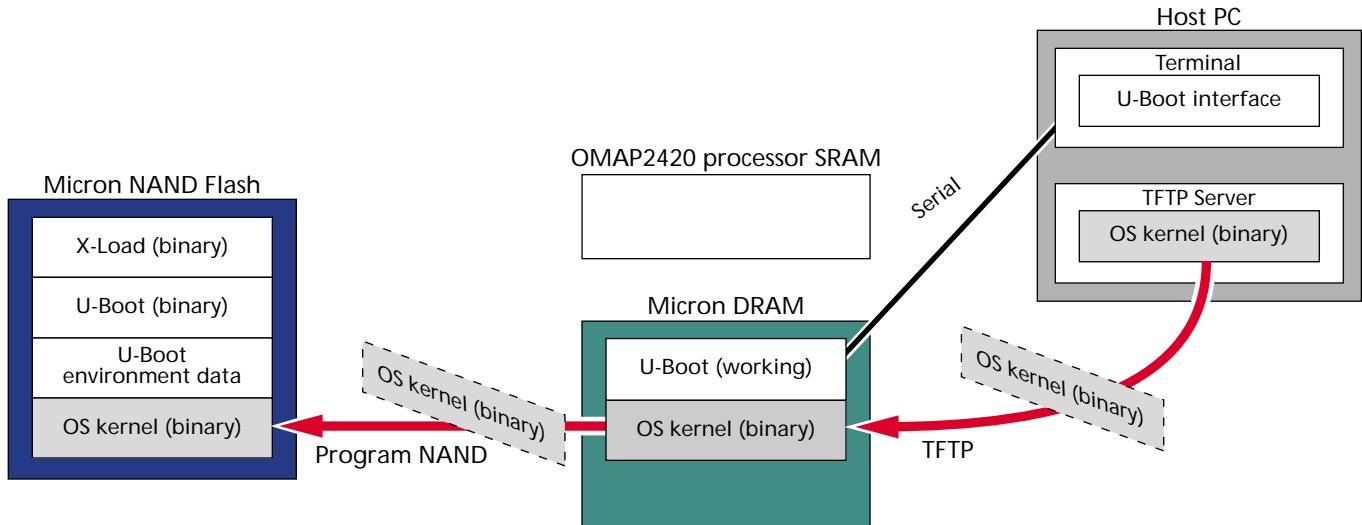


Write the OS Kernel to the NAND Flash

1. Place the OS kernel file in the TFTP server.
2. Write a copy of the OS kernel file to the DRAM using the U-Boot program.
3. Erase the area in the NAND Flash where the OS kernel will reside.
4. Use the U-Boot program to copy the OS kernel from the DRAM to the NAND Flash.

This step is illustrated in Figure 15.

Figure 15: Example of Writing the OS Kernel File to the NAND Flash

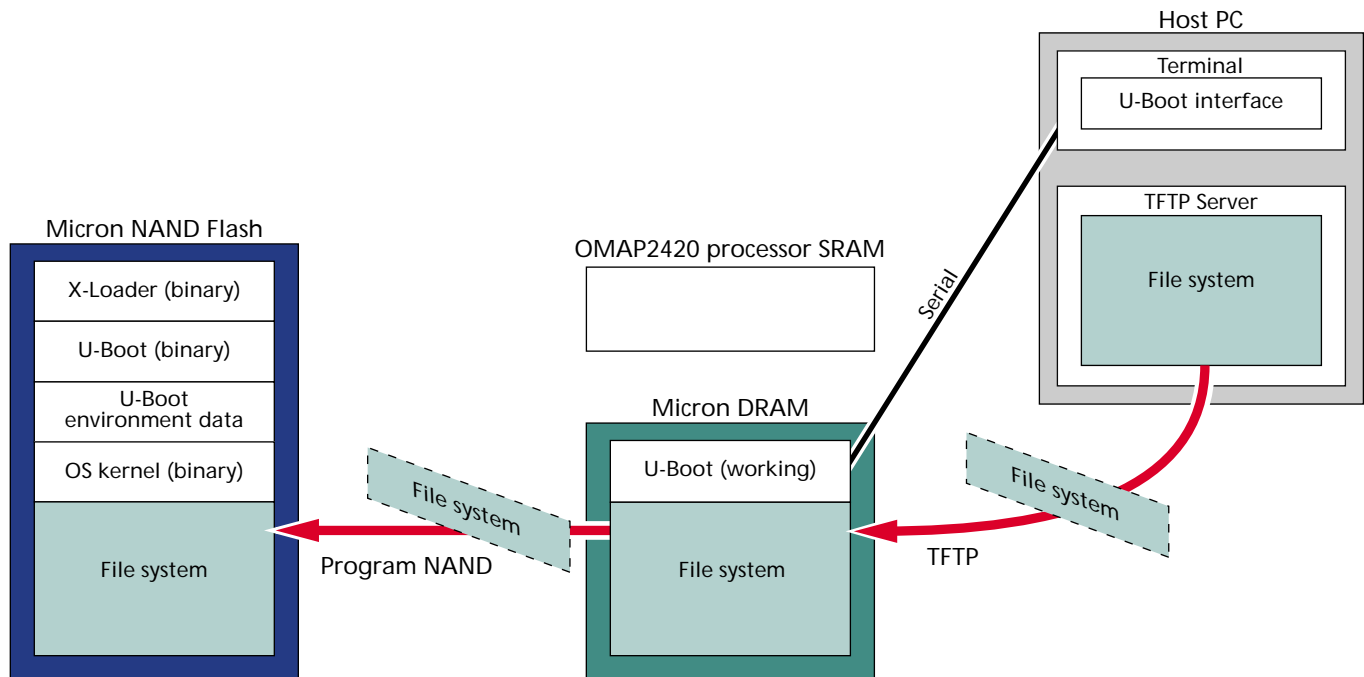


Write the File System to the NAND Flash

1. Place the root file system file in the TFTP server.
2. Write the root file system file to the DRAM using the U-Boot program.
3. Erase the area in the NAND Flash where the file system will reside.
4. Copy the file system from the DRAM to the NAND Flash.

This step is illustrated in Figure 16.

Figure 16: Example of Writing the File System to the NAND Flash



Recommendations for Maximizing Reliability of Boot Code

- When programming the X-Loader, U-Boot, OS kernel, and root file system to the NAND Flash device, program each page in its entirety with a single program operation.
- Verify that the X-Loader, U-Boot, OS kernel, and root file system were programmed correctly by performing a read-verify to compare the NAND Flash contents against the original binary image.
- Even a single bad bit in the code can cause a system failure, so error correction should be maximized in code storage areas of the NAND Flash.
- Avoid excessive reads to the area of the NAND Flash where code is stored. When repeated accesses are required, the code should be copied to the DRAM. This minimizes the probability of read-disturb errors in the NAND Flash device.

Conclusion

The OMAP2420 processor provides a solid foundation for system designers developing boot-from-NAND solutions using the Micron MT29F1G08ABB NAND Flash device. With boot-from-NAND capability structured as described in this technical note, embedded systems designers can take advantage of lower-cost NAND Flash for storage and can achieve higher performance using DRAM as the XIP memory.



8000 S. Federal Way, P.O. Box 6, Boise, ID 83707-0006, Tel: 208-368-3900

prodmktg@micron.com www.micron.com Customer Comment Line: 800-932-4992

Micron, the M logo, and the Micron logo are trademarks of Micron Technology, Inc. All other trademarks are the property of their respective owners.

Revision History

Rev. D	6/07
<ul style="list-style-type: none">• Changed MT29F1GxxABA to MT29F16xxABB throughout.• Table 2 on page 4: Changed MT29F1GxxABA to MT29F16xxABB, MT29F2GxxAAB to MT29F2GxxAAD; deleted MT29F2GxxABC; updated device IDs.• “Bad Blocks” on page 5: Revised description.• “Code Shadowing to the OMAP Processor SRAM” on page 6: Changed “SDRAM” to “SRAM.”	
Rev. C	7/06
<ul style="list-style-type: none">• Figure 1 on page 3: Updated content.• “Boot Stages” on page 3 and “Stage 1: Processor ROM Code” on page 4: Updated stage descriptions.• “Code Shadowing to the OMAP Processor SRAM” on page 6: Added Figure 4.• “Stage 2: Bootstrap” on page 7: Added Figure 8.• “Stage 3: Boot Loader” on page 10: Added Figure 9.• “Stage 4: Operating System” on page 11: Added Figure 10.• Updated all figures to correlate figure elements.• Refined descriptions throughout.	
Rev. B	4/06
<ul style="list-style-type: none">• “Building the X-Loader” on page 7: Updated step 3 and step 4 descriptions.• Figure 5 on page 8 and Figure 13 on page 14: Updated step 3 and notes.• Figure 6 on page 8: Deleted note.	
Rev. A	3/06
<ul style="list-style-type: none">• Initial release.	