Methods for Loading Custom Software on the i.MX31 and i.MX27 SOM-LV in Manufacturing

Application Note 390

Logic // Embedded Product Solutions
Published: October 2008

Abstract

This Application Note explains the different methods to load custom software on i.MX31 and i.MX27 SOM-LV.

Revision History

REV	DESCRIPTION	APPROVAL	DATE
A	Initial release	MAA	10/31/08

Table of Contents

1 Introduction

Customers planning to use the i.MX31 or i.MX27 SOM-LVs in their end-product have a few choices to get their exact software configuration on the SOM for manufacturing. This step in the product cycle is typically not thought about until the end of development, once the manufacturing team is ramping up. If this step can be considered earlier in the process, it can save time and money, as the details can be established during the design cycle.

There are two approaches for purchasing SOMs: buying a standard configuration SOM or a custom configured SOM. Buying a standard configuration SOM means the SOM will ship with LogicLoader, which is the boot code for the system and provides the essential functionality to program more software onto the board. Buying a custom configured SOM follows Logic's NPI (New Product Introduction) process. The process provides the opportunity for custom software to be loaded onto the SOM when it is manufactured, which means the SOM will ship with that custom software preloaded.

This document will detail these two approaches and describe how the software is loaded onto the SOM for each specific case.

2 Basics

The i.MX31 and i.MX27 SOMs include two components that can be programmed: on-board NAND and NOR flash chips. The standard i.MX SOM configurations ship with the LogicLoader bootloader programmed into the NOR flash and an empty NAND flash. By default, both i.MX SOMs boot from the NOR flash. LogicLoader allows for burning code to NOR and NAND flash from several sources and then automatically run that software after a power-cycle. The full functionality of LogicLoader is covered in the *LogicLoader User's Manual*, *LogicLoader User Manual Addendum*, and *LogicLoader Command Description Manual* documents available on the downloads page for each SOM.

2.1 Examples and Nomenclature within this Document

- All examples in this document require LogicLoader version 2.4.0 or later, and are written for the i.MX31.
- A difference to keep in mind between the i.MX31 and i.MX27 is the NOR flash address. The i.MX31 NOR flash is at address 0xA000_0000; the i.MX27 NOR flash is at address 0xC000_0000.
- The use of i.MX in this document refers to both the i.MX31 and i.MX27.

2.2 LogicLoader Functionality Summary

The following is a brief summary of LogicLoader functionality useful for manufacturing:

- Loading and/or running files from SD memory, CompactFlash memory, serial, and TFTP.
- Burning and creating raw NOR and NAND flash memory files.
- YAFFS – File system on flash (block management, wear-leveling, sharing files with Operating Systems, .etc.).
- Scripting – Auto-run, conditional operations (if/else), variables, etc.
- Video – Splash screens, instruction feedback with bitmaps, custom display support.
- Setting and reading GPIO for conditional operations (jumpers, configuration differences).

2.3 NOR Flash Overview

The standard configuration of NOR flash on the i.MX SOM is 2 MB and acts as the default boot medium. It is 16 bits wide, a linear memory-mapped device, and has a typical SRAM layout when read. Programming NOR flash requires a specific algorithm. Custom SOM configurations can change the amount of NOR flash to 0 or 4 MB if necessary; however, this change must be defined within the NPI process.

More information about NOR flash addressing and programming can be found in the *LogicLoader User's Manual*.

The standard i.MX SOM uses part JS28F160C3BD70 from Numonyx, a 2 MB chip (16 Mbit) that supports a minimum 100,000 erase cycles per block. A wear-leveling file system is recommended for applications that require frequent writes to NOR flash.

2.4 NAND Flash Overview

The standard configuration of NAND flash on the i.MX SOM is 64 MB and uses an 8 bit memory bus. NAND flash does not look like a typical SRAM layout when read, which complicates programming at manufacturing. Because NAND flash is made up of blocks, it requires both read and write algorithms. Custom i.MX SOMs can accommodate 0, 16, 32, 64 MB, and higher if necessary; however, a change to NAND flash configuration must be defined within the NPI process. Booting from NAND is not supported by LogicLoader.

More information about NAND flash addressing, programming, and block management can be found in the *LogicLoader User's Manual*.

Important Note: NAND flash **can and will** have bad blocks. The standard i.MX SOM uses part NAND512W3A2CN6E from Numonyx, a 64 MB chip (512 Mbit). The datasheet *NAND512-A2C Revision 3* from Numonyx specifies that out of 4096 blocks (1 block = 16,384 bytes = 16 KB), the minimum valid blocks over the lifetime of the chip is 4016. That translates to potentially 80 blocks that could be bad when shipped or that can develop over time. 80 blocks of 4016 is ~1.95% or ~1.25 MB of 64 MB. This is found in Table 4: 'Valid Blocks' of the *NAND512-A2C Revision 3* datasheet. Figure 2.1, below, diagrams the arrangement of blocks and pages within a NAND device.

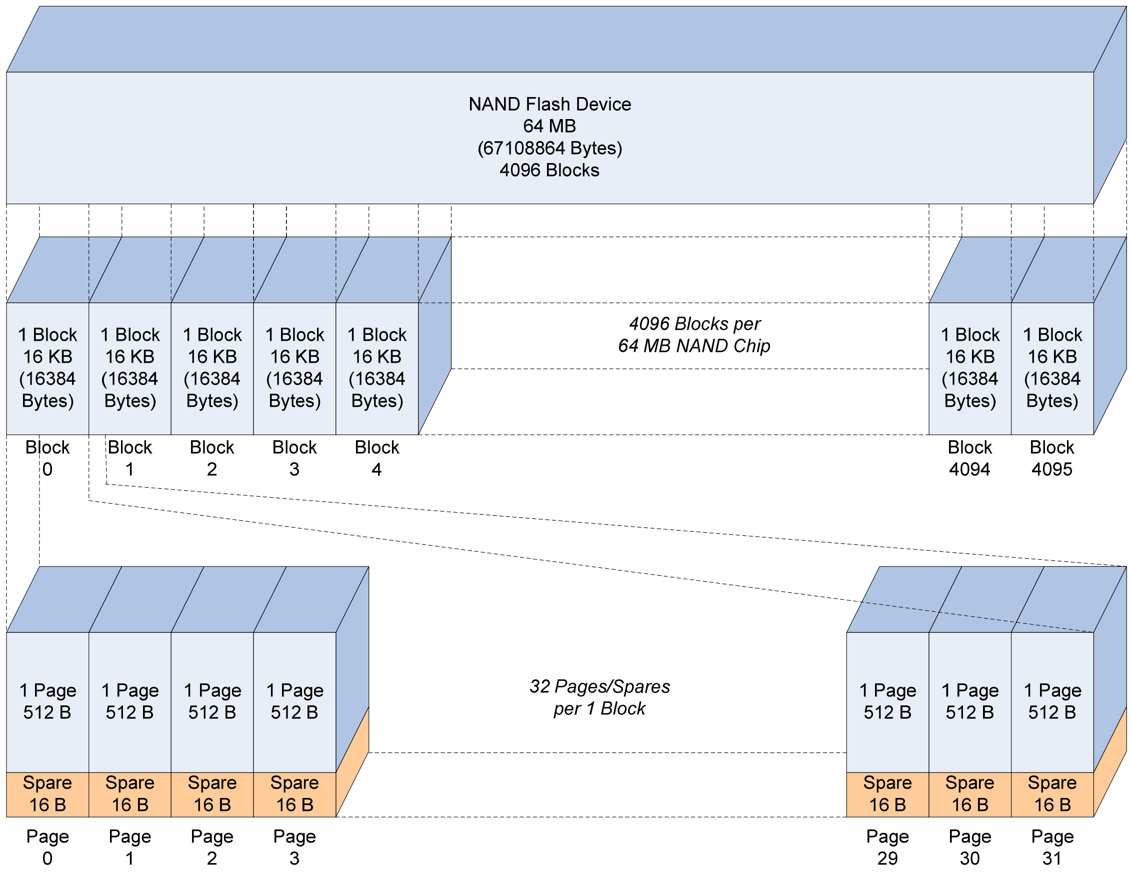


Figure 2.1: NAND Flash Block Management

It is extremely important to keep the potential of bad blocks in mind when using a file system on the NAND device. Plan for 2% of the blocks on device to go bad or already be bad. Keep a reserved space of 1.25 MB on each partition, as the file system will potentially need to use this space when moving files around to manage bad blocks. If multiple partitions are used, each partition must have 80 blocks reserved (~1.25MB) as those 80 bad blocks could occur anywhere on the device, potentially even all in one partition. The Numonyx datasheet also states that the device supports a minimum of 100,000 program/erase cycles per block. Any file system running on the NAND flash must have bad block management and wear-leveling implemented.

Because of the potential complexity of bad block management, Logic recommends only using one partition that covers the entire size of the NAND device. If multiple partitions are used, each must plan for 80 blocks to go bad, thereby doubling the number of ‘reserved’ blocks required, as well as additional metadata files needed. Figure 2.2, below, shows at a high level how bad blocks can be found in a NAND device. The figure also shows how two NAND devices with the same files can have the data arranged differently around bad blocks.

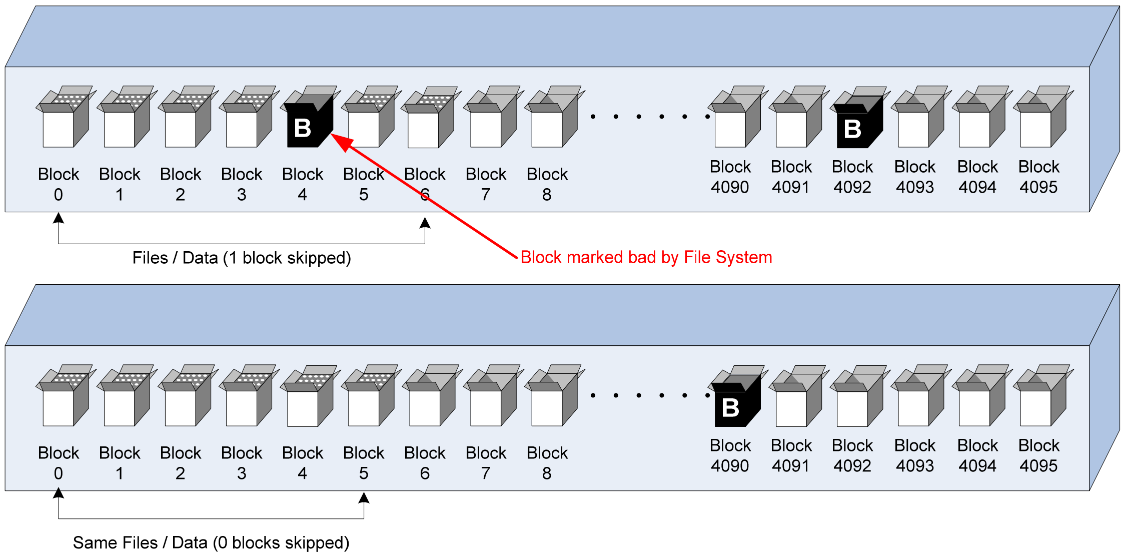


Figure 2.2: Locating Bad Blocks in NAND Flash

2.5 LogicLoader 'dd' command

The *LogicLoader User Manual* has a section dedicated to block devices, like NOR and NAND flash, including commands that can be used on these devices. The 'dd' command is a common Unix program that is available on almost every OS and toolset. LogicLoader also has a version of this command, which is useful for creating manufacturing programming files. The command's purpose is for conversion and copying of low-level raw data; it copies every piece of information on a device, including file system data, as opposed to the 'cp' command, which simply copies the active files on a device.

Logic has added some extra functionality to the 'dd' command, including the ability to skip bad blocks (skip_bad), which is active by default. The ability to skip bad blocks is important when copying from and to NAND flash devices.

Usage:

```
dd if:<source file/device> of:<destination file/device> [count:<
chunks> ibs:<bytes> obs:<bytes> is:< bytes> os:< bytes> seek:<blocks>
skip:<blocks> skip_bad:<0|1> bs:<bytes>]
```

- Copies <count> number of pages/chunks from <if> to <of>
- <ibs>/<obs>: The chunk size (in bytes) can be set with the <ibs> and <obs> parameters for source/destination file or with <bs> which sets <ibs>=<obs>=<bs>
- <bs>: forces <ibs>=<obs>=<bs> block size in bytes
- <skip>: skips <skip> number of <ibs>-sized blocks at the beginning of the source file (<if>)
- <seek>: skips <seek> number of <obs>-sized blocks at the beginning of the destination file (<of>)
- <skip_bad>: Choose to skip bad blocks (default=1) (Only set to 0 for looking at bad block info)
- <is>/<os>: Skips <is>/<os> bytes after every block in the source/destination file.
- Only the <if> and <of> parameters are required, the others will be set to their default:
- Defaults:
    - count=1

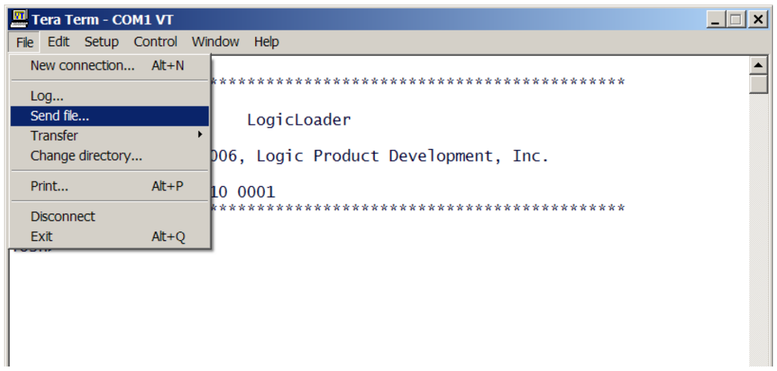
```
o  skip_bad=1
o  ibs/obs = 512
o  all the other options are set to 0
```

3 Standard Products

All standard configuration SOMs ship with LogicLoader installed; the bootloader software that boots to a command prompt and is accessible via the debug serial port UARTA. Customers can use a PC and a terminal emulation program (such as Tera Term) to send LogicLoader commands and scripts over the serial port to the SOM. These commands and scripts can be used to load custom software to the SOM. The *LogicLoader User's Manual* has more information on the functionality and commands available; additionally, Sections 4.1 and 4.2 in this document explain how to create raw programming files for NOR and NAND flash.

3.1 LogicLoader Scripting Example

An example would be to use Tera Term to send a script to LogicLoader. The Tera Term 'File' menu has a command called 'Send File' which allows for a file to be sent from the PC to the SOM. This file can be a text file with LogicLoader commands, which is essentially a script.



```
*********************************************
    LogicLoader
006, Logic Product Development, Inc.
10 0001
*********************************************
```

Figure 3.1: 'Send File' Command

The script could look something like this:

```
mount fatfs /dev/sdmmc0a /sd -rw;
video-open 5 16;video-clear w;
source /sd/autoscript.txt;
exit;
```

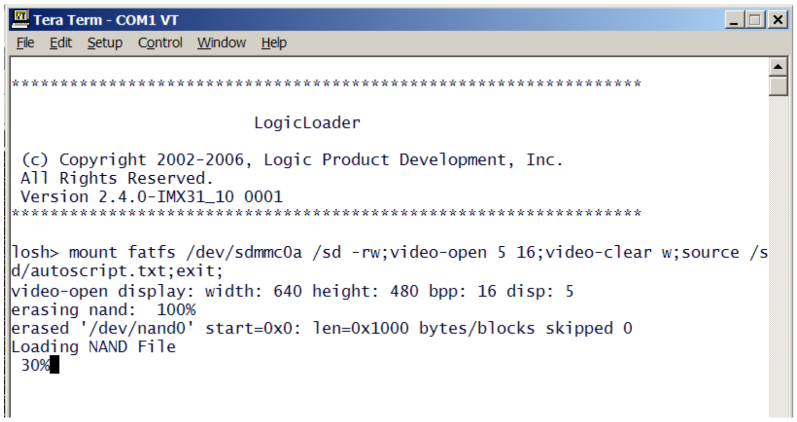
The commands in this script mount an SD card, enable the video (for user feedback), and then run another script that is located on the SD card.

The 'autoscript.txt' file located on the SD card could look something like this:

```
pass=1;
bitmap /sd/erasing_nand_flash.bmp;
erase B0 B4096 /dev/nand0;
if($?)
  bitmap /sd/erase_fail.bmp;
  echo "ERROR, erase failed";
  pass=0;
else
  echo "Loading NAND File";
  bitmap /sd/loading_nand_flash.bmp;
  dd if:/sd/NAND0.IMG of:/dev/nand0 count:128512 ibs:528 obs:528;
```

```
    bitmap /sd/complete_nand_flash.bmp;
  endif;
  echo "DONE";
  if($pass)
    echo "PASSED";
    bitmap /sd/passed.bmp;
  else
    echo "FAILED";
    bitmap /sd/failed.bmp;
  endif
  exit;
```

This script erases the NAND flash and copies the raw file into NAND flash, showing bitmaps to the user indicating the current stage of the process. A raw NAND flash file does not have to be used, instead the script could mount a YAFFS partition and copy individual files over to the partition.



```
*****************************************************************

                         LogicLoader

 (c) Copyright 2002-2006, Logic Product Development, Inc.
 All Rights Reserved.
 Version 2.4.0-IMX31_10 0001
*****************************************************************

losh> mount fatfs /dev/sdmmc0a /sd -rw;video-open 5 16;video-clear w;source /s
d/autoscript.txt;exit;
video-open display: width: 640 height: 480 bpp: 16 disp: 5
erasing nand:  100%
erased '/dev/nand0' start=0x0: len=0x1000 bytes/blocks skipped 0
Loading NAND File
 30%
```

Figure 3.2: Example Script Output

The examples presented here are very basic; using the associated LogicLoader documentation, more advanced loading schemes can be created. Besides SD memory cards, CompactFlash and TFTP are also available as download mediums. For example, the initial script presented above could use TFPT and would look like this:

```
ifconfig sm0 192.168.0.2 255.255.255.0 192.168.0.1;
video-open 5 16;video-clear w;
source /tftp/192.168.0.1:autoscript.txt;
exit;
```

4 Custom Product

Logic can also build custom i.MX SOM configurations. The NPI (New Product Introduction) process provides customers the ability to specify unique hardware populations and/or preload custom software on the SOM. This means the SOM will meet their exact needs when it arrives from the manufacturers. To proceed with the custom SOM NPI approach, minimum requirements must be met; a Logic sales contact can provide additional information and the NPI process document.

If the customer decides to have custom software loaded on the SOM during manufacturing, that software needs to be sent to Logic. On the i.MX31 and i.MX27 SOMs, this may include the NOR flash and/or NAND flash contents.

The following is a list of important aspects to consider about custom software on the i.MX SOM:

1. Larger file sizes = longer time to load = higher manufacturing costs and lead times. Understand how much of the memory is being used to be sure the manufacturing time is minimized.
2. NAND flash can have bad blocks. As mentioned in Section 2.4 above, for a 64 MB NAND device plan to reserve 80 blocks per partition to manage bad blocks.
3. A change to custom software preloaded on a SOM requires changing the model number, creating a manufacturing ECO (engineering change order), initiating the NPI process to build new first articles, and managing a new model number with distribution. This can cause several issues, including increased lead time, associated costs for the changes, and configuration management with distributors.

   To help avoid these issues, it may make sense to use a basic, small software file with the only purpose of being updated later in the end-product manufacturing process. This file consist of a LogicLoader configuration block containing a script to automatically mount an SD card and run a script from the SD card or it may be a basic OS with update features built-in. The benefit of this approach is the likelihood of never having to change the custom software.

   In the case of an auto-running boot script, there is only one thing Logic would ever burn into NOR flash: LogicLoader and the configuration block that contains the script. When the boards are received by the customer, the initial boot-up will initiate the auto-run script to look for the installation script on an SD card, CompactFlash, or over TFTP, and then load all the software on the SOM. This process allows the customer to more easily manage software updates at their manufacturer.

4. Plan for an “in-field” update process. How will the final product be updated once it is released to the field? Can this update process also be used during manufacturing? If the update software is already in place on the custom SOM, software updates can be managed effectively at manufacturing.

4.1 Creating a Custom NOR Flash Software File

In order for Logic to program the NOR flash, a raw program file must be created that is essentially a snapshot of the contents in NOR flash memory. LogicLoader commands can be used to create this file.

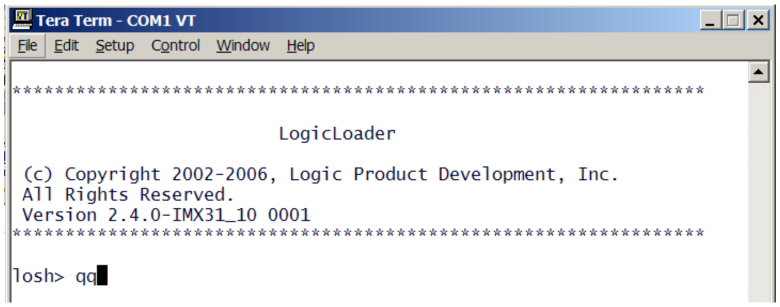
4.1.1 Creating the NOR raw programming file

What you will need:

- Two i.MX SOMs. (Two SOMs provide confirmation that the file is downloading correctly).
- A baseboard, like the Zoom i.MX LITEKIT baseboard, that has a CompactFlash or SD slot.
- A CompactFlash or SD card, large enough for a 2 MB file (the following procedure will assume use of an SD card).
- A serial connection to a PC to use LogicLoader’s command prompt.

Procedure:

1. Set up the SOM with the software in NOR flash exactly how it is to be for manufacturing. This includes LogicLoader, the config block (if used), and any other data that is stored in NOR flash. **Note:** If LogicLoader does not reside in NOR flash, see Section 4.1.2 below for preliminary steps before continuing with the procedure below.
2. Boot the board. If the board automatically runs a script at boot, use the ‘q’ command to break into the LogicLoader command prompt.



```
*********************************************************************

                          LogicLoader

 (c) Copyright 2002-2006, Logic Product Development, Inc.
 All Rights Reserved.
 Version 2.4.0-IMX31_10 0001
*********************************************************************

losh> qq
```

Figure 4.1: Use of the 'q' Command

3. If applicable, check the NOR flash content to make sure it is correct, including the LogicLoader version. In this example, also check the boot script located in the configuration block to make sure it is correct.

```
losh> cat /dev/config
LOLOmount fatfs /dev/sdmmc0a /sd -rw;source /sd/mfgcript.txt;exit;
```

4. Check the details of the memory configuration, which gives necessary information for commands used in steps below. In this case, the base address of NOR flash (flash0) is 0xA000_0000 (i.MX31-specific) and contains 32 blocks. The size is 32 blocks multiplied by 65,536 bytes per block, which equals 2,097,152 bytes (0x0020_0000 hex). The NAND information is not necessary for this NOR flash file creation

```
losh> info mem
Configuration of /dev/flash0:
        base_address:    a0000000
        num_blocks:      32
        bytes_per_block: 65536
        has_chunks:      0
Configuration of /dev/nand0:
        num_blocks:      4096
        bytes_per_block: 16896
        has_chunks:      1
        NAND_0 block 0 is marked bad by YAFFS
        num_chunks:       131072
        chunks_per_block: 32
        bytes_per_chunk:  528
        bytes_per_spare:  16
```

5. Mount the SD card as read/write.

```
losh> mount fatfs /dev/sdmmc0a /sd -rw
```

6. Extract the raw file using the 'dd' command. (Information about the 'dd' command can be found in the *LogicLoader User's Manual*.) This example assumes a copy of all 2 MB of NOR flash (size = 0x0020_0000 bytes, or 32 blocks, which is used by the 'dd' command).

```
losh> dd if:/dev/flash0 of:/sd/FLASH0.IMG count:32 ibs:65536 obs:65536
100%
in:32 out:32  skipped: in:0 out:0    (r:0)
```

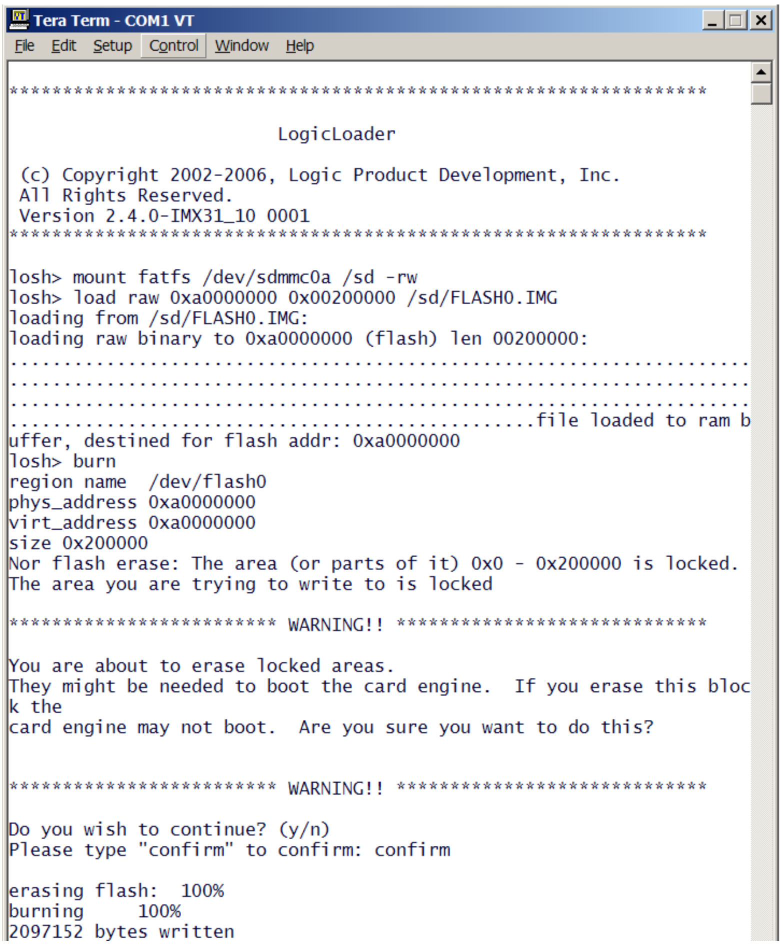
7. Perform a check with an md5sum on the memory in NOR flash and the raw file created to be sure they match.

```
losh> md5sum 0xa0000000 0x00200000
f7cdc6b27e81b40ca22c42d4d1c2c3c3
losh> md5sum /sd/FLASH0.IMG
```

```
f7cdc6b27e81b40ca22c42d4d1c2c3c3
```

8. Using the second SOM, mount the SD card and use the 'load raw' and 'burn' commands to burn the raw file. (**Note:** Make sure that the board does not lose power during this time; any interruption could cause the burn to fail and the board would no longer be able to boot properly without using a JTAG device to reprogram.) LogicLoader will ask for confirmation that parts of memory can be erased, type 'y' and 'confirm' at these steps. The commands should look like this and the output of this sequence is shown in Figure 4.2 below:

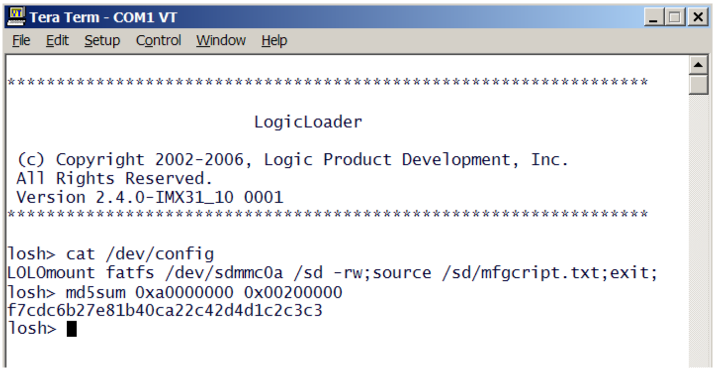
```
losh> mount fatfs /dev/sdmmc0a /sd -rw
losh> load raw 0xa0000000 0x00200000 /sd/FLASH0.IMG
losh> burn
```



```
*****************************************************************

                         LogicLoader

 (c) Copyright 2002-2006, Logic Product Development, Inc.
 All Rights Reserved.
 Version 2.4.0-IMX31_10 0001
*****************************************************************

losh> mount fatfs /dev/sdmmc0a /sd -rw
losh> load raw 0xa0000000 0x00200000 /sd/FLASH0.IMG
loading from /sd/FLASH0.IMG:
loading raw binary to 0xa0000000 (flash) len 00200000:
.....................................................................
.....................................................................
.....................................................................
.................................................file loaded to ram b
uffer, destined for flash addr: 0xa0000000
losh> burn
region name  /dev/flash0
phys_address 0xa0000000
virt_address 0xa0000000
size 0x200000
Nor flash erase: The area (or parts of it) 0x0 - 0x200000 is locked.
The area you are trying to write to is locked

************************* WARNING!! *****************************

You are about to erase locked areas.
They might be needed to boot the card engine.  If you erase this bloc
k the
card engine may not boot.  Are you sure you want to do this?


************************* WARNING!! *****************************

Do you wish to continue? (y/n)
Please type "confirm" to confirm: confirm

erasing flash:  100%
burning     100%
2097152 bytes written
```

Figure 4.2: Example Output of Burning Raw File to Flash

9. If erasing and burning come back at 100% and 2097152 bytes (0x00200000) were written, the board can be powered off and then back on. The board should run code as expected for production. A check for correct implementation can be performed by again breaking into the LogicLoader command prompt using the 'q' command and performing an md5sum check.



```
*****************************************************************

                         LogicLoader

 (c) Copyright 2002-2006, Logic Product Development, Inc.
 All Rights Reserved.
 Version 2.4.0-IMX31_10 0001
*****************************************************************

losh> cat /dev/config
LOLOmount fatfs /dev/sdmmc0a /sd -rw;source /sd/mfgcript.txt;exit;
losh> md5sum 0xa0000000 0x00200000
f7cdc6b27e81b40ca22c42d4d1c2c3c3
losh> 
```

Figure 4.3: Md5sum Check

10. Logic will need the FLASH0.IMG file. Send Logic a Zip file containing the FLASH0.IMG file and a description of what is in the file, including addresses if possible.

4.1.2 Creating the NOR raw programming file without LogicLoader

If LogicLoader is not a part of the NOR flash contents because a custom bootloader is used, preliminary steps to extract the raw file are required. Logic recommends using either a JTAG device or the custom bootloader code to extract the raw file.

If using a JTAG device, a RAM version of LogicLoader can extract the raw file.

1. Connect the JTAG device, like an Abatron BDI2000, to the SOM.
2. Boot the SOM using the JTAG device.
3. Load and run the SRAM setup .elf file (10XXXXX _SRAM_boot.elf) available in the LogicLoader release package.
4. Load and run the RAM LogicLoader (10XXXXX_RAM_lolo.elf) from the release package
5. From this point, follow the procedure in Section 4.1.1 above.

4.2 Custom NAND Flash Software

In order for Logic to program the NAND flash of a custom SOM, a raw program file must be created that is essentially a snapshot of the NAND flash memory contents. LogicLoader commands can be used to create this file. Note that this raw image will not look like a typical RAM memory, as the block and page structure complicates things and the NAND flash can contain a file system.

4.2.1 Creating the NAND raw programming file

What you will need:

- Two i.MX SOMs. (Two SOMs provide confirmation that the file is downloading correctly).
- A baseboard, like the Zoom i.MX LITEKIT baseboard, that has a CompactFlash or SD slot.
- A CompactFlash or SD card, large enough for the full size of NAND plus the spare area bytes, approximately 65 MB for a 64 MB NAND chip (the following procedure will assume use of an SD card).
- A serial connection to a PC to use LogicLoader’s command prompt.

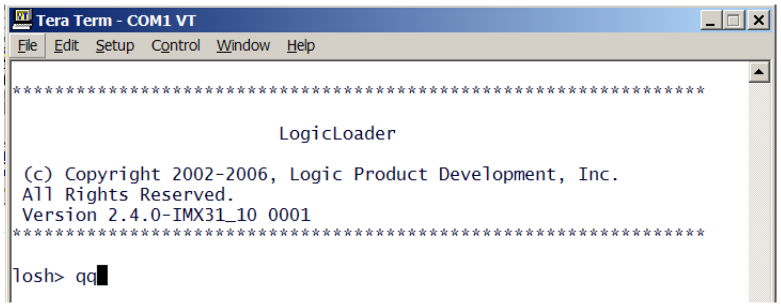
Procedure:

Important Note: If using a file system on the NAND device (such as YAFFS), Logic recommends erasing the NAND flash, placing the files on that device, then creating the

NAND raw programming file. When a file is erased, moved, or modified, most file systems will retain data in the file system and simply update the file meta data. To ensure that no unnecessary or unwanted data is in the NAND raw programming image, Logic recommends starting with an erased flash device. It is also important that the files on the partitions be at the beginning of the partition. To erase the entire 64 MB of NAND flash, use the following command:

```
losh> erase B0 B4096 /dev/nand0;
```

1. Set up the SOM with the software in NAND flash exactly how it is to be for manufacturing. This includes LogicLoader, the config block (if used), file system partitions, and any other data that is stored in NAND flash. **Note:** If you are not using LogicLoader, see Section 4.2.2 below for preliminary steps before continuing with the procedure below.
2. Boot the board. If the board automatically runs a script at boot, use the 'q' command to break into the LogicLoader command prompt.



```
**********************************************************************

                           LogicLoader

 (c) Copyright 2002-2006, Logic Product Development, Inc.
 All Rights Reserved.
 Version 2.4.0-IMX31_10 0001
**********************************************************************

losh> qq
```

3. Check the details of the memory configuration, which gives necessary information for commands used in steps below. In this case, the page size is 512 bytes (bytes per chunk minus bytes per spare) and the spare size is 16 bytes. The total space available to the user is the page size (512 bytes) multiplied by the number of chunks (131,072), which equals 67,108,864 bytes (64 MB, 0x0400_0000 hex). The total size does not include the spare area, which adds 16 bytes per page (chunk). The NOR information is not necessary for this NAND flash file creation

```
losh> info mem
Configuration of /dev/flash0:
        base_address:    a0000000
        num_blocks:      32
        bytes_per_block: 65536
        has_chunks:      0
Configuration of /dev/nand0:
        num_blocks:      4096
        bytes_per_block: 16896
        has_chunks:      1
        NAND_0 block 0 is marked bad by YAFFS
        num_chunks:       131072
        chunks_per_block: 32
        bytes_per_chunk:  528
        bytes_per_spare:  16
```

4. This example will use the entire size of the NAND flash (64 MB), although if it is known that the file system on the device is smaller than the total size, it is recommended to use the known file system size instead. This can save significant time during production when burning hundreds or thousands of boards.

   To create the NAND raw programming file it is necessary to know the total number of pages of the NAND contents. This example uses the entire 64 MB or 67,108,864 bytes (4096 blocks). To plan for the possibility that this partition could have 80 bad blocks, only 4016 blocks will be copied. (As mentioned above, it is important that device has all of its files placed at the beginning of the NAND partition, which can be assured by erasing the NAND prior to placing the files on it.) 4016 blocks multiplied by 32 chunks per block equals 128,512 chunks (pages). Existing bad blocks will be skipped on the device during the copy.

5. Mount the SD card

```
losh> mount fatfs /dev/sdmmc0a /sd –rw
```

6. Extract the raw file using the 'dd' command. (Information about the 'dd' command can be found in the *LogicLoader User's Manual*.) This example assumes a copy of all 62.75 MB (the size of 128,512 pages that is used by the 'dd' command) of NAND flash.

```
losh> dd if:/dev/nand0 of:/sd/NAND0.IMG count:128512 ibs:528 obs:528
100%
in: 128512 out: 128512 skipped: in:0 out:0    (r:0)
```

7. Perform a check with an md5sum on the files in NAND flash if YAFFS is mounted as a file system; also look at the directory and file structure.

```
losh> part-add /dev/nand0 a 0 4096;
losh> mount yaffs /dev/nand0a /YaffsPart1;
losh> md5sum /YaffsPart1/NK.bin
      125c64a58c84c3ffa268c88358bea2ee
losh> ls /YaffsPart1
     :                           NK.bin  8166363
     :                      testlog.txt       85
   D :                       lost-found      512
losh>
```

8. Using a second SOM if possible, erase all of the NAND flash memory to make sure the file burns appropriately, then do a complete power cycle to make sure that no file system partition info remains in RAM.

```
losh> erase B0 B4096 /dev/nand0;
```

9. Mount the SD card and use the 'dd' command to burn the raw file from the SD card to the NAND flash. (**Note:** make sure that the board does not lose power during this time; any interruption could cause the burn to fail and the board would no longer be able to boot properly without using a JTAG device to reprogram.) The commands should look like this:

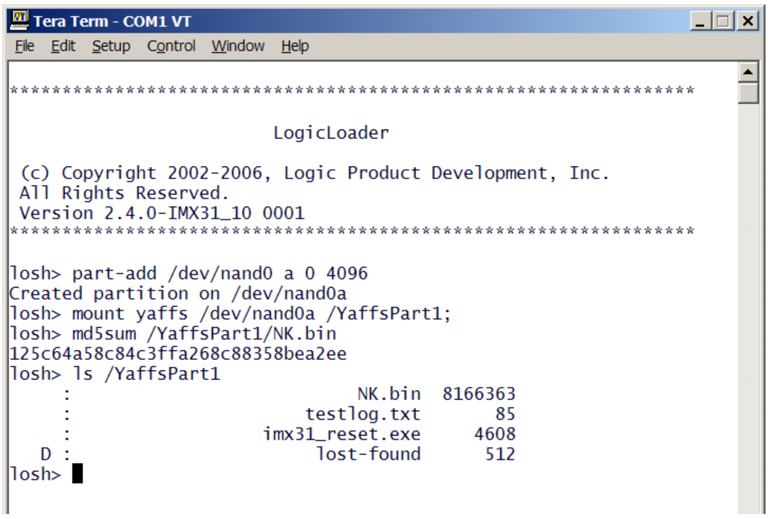
```
losh> mount fatfs /dev/sdmmc0a /sd -rw
losh> dd if:/sd/NAND0.IMG of:/dev/nand0 count:128512 ibs:528 obs:528
```

and the output of this sequence should look like this:

```
 100%
 in:128512 out:128512  skipped: in:0 out:0    (r:0)
losh>
```

10. If the 'dd' command returns without errors or with an 'unable to mount' error, the board can be safely powered off and back on. When powered back on, the board should run code as expected for production.

11. A checksum can be performed on specific files in YAFFS and should match the original md5sum values. Also, the file structure should look the same; see Figure 4.4 below.



```
*****************************************************************

                         LogicLoader

 (c) Copyright 2002-2006, Logic Product Development, Inc.
 All Rights Reserved.
 Version 2.4.0-IMX31_10 0001
*****************************************************************

losh> part-add /dev/nand0 a 0 4096
Created partition on /dev/nand0a
losh> mount yaffs /dev/nand0a /YaffsPart1;
losh> md5sum /YaffsPart1/NK.bin
125c64a58c84c3ffa268c88358bea2ee
losh> ls /YaffsPart1
     :                    NK.bin  8166363
     :                testlog.txt       85
     :            imx31_reset.exe     4608
   D :                 lost-found      512
losh> █
```

Figure 4.4: Md5sum Check

12. Logic will need the NAND0.IMG file. Send Logic a Zip file containing the NAND0.IMG file, a description of what is in the file, and a document that includes all the steps used to create the image (including command line arguments used in the 'dd' command). An easy way to provide the steps used to create the image is to send a terminal log of the LogicLoader session that created the image. Providing this information will help Logic burn the image and verify that all data is properly captured, as well as to help troubleshoot any potential failures in the future. **Note:** For large file transfers, Logic can set up a WebDAV (i.e., FTP) account for you.

4.2.1.1 Creating the NAND raw programming file for multiple partitions

If two partitions are contained on the NAND device, a raw programming file will have to be created for each partition (e.g., NAND0a.IMG and NAND0b.IMG). If the 'dd' command were used to create one image for an entire NAND device with two partitions, it could write file data from one partition to the other since it has no idea of what a partition is or where the partitions are located. The following provides an example of using the 'dd' command twice to create two different raw programming files for the two partitions.

For this example, assume there are two partitions, each covering half of the NAND device. Each partition covers 32 MB or 2048 blocks; however, reserving 80 blocks in each partition for bad blocks reduces the size available for the image to 1968 blocks or 62,976 pages (chunks). Both 'dd' commands have the same page count of 62,976 pages, but the second command needs to start at the exact location of the second partition. Specifying this start location can be done with the 'skip' parameter. (**Tip:** The first block is number 0, so the second partition would start on block 2048 or page 65,536; 2048 multiplied by 32 pages per block equals 65,536 pages.)

```
losh> dd if:/dev/nand0 of:/sd/NAND0A.IMG count:62976 ibs:528 obs:528

losh> dd if:/dev/nand0 of:/sd/NAND0B.IMG count:62976 ibs:528 obs:528
skip:65536
```

When creating these two file, it is extremely important to provide Logic with this information (the 'dd' command options) about how they were created so Logic can recreate and program the parts correctly. Both .IMG files and the text showing the 'dd' command usage must be provided to Logic.

When burning to a new NAND device, the two LogicLoader commands would look like what follows:

```
losh> dd if:/sd/NAND0A.IMG of:/dev/nand0 count:62976 ibs:528 obs:528

losh> dd if:/sd/NAND0B.IMG of:/dev/nand0 count:62976 ibs:528 obs:528
seek:65536
```

Notice the second command string uses the 'seek' parameter on the 'of' file; this parameter specifies the start location of the second partition.

4.2.2 Creating the NAND raw programming file without LogicLoader

If LogicLoader is not a part of the NAND flash contents because a custom bootloader is used, preliminary steps to extract the raw file are required. Logic recommends using either a JTAG device or the custom bootloader code to extract the raw file.

If using a JTAG device, a RAM version of LogicLoader can extract the raw file.

1. Connect the JTAG device, like an Abatron BDI2000, to the SOM.
2. Boot the SOM using the JTAG device.
3. Load and run the SRAM setup .elf file (10XXXXX _SRAM_boot.elf) available in the LogicLoader release package.
4. Load and run the RAM LogicLoader (10XXXXX_RAM_lolo.elf) from the release package
5. From this point, follow the procedure in Section 4.2.1 above.

5 Summary

There are several different methods for programming software on the i.MX SOM at manufacturing and the sooner these methods are considered in the development process, the more time will be saved when manufacturing begins. Logic's bootloader, LogicLoader, provides the tools to assist with software programming at manufacturing time. But even if LogicLoader is not used by the customer, this document can assist with the decision making process by describing specifics to consider for how software is burned to the SOM at manufacturing.

6 References

- Logic's *LogicLoader User's Manual (LoLo version 2.4)*
- Logic's *i.MX31SOM-LV Addendum to LogicLoader User's Manual*
- Logic's i.*MX27 SOM-LV Addendum to LogicLoader User's Manual*
- Logic's *LogicLoader Command Description Manual (LoLo version 2.4)*
- Logic's *Application Note 339: LogicLoader Configuration Block Usage*
- Logic's *Customer Module NPI Process* – Available upon request
- Numonyx datasheet *NAND512-A2C Revision 3*