

# Kickstart and Stage 1 loader

# Table of contents

<b>1</b>	<b>Introduction.....</b>	<b>3</b>
<b>2</b>	<b>Kickstart loader .....</b>	<b>4</b>
2.1	<i>Kickstart loader operation .....</i>	4
2.2	<i>Stage 1 applications .....</i>	5
<b>3</b>	<b>Stage 1 loader .....</b>	<b>5</b>
3.1	<i>Stage 1 loader operation process .....</i>	5
3.2	<i>Stage 1 loader resource usage .....</i>	6
3.2.1	IRAM organization.....	6
3.2.2	Stage 1 loader persistent data storage.....	7
3.2.3	First time NAND FLASH configuration .....	7
3.3	<i>Stage 1 loader monitor program operations .....</i>	7
3.3.1	Accessing the monitor program.....	7
3.3.2	Monitor program commands .....	8
3.3.3	Loading and executing files.....	19
3.3.4	Saving files in FLASH .....	22
3.3.5	Setting up autoboot.....	22
3.3.6	Load, save, and boot examples .....	23
<b>4</b>	<b>Notes.....</b>	<b>25</b>
4.1	<i>WinCE and reserved block marking.....</i>	25
4.2	<i>Known issues with SIL.....</i>	25
4.2.1	MMUENAB command sometimes hangs .....	25
4.2.2	INFO shows image loaded in memory on ABOOT failure .....	25
4.3	<i>FLASH controller ECC support .....</i>	25
4.3.1	ECC impact to FLASH commands.....	25

# List of tables

<b>Table 1 Kickstart and stage 1 application FLASH data organization</b>	<b>4</b>
<b>Table 2 Kickstart loader operation</b>	<b>5</b>
<b>Table 3 IRAM organization for stage 1 loader</b>	<b>6</b>
<b>Table 4 Support image load options</b>	<b>20</b>

## 1 Introduction

There are several bootloaders in the NAND FLASH of the Phytex LPC3250 board that help make program loading and development easier. These consist of the kickstart loader and the stage 1 application loader.

The kickstart loader sits in block 0 of FLASH and is responsible for loading an application that starts in FLASH block 1. The main features of the kickstart loader are shown below:

- Allows loading images greater than 1 block length (stage 1 application) (*An image of about 54K is the maximum size that can be booted with the LPC3250 boot ROM*)
- Loads stage 1 applications into internal RAM (IRAM) at address 0x0

The stage 1 loader is the default application loaded by the kickstart loader. The stage 1 loader provides a monitor program with a collection of functions to help with application development. The main features of the stage 1 loader are shown below:

- Register and memory change and dump
  - Poke, peek, dump, fill
- Image load via a serial port , SDMMC card, or FLASH
  - Supports raw binary and S-record files
  - Images can be executed after loading
  - Images can be saved in FLASH
- NAND FLASH support
  - Erase of NAND blocks
  - Direct read and write of FLASH blocks and pages
  - Bad block management
  - Reserved block management for operating systems
- MMU functions
  - Data and instruction cache control
  - Virtual address translation enable/disable
  - Virtual address remapping
  - Page table dump
- System support functions
  - Baud rate control, clock control, system information
- Automatic load and run support
  - Automatic load and execution of images from FLASH, SDMMC, or via the terminal

The source code for the kickstart loader is provided in the Phytex LPC3250 Board Support Package (BSP).

## 2 Kickstart loader

The kickstart loader sits in block 0 of FLASH and is responsible for loading an application that starts in FLASH block 1. The kickstart loader is loaded and executed when the LPC3250 is powered up and/or reset by the LPC3250 internal boot ROM (IROM). The IROM FLASH load function has a load size restriction of 1 block. The kickstart loader, once loaded by the internal boot ROM, allows loading of images that span multiple blocks starting at FLASH block 1. Normally, the kickstart loader will load and start the stage 1 loader when the Phytex LPC3250 board is started. Optionally, user applications can be placed into FLASH block 1 and started by the kickstart loader.

### 2.1 Kickstart loader operation

When the LPC3250 is powered and/or reset, the internal boot ROM checks the image in FLASH block 0 and loads it if it is valid. *See the LPC3250 User's Guide for more information on the boot process.* The image is loaded and execution is started at address 0x0. Once loaded, the image is moved to the top 16K of internal IRAM (IRAM). This leaves 240K of IRAM starting at address 0x0 available for the stage 1 application.

The kickstart loader then loads page 0 of block 1 to get the sizing information for the stage 1 application. The kickstart loader will skip any blocks during the load process if the block is marked as bad, so if block 1 is bad, it will load block 2. The first 4 bytes of the data from page 0 are used as the size in bytes of the image to load starting at page 1 of the same block. The second 4 bytes in page 0 are the 1's complement value of the first 4 bytes and is used to verify that the image stored in FLASH for the stage 1 application is valid. With the loaded size, the kickstart application will continue reading blocks and pages into IRAM starting at address 0x0 until all the data is loaded. Bad blocks will be skipped.

Once all the data is loaded, execution is passed to the image at address 0x0. When burning an image into FLASH starting at block 1 that is to be loaded with the kickstart loader, it is important to correctly set the size and 1's complement of the size in the first page of the first good block after block 0 of the kickstart loader may not work correctly.

For a small block NAND FLASH device with 32 pages per block, the kickstart loader and stage 1 application are organized as shown in Table 1.

**Table 1 Kickstart and stage 1 application FLASH data organization**

Block	Page(s)	Data in NAND FLASH
0	0	ICR data needed by the LPC3250 IROM
0	1-31	Kickstart loader
1 <sup>1</sup>	0	Size of stage 1 application (4 bytes at offset 0) and 1's complement of size (4 bytes at offset 4)
1 <sup>1</sup>	1-31	Start of stage 1 application

2-? <sup>1</sup>	0-31	Stage 1 application code
------------------	------	--------------------------

<sup>1</sup>Bad blocks will be skipped. Block 0 is assumed to always be good.

The LPC3250 IROM and kickstart loader sequence is shown in Table 2.

**Table 2 Kickstart loader operation**

Operation	Function
LPC3250 is reset	IROM loads image from NAND FLASH based on ICR data
Kickstart loader started	IROM passes control to kickstart loader at address 0x0
Kickstart relocation	Kickstart loader is relocated to top 16K of IRAM and execution is passed to top of IRAM
Get stage 1 application size	Kickstart loader reads first page of first good block after block 0 to get size and inverse size data
Read in data	Kickstart loader reads data starting at address 0x0 from FLASH based on fetched size starting at page 1
Transfer to stage 1 application	Transfer of control is passed to loaded application at address 0x0

## 2.2 Stage 1 applications

Any application that loads and executes at address 0x0 and uses less than 240KB of memory can be used as the stage 1 application. The board is preprogrammed with the stage 1 loader as the default stage 1 application. This application can be replaced with any other stage 1 application as long as the organization shown in Table 1 is followed.

## 3 Stage 1 loader

The default stage 1 application loaded by the kickstart loader is the stage 1 loader. This application initializes the board and chip and provides a monitor program for simple development and program execution options.

### 3.1 Stage 1 loader operation process

The stage 1 loader is started at address 0x0 once it is loaded from the kickstart loader. The stage 1 loader first initializes the board with the code from the phy3250\_startup\_entry.s and phy3250\_startup.c files before starting the monitor program. The startup code sets up the board in states as shown below:

- MMU page table located at address 0x0803C000 (section table only)
- Default MMU virtual to physical address mapping
- MMU is enabled; data and instruction caches are enabled
- Sets up ARM clock to 208MHz, bus clock to 104MHz, and PCLK to 13MHz
- Initializes SDRAM at address 0x80000000
- Sets up initial GPIO directions and states and some peripheral muxing
- Disables all peripheral clocks
- Sets up static memory timing

Initializes all stacks in IRAM

Once startup is completed, the monitor program is started. Depending on how the stage 1 loader is configured, the monitor program may attempt to load and execute an image, or go to the monitor prompt.

### 3.2 Stage 1 loader resource usage

The stage 1 loader uses UART5, SSP0, the SD card controller, the millisecond timer, and the NAND MLC and SLC FLASH controllers during its operation. Data may be saved into FLASH or the serial EEPROM during its use. SDRAM is not directly used by the stage 1 loader, although some interactive operations may affect SDRAM.

When applications are loaded and executed with the stage 1 loader, the loader will disable all its used peripherals prior to executing the loaded application. Some peripheral and loader settings are passed unchanged to the application; these include the settings configured by the startup code. When the application is called, the system is in the following state:

- MMU page table located at address 0x0803C000 (section table only)
- MMU may be enabled or disabled
- Data and instruction caches may be enabled or disabled
- Clock rates as configured by the monitor program
- SDRAM initialized at address 0x80000000
- GPIO directions, states, and muxing unchanged from startup code
- All peripherals disabled
- All stacks in IRAM

Applications called from the stage 1 loader may safely use the resources setup by the stage 1 loader. If the application wants to return to the stage 1 loader, the application must return the system back to its original state before exiting. Applications that require bigger stacks should setup their own stacks as part of the application. If an application overwrites the memory used by the stage 1 loader, returning from the application may cause the system to crash.

#### 3.2.1 IRAM organization

IRAM is organized as shown in Table 3 for the application called from the stage 1 loader.

**Table 3 IRAM organization for stage 1 loader**

IRAM base	Size (bytes)	IRAM use
0x0003C000	16K	MMU page table
0x0003C000 <sup>1</sup>	512	FIQ mode stack
0x0003BE00 <sup>1</sup>	1024	IRQ mode stack
0x0003B800 <sup>1</sup>	512	Abort exception stack
0x0003B600 <sup>1</sup>	512	Undefined instruction exception stack
0x0003B400 <sup>1</sup>	512	System mode stack

0x0003B200 <sup>1</sup>	4096	Service mode stack
0x00000000	233K	Stage 1 loader code and data

<sup>1</sup>Stacks grown down.

### 3.2.2 Stage 1 loader persistent data storage

The stage 1 loader stores configuration data in the serial EEPROM located on SSP0. This data contains information such as the prompt string, baud rate, autoboot parameters, and image stored in FLASH. These values are used by the stage 1 loader between power cycles to remember the last stage 1 loader configuration. Some commands automatically updated this data (such as the prompt command) whenever a configurable options is changed. The first time the board is powered up, the data is populated with default values.

The following configurable items can be changed with the corresponding data stored in the serial EEPROM:

- Prompt string
- Autoboot timeout
- Autoboot source, file type, load address, filename, execution address
- FLASH initial format status
- Configured baud rate
- Saved FLASH image status
- System clock settings

#### 3.2.2.1 Resetting default values

If for some reason, the board is configured in such as way as to become unbootable after the stage 1 loader starts, holding down the BTN1 button as the board is booting will restore the default values for the configuration data. The restored values will only apply for that power cycle.

#### 3.2.3 First time NAND FLASH configuration

The first time the board is powered up, the NAND FLASH is checked for bad blocks and the bad blocks are relocated if necessary<sup>1</sup>. A flag indicating this operation was performed is saved in persistent storage and can not be erased by resetting the default values.

<sup>1</sup>Bad block relocation is not performed or needed for small block FLASH.

## 3.3 Stage 1 loader monitor program operations

This section explains the monitor program operations that can be executed from the command prompt of the stage 1 loader.

### 3.3.1 Accessing the monitor program

The monitor program can be accessed by connecting a serial cable to the bottom serial connector on the Phytex LPC3250 board. A terminal program (such as Teraterm) configured for 115.2Kbits/sec, 8 data bits, no parity, and 1 stop bit will be needed. If the

©2006-2007 NXP Semiconductors. All rights reserved.

default configuration is used, a prompt such as *PHY3250>* should appear on the terminal. If the default configuration has been changed, the prompt may be different or another program may have been configured to be autobooted.

Commands are entered at the monitor program prompt. All commands are case insensitive. Pressing DEL on the current command line will erase the current command entered. After each command has been typed, pressing enter will execute the command.

### 3.3.1.1 Command syntax

Commands consist of a base command and a required or optional list of arguments. A command has the following syntax:

*command [required argument] <optional argument>*

Arguments for commands with options may not always be required. For some commands, arguments are required and the command will fail if the arguments are not specified.

Argument types vary per command, but most commands share the type of arguments that can be used. Examples of common argument types are listed below and the required format for that argument.

*<hex>*

This value must be a hexadecimal value preceded with a “0x” such as 0x1234, 0x00800100, or 0x9.

*<bytes>*, *<width 1, 2, 4>*, *[starting sector]*, etc.

These values are decimal such as 128, 1, or 500.

## 3.3.2 Monitor program commands

### 3.3.2.1 Support and configuration commands

#### 3.3.2.1.1 *help*

The *help* command is used to display the menu or the syntax of a specific command. Type *help menu* to see the current supported commands, or type *help <command>* to display the syntax and instructions for a specific command.

Command syntax:

*help menu*

*help all*

*help <group>*

*help <command>*

#### 3.3.2.1.2 *baud*

The *baud* command is used to reset the baud rate. The usable baud rates in bits per second are 115200, 57600, 38400, 19200, and 9600. Once a baud rate has been set, it becomes immediately effective. Configured baud rate is persistent and will remain set to this new value across power and reset cycles.

Command syntax:

*baud [new baud rate (115200, 57600, 38400, 19200, 9600)]*

Persistent configuration:

Saved across power cycles and resets.

### 3.3.2.1.3 *bwtest*

Measures the amount of Microseconds required to copy data from one region of memory to another. This function can be used to measure typical bandwidth between cached and uncached SDRAM and SRAM regions. Burst operations sized at 4 32-bit words are used to transfer data between the source and destination. Both the source and destination addresses must be aligned on a 32-bit boundary. The number of bytes to test must be divisible by 16. The actual number of bytes transferred and the total time in MicroSeconds required to transfer the data will be displayed.

Command syntax:

*bwtest [hex addr 1] [hex addr 2] [bytes] [loops]*

Examples:

*Bwtest 0x80000000 0x82000000 1048576 50 (Move 1048576 bytes of data between source address 0x80000000 and destination 0x82000000)*

### 3.3.2.1.4 *clock*

Resets the system clock rates. The ARM core clock, bus clock (HCLK), and the peripheral clock (PCLK) are all set based on the passed arguments. Care must be used with this command as driving the part too fast may cause the system to crash or become unstable.

Command syntax:

*clock [ARM freq MHz] [HCLK divider (1, 2, 4)] [PERIPH\_CLK divider (1 to 32)]*

Examples:

*clock 208 2 16 (Sets ARM clock to 208MHz, HCLK to 104MHz, and PCLK to 13MHz)*  
*clock 150 75 12 (Sets ARM clock to 150MHz, HCLK to 75MHz, and PCLK to 12.5MHz)*

Notes:

It is recommended that values be selected that prevent the HCLK from exceeding 104MHz and PCLK should be as close to 13MHz as possible. System timings dependent

on the new clock settings will be optimized to get the best performance with this command. A small glitch on the terminal may occur as the clocks are adjusted.

Persistent configuration:

Clock settings are saved across power cycles. If the board fails to boot due to a bad clock setting, boot the board in the default (208MHz) configuration by holding down the BTN1 button when the board is reset.

### **3.3.2.1.5 comp**

Performs a bitwise data comparison between 2 data regions.

Command syntax:

*comp [hex addr 1] [hex addr 2] [bytes]*

Examples:

*Comp 0x80000000 0x82000000 1024 (Compares 1024 bytes)*

### **3.3.2.1.6 copy**

Performs a bitwise data copy between 2 data regions.

Command syntax:

*copy [hex addr 1] [hex addr 2] [bytes]*

Examples:

*copy 0x80000000 0x82000000 1024 (copies 1024 bytes from 0x80000000 to 0x82000000)*

### **3.3.2.1.7 dump**

Dumps a range of data sized in 8-bit, 16-bit, or 32-bit chunks. The command can be used without arguments to continue a dump where it previously ended with the same characteristics of the previous dump command.

Command syntax:

*dump <hex addr> <bytes> <width 1, 2, 4>*

Examples:

*dump 0x80000000 256 4 (Dumps 64 32-bit words starting at address 0x80000000)*

*dump (Dumps data based on the previous dump command at the last address)*

Notes:

Dumping invalid memory ranges when the MMU is enabled will cause the stage 1 loader to generate an exception.

### 3.3.2.1.8 *fill*

Fills a data range with a value.

Command syntax:

*fill [hex addr] [hex bytes] [hex value] [width 1, 2, 4]*

Examples:

*fill 0x80008000 0x80 0xAA 1 (Fills 128 bytes starting at address 0x80008000 with 0xAA)*

*fill 0x80010000 0x40 0x00 2 (Fills 32 16-bit words starting at address 0x80010000 with 0x0000)*

Notes:

Filling invalid memory ranges when the MMU is enabled will cause the stage 1 loader to generate an exception.

### 3.3.2.1.9 *info*

Displays current system information such the MMU enabled status, clock speeds, FLASH geometry, loaded image, image in FLASH, and autoboot configuration.

Command syntax:

*info*

### 3.3.2.1.10 *peek*

Dumps the 8-bit, 16-bit, or 32-bit value at a specific address location. This function is good for examining individual locations. Consecutive calls to peek without arguments will always dump the same address and size.

Command syntax:

*peek <hex addr> <width 1, 2, 4>*

Examples:

*peek 0x80008000 1 (Dumps the hex byte at address 0x80008000)*

*peek (Dumps the same hex information as the previous use of this command)*

Notes:

Peeking invalid memory when the MMU is enabled will cause the stage 1 loader to generate an exception.

### 3.3.2.1.11 *poke*

Places a 8-bit, 16-bit, or 32-bit hex value into an address.

Command syntax:

*poke [hex addr] [hex value] [width 1, 2, 4]*

Examples:

*poke 0x80000000 0x99 1 (Places the value 0x99 into the byte address 0x80000000)*

*poke 0x80000000 0x99 2 (Places the value 0x0099 into the 16-bit address 0x80000000)*

Notes:

Poking invalid memory when the MMU is enabled will cause the stage 1 loader to generate an exception.

### **3.3.2.1.12** *prompt*

Changes the default prompt and autoboot delay. The prompt can be changed to any string with a maximum length of 16 characters. The timeout delay is used to prevent an image from automatically loading and starting when the stage 1 loader is started. A timeout delay of 0 will always cause the command prompt to be displayed.

Command syntax:

*prompt [name] [timeout(secs), 0 = no timeout]*

Examples:

*prompt lpc> 0 (Sets the prompt to lpc> and disabled the autoboot delay)*

*prompt : 10 (Sets the prompt to : and sets the autoboto delay to 10 seconds)*

Persistent configuration:

Saved across power cycles and resets.

### **3.3.2.1.13** *rstcfg*

Restores the default system configuration. Resets the prompt to lpc3250>, the baud rate to 115200, and the autoboot timeout to 0. Also restores default system clock values.

Command syntax:

*Rstcfg*

### **3.3.2.1.14** *ls*

Display the files in the root directory of the SDMMC card. Files are display in 8.3 file format.

Command syntax:

*ls*

Notes:

SD and MMC cards are supported. High capacity cards are not supported.

### **3.3.2.1.15** *maddr*

Change the ethernet MAC address.

Command syntax:

*maddr ha:ha:ha:ha:ha:ha*

Examples:

*maddr 00:08:01:9A:BF:9A*

Persistent configuration:

Saved across power cycles and resets. The MAC address is saved in the Phytec hardware descriptor structure and is not part of the stage 1 loader. Use with care.

### 3.3.2.2 MMU control commands

#### 3.3.2.2.1 *dcache*

Enables, disables, or flushes the data cache. The data cache is only operational when the MMU is enabled. When disabling the data cache, the data cache is flushed prior to the data cache being disabled. When enabling the data cache, the data cache is invalidated prior to being enabled.

Command syntax:

*mmu dcache [0, 1, 2]*

Examples:

*dcache 0 (Flush and disable data cache)*

*dcache 1 (Invalidate and enable data cache)*

*dcache 2 (Flush data cache)*

Persistent configuration: No, always enabled when the stage 1 loader is started.

#### 3.3.2.2.2 *inval (Cache flush and invalidate)*

Flushes and invalidates both the instruction and data caches.

Command syntax:

*inval*

#### 3.3.2.2.3 *icache*

Enables or disables the instruction cache. When enabling the instruction cache, the instruction cache is invalidated prior to being enabled.

Command syntax:

*mmu icache [0, 1]*

Examples:

*icache 0 (Disable instruction cache)*

*icache 1 (Invalidation and enable instruction cache)*

Persistent configuration: No, always enabled when the stage 1 loader is started.

#### **3.3.2.2.4 map**

Maps or unmaps a range of physical addresses to virtual addresses. Depending on the argument type, a virtual address range can be mapped as cached or uncached.

Command syntax:

*map [virt hex addr] [phy hex addr] [sections] [0, 1, 2]*

Examples:

*map 0x40000000 0x80000000 32 0 (Maps 32Mbytes of data at address 0x80000000 to virtual address 0x40000000 as uncached)*

*map 0x40000000 0x80000000 8 1 (Maps 8Mbytes of data at address 0x80000000 to virtual address 0x40000000 as cached)*

*map 0x40000000 0x80000000 32 2 (Unmaps 32Mbytes of data at virtual address 0x40000000)*

Notes:

Unmapped regions will generate an exception if accessed.

Persistent configuration: No, default MMU page table from startup code is used when the stage 1 loader is started.

#### **3.3.2.2.5 mmuenab**

The *mmuenab* command enables and disables the MMU. When disabling the MMU, the data cache is flushed prior to the MMU being disabled. When enabling the MMU, the data and instruction caches are invalidated prior to being enabled.

Command syntax:

*Mmuenab <0, 1>*

Notes:

Use 0 to disable the MMU or 1 to enable it. Because the stage 1 loader's virtual and physical addresses are identical, this can be safely performed in the stage 1 loader.

Persistent configuration: No, always enabled when the stage 1 loader is started.

#### **3.3.2.2.6 mmuinfo**

Dumps the current MMU, data cache, and instruction cache enabled status. Also dumps the current virtual to physical address mapping, region sizes, and cached region status.

Command syntax:

*mmuinfo*

### 3.3.2.3 NAND support commands

#### 3.3.2.3.1 *erase*

Erase a range of FLASH blocks. Bad blocks are not erased unless specifically requested to be erased.

Command syntax:

*erase [starting block] [number of blocks] [erase bad blocks]*

Examples:

*erase 100 1000 0* (Erases blocks 100 to 1099, ignoring bad blocks)

*erase 300 800 1* (Erases blocks 300 to 1099, including bad blocks)

Notes:

Care must be taken to not erase any applications loaded in FLASH. If the kickstart loader or stage 1 loader FLASH data areas are attempted to be erased, verification will be requested for the operation. Erasing an application stored with *nsave* will prevent the *nload* function from working although the *info* command will still indicate that the image is stored in FLASH.

#### 3.3.2.3.2 *nandbb*

Generates a list of bad blocks on the FLASH device.

Command syntax:

*nandbb*

#### 3.3.2.3.3 *read*

Reads a range of FLASH sectors into memory. Bad blocks are normally skipped, but can be optionally loaded during the operation.

Command syntax:

*read [starting sector] [sectors] [hex address] [erase bad blocks]*

Examples:

*read 100 32 0x80000000* (Reads 32 sectors starting at sector 100 into address 0x80000000, bad blocks are skipped)

*read 480 64 0x80000000 1* (Reads 64 sectors starting at sector 480 into address 0x80000000, bad blocks are also read)

Notes:

Sectors do not need to be aligned to the first page in a block.

#### 3.3.2.3.4 *write*

Writes data from memory into a range of FLASH sectors. Bad blocks are normally skipped, but can be optionally used during the operation.

Command syntax:

*nwrite [starting sector] [sectors] [hex address] [erase bad blocks]*

Examples:

*write 100 10 0x81000000 0* (Writes 10 sectors of data from address 0x81000000 to FLASH starting at sector 100, skips bad blocks)

*write 500 25 0x81000000 1* (Writes 25 sectors of data from address 0x81000000 to FLASH starting at sector 500, bad blocks are not skipped)

Notes:

Sectors do not need to be aligned to the first page in a block.

#### 3.3.2.3.5 *nburn*

Places an image loaded into memory at a contiguous location in FLASH. This function is typically used to place large amounts of data in FLASH.

Command syntax:

*Nburn [starting block][overwrite bad blocks]*

Example:

*nburn 100 0* (Stores the image in FLASH starting at block 100, skips bad blocks)

#### 3.3.2.3.6 *nrsv*

Marks a range of blocks as reserved. See Section 4.1 for more information on this command.

Command syntax:

*nrsv [first block][number of blocks]*

Example:

*nrsv 25 600* (Marks 600 blocks starting at block 25 as reserved)

#### 3.3.2.3.7 *nandrs*

Displays a list of reserved and non-reserved blocks.

Command syntax:

*nandrs*

Notes:

Bad blocks will shown up as reserved blocks with this command. Trying to reserve a bad block is not possible.

### 3.3.2.3.8 *nexdump*

Dumps the data in the spare data area of a NAND FLASH range of sectors.

Command syntax:

*nexdump* [*starting sector*][*number of sectors*]

Example:

*nexdump* 3200 10 (*Dumps 10 sector's spare data area starting at sector 3200*)

### 3.3.2.4 Application support commands

These commands support loading, saving, executing, and booting user applications.

#### 3.3.2.4.1 *nload*

Loads an image stored in FLASH into memory. The image was previously saved with the *nsave* command as a raw binary image. The information for the currently loaded image can be examined with the *info* command. The *info* command also shows the information for the currently saved image in FLASH from the *nsave* command.

Command syntax:

*nload*

Notes: See Section 3.3.4 for more information on this command.

#### 3.3.2.4.2 *nsave*

Saves an image stored in memory into FLASH that can be used with the *nload* command. The information for the currently loaded image can be examined with the *info* command. After this command, the *info* command shows the updated information for the saved image in FLASH.

Command syntax:

*nsave*

Notes: Only raw binary images that use a contiguous data area can be saved in FLASH. See Section 3.3.4 for more information on this command.

Persistent configuration: Updates data about the saved image in FLASH.

#### 3.3.2.4.3 *load*

This command loads an image from the terminal, SD or MMC card, or from NAND into memory. Supported image types include raw binary, and S-record files. Files loaded

from an SD or MMC card also require a filename in the 8.3 file format. Only files in the root directory of the SD or MMC card can be loaded.

Command syntax:

```
load [src <file>] [ty] <hex addr 1> <hex addr 2>
```

Examples:

```
load blk im1.bin raw 0x80000000 (Loads the im1.bin file from the root directory of the SD/MMC card into address 0x80000000 with execution address 0x80000000)
```

```
load term raw 0x80008000 0x8000A900 (Loads a raw binary image from the terminal at address 0x80008000 with execution address 0x8000A900)
```

```
load flash srec (Loads the image saved with nsave using an S-record format)
```

Notes: See Section 3.3.3 for more information on this command.

#### 3.3.2.4.4 *aboot*

Sets up the board to automatically load and boot an image from the terminal, SD or MMC card, or from FLASH.

Command syntax:

```
aboot [src <file>] [ty] <hex addr 1> <hex addr 2>
```

Examples:

```
aboot blk ldr.hex srec (Loads the S-record file ldr.hex from the root directory of the SD or MMC card and executes it)
```

```
aboot flash srec (Loads the SREC file stored in FLASH using the nsave command. Note that the nsave image was loaded and stored as a raw binary image).
```

Notes: See Section 3.3.5 for more information on this command.

Persistent configuration: Updates autoboot information in FLASH.

#### 3.3.2.4.5 *boot*

Boots the current autoboot image. The current setup configured with the *aboot* command will be loaded and started.

Command syntax:

```
boot
```

### 3.3.2.5 Other commands

#### 3.3.2.5.1 *update*

Updates the kickstart loader or the stage 1 loader from a loaded raw binary image.

Command syntax:

*Update <kick>*

Examples:

*update (updates the stage 1 loader with the loaded image)*

*update kick (updates the kickstart loader with the loaded image)*

Notes:

The stage 1 loader image should not exceed 240K. The kickstart loader should not exceed 15.5K.

Persistent configuration: Will not change persistent configuration. Updates to the stage 1 loader may change the saved data structure in the serial EEPROM automatically resetting the defaults.

### 3.3.2.5.2 memtst

Runs one or more memory tests on the specified range of memory. A specified range can be cached or uncached memory with a *width* of 1 (bytes), 2 (half-words), or 4 (words). Using a *test* value of *all* will run all the tests. A test will fail and return the failed address, actual value, and expected value on the first failed data occurrence.

Test values for the *test* argument are shown below:

0 – waling ‘1’ test

1 – walking ‘0’ test

2 – inverse address test

3 – Pattern (0xAA, 0x55, 0xA5, 0x5A) test

4 – Pattern (0x00, 0xFF) test

Command syntax:

*Memtst [hex add] [bytes] [width] [test]*

Examples:

*Memtst 0x80000000 102400 4 0 (runs test 0 on words at address with size)*

*Memtst 0x94000000 67108864 1 all (runs all tests on bytes at address with size 64M)*

## 3.3.3 Loading and executing files

### 3.3.3.1 Image types

The stage 1 loader supports 2 types of file: raw binary and S-record files. The *load* and *aboot* commands handle one of these file types based on the arguments used with those commands.

Raw binary files are absolute raw data that is loaded at an address. There is no extra data or structures in the file that identifies where the file is loaded or executed. Most build tools can generate a file of this type. Because these files don't have information on where they are loaded or executed, the load address and execution address need to be manually provided when the image is loaded. A binary file always loads in a contiguous data region using memory starting at its load address. Loading a binary image from FLASH and an SD/MMC card is error free, but loading an image of this type via the terminal may have errors.

S-Record files are text files formatted in the Motorola S-record format. These files consist of records of text that define an address and data. Another record type defines the execution address. S-records also provide checksums for each record, so any errors that may occur during a terminal load can be detected. S-records may or may not be loaded in a contiguous data range and are well suited for images with multiple load regions.

Regardless of the file type that is loaded, it is tagged as a raw image once in internal memory. Table 4 shows the supported file load options.

**Table 4 Support image load options**

File type load options		
	S-record	Raw binary
FLASH	Yes <sup>1</sup>	No <sup>3</sup>
Terminal	Yes <sup>2</sup>	Yes <sup>2</sup>
SD/MMC card	Yes	Yes

<sup>1</sup>Data stored in FLASH can only be a raw image. Data loaded from FLASH can be any file type.

<sup>2</sup>Files transferred to the board via the terminal are not guaranteed error free. Use an SD/MMC card for error free operation.

<sup>3</sup>Raw binary downloads of images saved in FLASH are not supported. If a raw binary image is loaded from FLASH using the *load* command, the load and execution in the *load* command arguments are not used. The saved values are used instead.

### 3.3.3.2 Loading an image

The *load* command can be used to load an image from FLASH, an SD or MMC card, or the terminal. The context and function of the command changes slightly depending on the load source and file type used. The command syntax is shown below:

```
load [src <file>] [ty] <hex addr 1> <hex addr 2>
```

#### 3.3.3.2.1 Image load sources

The source (src) can be the *term* (terminal), *blk* (clock device - SD or MMC card), or *flash*.

If the source is *blk*, a filename must be included after the *blk* source and before the file type *ty*. The filename is in an 8.3 format and must be located in the root directory of the

SD/MMC card. Long file names are not supported. The *ls* command can be used to list the files on the card.

Example: *load blk lcd.bin 0x80000000*

If the source is *flash*, then an image saved with *nsave* will attempt to be loaded.

Example: *load flash srec*

If the source is *term*, the command prompt will disappear and a message will appear requesting download via the terminal. The image should be sent over as a binary. Free terminal programs such as *teraterm* work well. The transfer can be cancelled by sending a *BREAK* from the terminal program to the stage 1 loader.

Example: *load term bin*

### 3.3.3.2 Image types and load command arguments

The image type (*ty*) can be *raw* (raw binary) or *srec*(S-record) file.

If the image type is *raw*, one or two hex addresses must be provided after the image type. The first address is required and specifies where to load the image in memory. Memory will be loaded as a contiguous data region from the specified load address in the first address field. The second address is optional and specifies the execution address. If the second address isn't provided, the load address will be used as the execution address.

Example: *load term raw 0x80000000 0x80001880*

If the image type is *srec*, the load and execution addresses are not used as they are already specified in the S-record file. S-records without an execution address record in the file will not load successfully. S-record files may be loaded in multiple, non-contiguous data regions.

Example: *load flash srec*

Loaded image information can be examined using the *info* command.

### 3.3.3.3 Executing a loaded file

Once an image is loaded, it can be executed using the *exec* command. The address for the start of execution can be examined with the *info* command. Execution at a specific address can be forced by providing an optional hex address after the *exec* command.

#### 3.3.3.3.1 Completion of an executed image

The executed image can exit and the command prompt will be restarted. For this to work correctly, the loaded image must return the system to its pre-run state.

### 3.3.4 Saving files in FLASH

#### 3.3.4.1 Files types that can be saved in FLASH

Only raw binary images can be stored in FLASH. Regardless of the file type loaded with the *load* command, the file is a raw image once in memory and can be saved in FLASH with the *nsave* command if the data for the loaded image is contiguous.

Along with the saved image in FLASH, information is stored in the serial EEPROM that identifies the saved FLASH image such as load and execution addresses and image size. This information is needed to retrieve the image later with the *nload* command.

##### 3.3.4.1.1 Contiguous vs. non-contiguous files

Files with multiple load regions are not contiguous in memory and can't be programmed into FLASH. S-record files may be non-contiguous. If a non-contiguous file needs to be stored in FLASH, it should be loaded as a raw binary using the *load* command and then saved in FLASH with *nsave*. The *load* command can be used with the FLASH source to load the file as its native file type.

Example of storing an S-record file in FLASH:

```
load term raw 0x80000000
```

*Send the S-record file to the stage 1 loader as a raw binary*

```
nsave
```

```
load flash srec
```

*The file is loaded from FLASH as an S-record. Notice the *nload* command is not used*

```
exec
```

*The loaded file is started*

#### 3.3.4.2 Updating the kickstart and stage 1 loaders in FLASH

The kickstart and stage 1 loader can be updated using the *update* command. To use the *update* command, first load either the kickstart or stage 1 loader/application image using the *load* command. Then use the *update kick* or *update* command to update the kickstart loader or the stage 1 application in FLASH.

### 3.3.5 Setting up autoboot

Autoboot allows an image on FLASH, an SD or MMC card, or loaded through the terminal to automatically load and execute. This is setup with the *aboot* command.

#### 3.3.5.1 Autoboot timeout and load failures

The autoboot capability will not start until the autoboot timeout has expired. This timeout is in seconds and is set with the *prompt* command. A minimum of 1 second is required. The timeout allows a key to be pressed on the terminal to override the autoboot sequence and get the monitor program prompt.

After autoboot timeout and if no key on the terminal has been pressed, an image load attempt will be made. If the load attempt fails, the monitor program prompt is displayed instead.

### 3.3.5.2 Auto-booting raw images from FLASH

Raw images saved with the *nsave* command can be autobooted. Note that a raw image is an image loaded with the *load <src> raw* command and may be a image file type of raw binary or S-record. If the file type is raw, the load and execution addresses used with the *about* command are not used – the saved FLASH image values are used instead.

### 3.3.6 Load, save, and boot examples

#### 3.3.6.1 Terminal->S-record->FLASH

An S-record file is loaded as a raw image from the terminal, saved in FLASH, and then autobooted from FLASH.

Step 1: Image is loaded as a raw image into SDRAM  
*load term raw 0x80000000*

Step 2: Image is saved into FLASH  
*nsave*

Step 3: Autoboot is setup to load an S-record from FLASH  
*about flash srec*

Step 4: Autoboot timeout is set to 1 second.  
*prompt lpc3250> 1*

On the next power cycle or reset, the S-record will be loaded from FLASH and execution will start automatically after 1 second.

#### 3.3.6.2 SD/MMC card->raw binary

A raw binary image stored in the root directory of an SD/MMC card is loaded and autobooted.

Step 1: Place image in the root directory of the SD/MMC card (example name: *img.bin*)

Step 2: Autoboot is setup to load a raw binary image from the SD/MMC card.  
*about blk img.bin raw 0x80000000*

Step 3: Autoboot timeout is set to 5 seconds.  
*prompt lpc3250> 5*

On the next power cycle or reset, the raw binary image will be loaded from the SD/MMC card and execution will start automatically after 5 seconds.

### 3.3.6.3 Terminal->raw binary->FLASH

A raw binary image is loaded from the terminal, saved in FLASH, and then autobooted from FLASH.

Step 1: Image is loaded as a raw image into SDRAM  
*load term raw 0x80000000 0x80001000*

Step 2: Image is saved into FLASH  
*nsave*

Step 3: Autoboot is setup to load a raw image from FLASH. The first address is required, but not used. The address used with the load command will be used instead when the autoboot occurs.  
*aboot flash raw 0x12345678*

Step 4: Autoboot timeout is set to 1 second.  
*prompt lpc3250> 1*

On the next power cycle or reset, the raw image will be loaded from FLASH and execution will start automatically after 1 second. Image is loaded at address 0x80000000 and execution starts at address 0x80001000.

## 4 Notes

### 4.1 WinCE and reserved block marking

If FLASH support is enabled in the WinCE build, WinCE may attempt to partition and format FLASH for its use. This will cause the kickstart and stage 1 loader to be erased, as well as any stored data in FLASH.

To prevent this from happening, the kickstart and stage1 blocks are marked as reserved in FLASH. A reserved block is determined by checking bit 1 of a page at offset 517. If the bit is 0, the block is reserved.

Data stored with *nsave*, *write*, and *nburn* do not tag blocks as reserved. The *nrsv* command should be used to mark reserved blocks. The *nrsv* command can only mark blocks as reserved. To remove a reserved block mark, the block must be erased or overwritten with the *nsave*, *write*, or *nburn* functions.

### 4.2 Known issues with S1L

#### 4.2.1 MMUENAB command sometimes hangs

Avoid using the *mmuenab* command, as it is unreliable when disabling the MMU.

#### 4.2.2 INFO shows image loaded in memory on ABOOT failure

If the autoboot setup fails to boot for some reason (for example, when autoboot is setup to boot from an SD card and an SD card is not inserted), the *info* command will show that an image has been loaded into memory. This issue only occurs when an autoboot occurs from an SD card and the SD card is not inserted or the file is not found on the SD card.

### 4.3 FLASH controller ECC support

The kickstart and stage 1 loader use both the SLC and MLC NAND controllers of the LPC3250 device. All handling performed with the MLC supports ECC in the hardware. SLC support uses software based ECC support.

The only command that uses the MLC is the *update* command. All other commands use the SLC. The kickstart and stage 1 applications are stored in FLASH using the MLC ECC algorithm, while the other NAND based operations use 1-bit ECC.

#### 4.3.1 ECC impact to FLASH commands

Some S1L FLASH commands (read, etc.) will generate an error if a sector is read with a bad ECC computation. This is normal and indicates an uncorrectable error in FLASH for the read data.