LogicLoader User's Manual

User's Manual for Logic's SOM Development Kits

REVISION HISTORY

REV	EDITOR	REVISION DESCRIPTION	APPROVAL	DATE
A	Bill O'Donnell	Release	B.O.D	01/21/03
B	Bruce Rovner	Release	B.R.	05/05/03
C	Bruce Rovner	Release	B.R.	07/24/03
D	Hans Rempel	Editing and Section 6 Update; LogicLoader Version 1.2.0	HAR	09/15/03
E	James Wicks	Document Format and Edit	JAW	09/30/03
F	Bruce Rovner James Wicks	Preliminary Release LogicLoader Version 1.4	B.R.	3/31/004
G	Bruce Rovner	Document Edit/ Test Commands Update LogicLoader Version 1.5/ MOT Pilot Release	JAW	6/08/04
H	Mike Aanenson James Wicks	Document Edit: Added SOM to Acronyms; Command Added: mem-copy	JMC	8/30/04
I	James Wicks	Attached SOM Cover page	JAW	10/14/04

Please check www.logicpd.com for the latest revision of this manual, errata's, and additional application notes.

Table of Contents

Table of Figures

LogicLoader™ BOOTLOADER / MONITOR

For SH / ARM / ColdFire Achitectures

The LogicLoaderTM (bootloader/monitor) provides the capability for loading both operating systems and applications. In addition, it provides a full suite of commands for configuring hardware platforms, in-field device management, hardware debug, manufacturing and test. LogicLoader is a key tool in reducing the time for development, manufacturing, and test. This results in an embedded product development cycle in **less time, less cost, less risk ... more innovation.**

Developing Products is as simple as

A B C

- A Application Development Kits
- B Board Support Packages
- C Card Engines

PRODUCT HIGHLIGHTS

- Ships standard on all System on Modules
- Programmable APIs
- Manufacturing & Test Capabilities
- Source code is available for purchase, please contact Logic Sales for more information.

- **LogicLoader Description**
  - Multiform support (SH / ARM / ColdFire)
  - Fully integrated TCP/IP stack with DHCP and TFTP support
  - Supports Compact Flash FAT file system
  - Network bootstrap support including setup and download using bootp
  - Customizable and user extendable
  - Source code available
- **Operating System Bootstrap**
  - Load an operating system (OS) from
    - Compact Flash
    - Resident Flash Array
    - Serial connection
    - Ethernet connection
  - Fully configure a hardware platform for the operating system
    - System on Module initialization
    - Link in custom software functions to initialize hardware before the OS starts
    - Power-on self test capability
  - Load multiple Operating Systems: Linux, Windows CE, etc. See available BSPs.
- **Programmable APIs**
- **In-Field Device Management**
  - Modify boot actions at run-time using LogicLoader"s configuration utilities
  - Remote device management eases debugging and upgrading
- **Hardware Debug**
  - Link in custom test functions to verify custom hardware
  - Use a familiar Unix-like interface for debugging the device
  - Ethernet based download & debug interface for Windows CE
- **Custom Applications**
  - Use LogicLoader to burn and jump to any custom embedded application
  - Link to Logic's libraries or write custom libraries
- **Download Formats**: SREC, ELF, and BIN
- **Manufacturing and Test**
  - Add in custom functional test software for your specific device needs
  - Take advantage of the fast Ethernet connectivity to reduce manufacturing test time
- **System Requirements**
  - Host PC running Windows 2000/ XP with Tera Term or equivalent terminal application

CUSTOMER SERVICE

Logic provides technical support for Application Development Kits. Various support packages are available; contact us for more information.

CONTACT

For more information on our Embedded Product Solutions, please contact Logic Sales at www.logicpd.com or 612.672.9495.

●2004 LOGIC PRODUCT DEVELOPMENT 411 N. WASHINGTON AVE SUITE 101 MINNEAPOLIS MN 55401 TEL: 612.672.9495 FAX: 612.672.9489 WWW.LOGICPD.COM

PN: 70000023 Rev E

1.2 Acronyms

API	Application Programming Interface
CPLD	Complex Programmable Logic Device
CF	CompactFlash®
DHCP	Dynamic Host Configuration Protocol
EEPROM	Electrically Erasable Programmable Read-Only Memory
ELF	Executable Linkable Format
FAT	File Allocation Table
GNU	GNU is not UNIX
I/O	Input/Output
IP	Intellectual Property
IP	Internet Protocol
JTAG	Joint Test Action Group
OS	Operating System
RAM	Random Access Memory
RISC	Reduced Instruction Set Computer
SOC	System On a Chip
SOM	System on Module – family name for Card Engine (SOM-Card Engine) and Fire Engine (SOM-Fire Engine)
TCP/IP	Transport Control Protocol/Internet Protocol
TFTP	Trivial File Transfer Protocol

1.3 Technical Specifications

Please refer to the component specifications and data sheets applicable to your SOM:

- ARM/SH/ColdFire IO Controller Specification
- ARM/SH/ColdFire Hardware Specification
- ARM/SH/ColdFire Processor Manual

1.4 LogicLoader Advantages

The LogicLoader program is a bootloader/firmware-monitor program developed by Logic Product Development. LogicLoader provides a command-rich shell as well as a bootstrap environment for a wide range of embedded operating systems such as Linux and Microsoft Windows CE. LogicLoader includes system initialization functions, system diagnostic functions, a Unix-like command line interface, and OS download and booting. LogicLoader is field upgradeable.

LogicLoader is the bootstrap and monitor environment that runs on Logic's development kits. It provides the following functions:

- Low level initialization
- Operating System (OS) recognition, loading, and booting from flash, CompactFlash, and serial interfaces
- Diagnostic utilities
- Flash memory utilities
- Monitor functions, including a Unix-like command line interface
- Support for image download via Ethernet or serial port
- Timer utilities

LogicLoader Features

- Unix-like shell command interface
- Debug information via serial port
- Product-ready bootloader/monitor
- Load operating system or user developed application program from Ethernet, serial port, flash, or CompactFlash interfaces
- Quickly port to new platforms

- Easy to customize
- Fully integrated TCP/IP stack with DHCP and ping utilities
- Network bootstrap support including setup and download using bootp and TFTP protocols
- Static IP address capable
- Boot-time script execution

Future versions of the LogicLoader will include

- More configuration options: control of default OS boot image, failsafe image loading, etc…

2 LogicLoader (LoLo™)

2.1 LoLo Overview

The LogicLoader (LoLo) is a bootloader/firmware-monitor program developed by Logic. LoLo is designed to initialize an embedded device, load and bootstrap an operating system, and provide a low-level firmware monitor with debugging functionality.

2.2 LoLo Basics

Most operating systems rely on an underlying bootloader to initialize a computer from its reset condition. In general, operating systems are designed with the assumption that the system will be in a specific pre-defined state before the operating system is started. Some example assumptions might be that system RAM has been initialized and cleared, processor interrupts are disabled, and a timer has been initialized to provide a system tick for the OS. The LogicLoader program initializes Logic Product Development's SOM platforms and prepares them for use by an operating system.

Another basic functionality of LoLo is the capability to upgrade device software after deployment. This "in-field upgrade ability" requires a bootloader program which is capable of loading an operating system from various sources as well as committing loaded images to non-volatile memory. The LogicLoader implements this by giving the system the ability to boot system software from flash memory, a CompactFlash storage card, or even on another computer attached to the system's serial port. The LogicLoader also has the ability to upgrade an existing operating system residing in system flash.

A major reason for the development of LoLo was the need for an OS and processor independent bootloader that can interface with a variety of hardware transports. LoLo is designed to build with any embedded development environment. The GNU tools distributed by Logic are cross-platform capable.

2.3 Debugging Advantages with LoLo

The LogicLoader implements a feature-rich firmware monitor. Included with LoLo is the LogicLoader shell, also known as "losh." Losh is a command interpreter with advanced features such as command recall and command-line editing. Losh includes many commands designed specifically to help software and hardware engineers debug low-level interfaces. Some examples include the 'x' and 'w' commands that allow formatted data to be read from and written to arbitrary memory locations. Other commands run specific tests designed to verify Logic's SOM hardware platforms. Refer to the appropriate section of this manual for a complete description of available commands.

Developers may code their own test programs using the provided GNU development toolchain and use the LogicLoader to load and run their software. This provides the ability to verify and debug hardware interfaces without the overhead of building, downloading, and running large operating system images. By using LoLo's application programming interface (API), developers will have access to a powerful UNIX-like interface including formatted I/O, network sockets, graphics and windowing routines, and an audio interface.

2.4 Manufacturing Advantages with LoLo

The LogicLoader can be used with a desktop software utility to load a device's system software on the manufacturing line. This utility may be customized to suit your desired transfer mechanism and additional needs. LoLo may also be augmented with functional test software to completely verify a device before it leaves the manufacturing line. For example, LoLo may be used to launch a device's final functional test at the end of a manufacturing line. It may then be used to load the device's final software image before packaging. Contact Logic for more information on using LoLo to streamline manufacturing.

2.5 Ongoing Development with LoLo

Logic is constantly improving the LogicLoader. New features are frequently being defined and implemented to help our customers develop their products faster and easier. Continue to look for updates on the Logic website. If you have a specific feature request, contact Logic for further information.

3 Block Zero Loader™ (BoLo™)

3.1 BoLo Overview

The block zero loader (BoLo) is a fall-back feature on Logic's Zoom™ Development Kits. BoLo is a stripped down version of LoLo. It is stored in the first block of flash (block zero) resident on most SOM shipped with Zoom Development Kits.

3.2 BoLo Basics

BoLo is a safety precaution for developers using the Zoom Development Kits. Upon boot-up, most SOM's begin executing code found in block zero of their resident flash array. If the code in flash block zero becomes corrupted, the SOM becomes useless for software development without employing special development tools such as a JTAG debugger.

In an effort to protect our customers from accidentally corrupting flash block zero, both the LogicLoader (LoLo) and the Block Zero Loader (BoLo) verify user-downloaded images that are destined for flash. If an image is downloaded to the device that would overlap flash block zero, confirmation from the user is required before proceeding. This prevents users of the Zoom Development Kits from accidentally corrupting flash block zero. It also serves to warn users of the potential ramifications of burning any code into flash block zero which is not designed to bring the system out of reset.

3.3 Using BoLo

Under most operating circumstances, users of Logic's Zoom Development Kits will never interact with BoLo. BoLo is designed to look for escape cues or a boot script and, if not seen, jump to code located in block one of the SOM's system flash. Zoom Development Kits ship with BoLo loaded into block zero and LoLo loaded into block one of the SOM's system flash. In this configuration, BoLo will launch LoLo at boot time unless: 1) the user interrupts BoLo, or 2) a BoLo script is found in the serial EEPROM that does not return. LoLo will look for a LoLo script in the serial EEPROM to run-- if one is not found, or the script returns after it executes, the user will be presented with the LogicLoader welcome screen and input prompt. Please refer to *Figure 3.1: BoLo/LoLo Interaction and Boot Sequence.*

If the code in block one of the SOM's flash array becomes corrupted or unusable, it is still possible to update the flash image by entering into BoLo at boot-time. Entering into BoLo is accomplished by booting the Zoom Development Kit while continuously transmitting the letter 'q' to the kit's serial port.

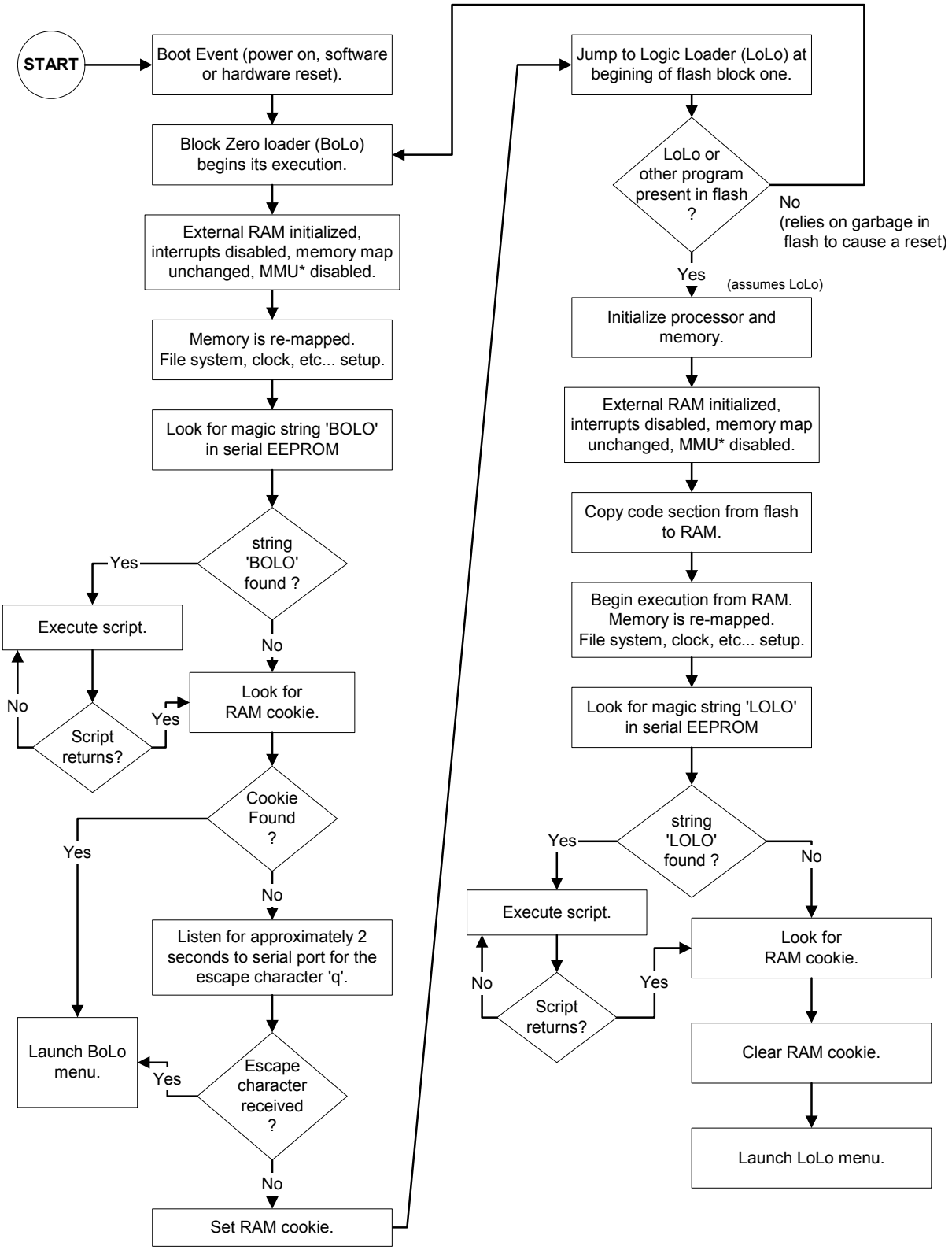
Before BoLo jumps to code in flash block one, it checks the serial EEPROM for a valid BoLo Script, then it looks to the serial port for approximately two seconds. If the letter 'q' is received during this period of time, BoLo will not jump to flash block one. Instead, BoLo will present the user with a welcome screen and a menu similar to LoLo's. This will allow the user to download new code to the device as well as give the capability to read and write device memory mapped locations. This behavior is consistent with BoLo's design as a safety precaution for users of the Zoom kits.

Without BoLo, users may find themselves in the position of having to use a JTAG debugging tool to recover the system from a failed LoLo/flash block one update. BoLo presents the minimum level of functionality required to recover a system after an image download or update has failed.

BoLo also allows users who do not require the download, debugging, and user interface options that LoLo provides to replace LoLo with their own application. As the user application evolves, it may also be updated by BoLo.

IMPORTANT NOTE: To exit BoLo, reset the system or use the 'jump' or 'exec' command to launch another program.

Figure 3.1: BoLo/LoLo Interaction and Boot Sequence



*Not all SOC's have an MMU.

4 The LogicLoader Shell (losh™)

4.1 Losh Overview

A major feature of the LogicLoader program is its Unix-like command shell: losh. Losh is patterned after the shell commonly found on Unix-like workstations.

Developers familiar with a Unix-like command line interface should find the losh implementation familiar and easy to work with. Many of losh's commands are patterned after their Unix counterparts and share the same syntax.

4.2 Losh Basics

As in Unix or Linux, losh employs the idea of a standard output stream (stdout). By default, stdout refers to a SOM's debug serial port. The output of any command that displays information to stdout (i.e. the 'cat' command) can be viewed using the terminal emulation program connected to the SOM's debug serial port. Likewise, the standard input stream (stdin) by default also refers to the SOM's debug serial port. In the future the ability to re-route stdout to a video screen, file, or network connection may be implemented.

The LogicLoader Shell includes a virtual file system that uses standard Unix path names. The highest-level (or root) directory is designated by the identifier '/'. A special sub-directory of the root with the name 'dev' is used to enumerate and interact with system's various peripherals and their associated device drivers.

4.3 Using Losh

The LogicLoader Shell includes a command history feature. This provides users with a convenient way to repeat commands. Using the up and down arrows, a user may scroll through a list of previously executed commands. When a desired command is displayed, press the return key to repeat the command.

Losh has a basic command line editing feature. This feature is like the command line editing feature in Unix or DOS.

Losh includes a user help feature through the 'help' command. Typing 'help' followed by any command name at the losh prompt will display the command's syntax, usage, and an example. This may be especially helpful to users who are just becoming familiar with the LogicLoader Shell.

Commands may be run in the background by adding a ' &' suffix.

5 Losh Commands

In this section, losh commands are listed with any required or optional arguments. Arguments required by a command are noted inside angle brackets '< >'. Arguments optional to the command are designated by square brackets '[ ]'. For example, the 'load' command requires an argument that specifies the type of file to load, but optionally, a user may also specify an input stream or filename. The command's syntax is documented as: load <type> [source]. In most cases, optional arguments are filled with default values if not specified by the user. For example, if a user does not specify the 'source' argument to the load command, the load is assumed to come from the standard input stream (stdin).

Figure 5.1: Losh User Commands*

beep	**jump**	test-cf
bench-mark	kill	test-eeprom
bitmap	**load**	test-flash
bootme	**ls**	test-mem
burn	md5sum	test-net
cache-flush	**mem-cmp**	test-reg
cache-off	**mem-fill**	test-rtc
cache-on	**mem-copy**	test-screen
cat	**mount**	test-touch
cd	paint	test-usb
codec	ping	tsleep
date	play-wav	touch-cal
draw-test	ps	touch-setcal
echo	**pwd**	**tlb-flush**
erase	**remap**	**unset**
exec	scr-getparam	**update**
funtest	scr-setparam	video-clear
help	**set**	video-close
hd	sleep	video-open
ifconfig	slide-show	video-set-default
ifmac	**source**	**w**
info	test-audio	**x**

*Note: commands are case sensitive

The commands listed in *Figure 5.1* above can be grouped into 7 basic categories:

- General commands
- Test commands
- File system commands
- Directory commands
- Multimedia commands
- Thread commands
- Network commands

The Losh command set may vary according to the different features and requirements of the different engines. You may view all of the available commands for your engine by entering 'help all' at the losh prompt. Typing the entry 'help' at the losh prompt will print a listing of the available sub-menus. The sub-menu listings are intended as a prompt to the user when needed. In LoLo, all commands in all categories that have been implemented for your engine are always available from the losh prompt.

The commands available in BoLo are written in **bold**, above.

5.1 General Commands

5.1.1 burn

Usage: **burn [device]**

Examples:
- burn
- burn /dev/flash0

This command programs a binary image that has been loaded, using the 'load' command, into a flash device [device]. The loaded image must have been linked with a start address falling within the flash device's address range.

The 'burn' command erases any of the blocks that fall inside the image boundaries before programming the new image.

The 'burn' command will use information gathered by the 'load' command from within the binary image file to decide where in flash the program should be stored. When [device] is not specified the default flash device (usually the boot device) is used. As a precautionary measure, this command will not re-write the device's flash block zero unless manually confirmed to do so.

IMPORTANT NOTE: Programming flash block zero with incorrect data can make the device unbootable. See the discussion of the Block Zero Loader in Section 3, above, for more information concerning flash block zero.

5.1.2 date

Usage: **date**

Example:
- date

This command displays the internal count of days, hours, minutes, and seconds elapsed since the last system reset.

IMPORTANT NOTE: If the 'date' value stays at 0, and LoLo is running, the JTAG jumpers may be in the wrong position for your current configuration. Please check the Zoom Development Kit User's Manual for correct JTAG jumper positioning.

5.1.3 erase

Usage: **erase <start address> <length> [device]**

Examples:
- erase 0x400c0000 1024
- erase 0x400c0000 1024 /dev/flash11

This command erases flash device [device] from <start address> for <length> bytes.

This utility is used to clear data burned into a flash device. The parameter [device] is optional, but if specified, it must be a flash memory device. The 'erase' command will prompt the user before erasing flash block zero.

IMPORTANT NOTE: Erasing is performed in block-sized units, usually 256k bytes. At least one block will be erased when this command is used.

5.1.4 exec

Usage: **exec [address -] [kernel command line]**

Examples:
- exec
- exec 0x400c0000 -
- exec 0x400c8000 - root=nfs
- exec root=nfs

The 'exec' command disables cache & interrupts, then jumps to a loaded OS, or to [address]. In order to use this command you must first load an image or specify an address. Note that when the [address] parameter is used, the address must be followed by a '-' even when the [kernel command line] parameter is empty.

The 'exec' command is similar to the 'jump' command, except that before jumping it disables the MMU, cache, and interrupts. On SOM's that have an MMU, it is necessary to specify a valid physical address when using the [address] option. Also, 'exec' can optionally pass a command line to the program.

5.1.5 help

Usage: **help <test|file|dir|video|net|thread|all|<cmd name>>**

Examples:
- help all
- help dir

The 'help' command is used to obtain help on other losh commands. Typing 'help' alone produces a list of command categories.

Typing 'help cmd_category' prompts losh to list the commands from that category along with a brief description.

Typing 'help cmd_name' prompts losh to print the help available for the requested command.

5.1.6 info

Usage: **info <version|arch|mem|net|cpu|intr|config>**

Examples:
- info version
- info net

The 'info' command displays important information about the hardware and software included on the SOM as shown in the table below.

Figure 5:2: 'info' Argument Description

info argument	Description
arch	prints information about major subsystem components external to the SOC on the SOM (i.e. CPLD, graphics chip)
config	prints information about the detected configuration for hardware and software Note: the amount of 'usable' memory may be less than the amount of 'detected' installed memory. Please see appropriate engine memory map addendum document for specific memory map details.
cpu	prints information about virtual memory and the current state of the cpu
intr	prints information about each interrupt
mem	prints important memory locations and heap statistics
net	prints several network statistics
version	prints the BoLo/LoLo version and build info, and the CPLD revision number

5.1.7 jump

Usage: **jump [address]**

Examples:
- jump
- jump 0x400c0000

The 'jump' command jumps to a loaded program or [address]. LoLo will continue executing interrupt handlers and other threads in the background provided the downloaded application does not disturb its RAM space or clobber its interrupt handling. When the 'load' command has been used to download a program in a recognized format, the program's starting address is saved and becomes the default [address] argument for the 'jump' command. This is the only case where the 'jump' command may be used without any parameters. If there is no recent successful download, the [address] parameter is required. 'jump' functions in the same way as a function call with no parameters.

5.1.8 load

Usage: **load <type> [source]**
<type> /tftp/<server:filename>
raw <dest addr> <length> [source]
<type> -dhcp

Example:
- load elf
- load elf /cf_card/IMAGE_FILE
- load elf /tftp/my_server:my_file
- load srec -dhcp

The 'load' command is used to load a binary image into memory. The 'load' command may be used to download Executable Linkable Format (ELF), Motorola S-record (S-rec), raw, or Microsoft Windows CE .bin image formatted binary files to the device. The raw formatted files require a destination address and a length argument. The default 'source' is stdin. The binary file may be read from any of the byte-stream devices supported by LogicLoader. This currently includes the debug serial port (stdin), a CompactFlash card, and the network. The 'load' command uses

address information contained in the binary image file to determine if the file is destined to be stored in flash or RAM memory.

If a binary file is destined for RAM, the 'load' command will place the image directly into system RAM and arrange the sections as specified by the file headers. This process is done regardless of what is currently loaded in system RAM. Programs destined for RAM should be linked so that images do not interfere with the operation of the LogicLoader. LogicLoader will protect itself. The sample applications provided with the Zoom Development Kits take into account the location of the LogicLoader's RAM space. Developers may use these applications, along with the SOM memory map addendum, as a reference for building their own applications.

If a program is destined for flash, the 'load' command will download the binary file into a temporary buffer in the system RAM. After the entire file has been received and any verification performed, the 'burn' command can be used to save the image into system flash. The sample application provided with the Zoom Development Kits is an example of how a program may be linked so that it can be stored in flash and relocate itself from flash to RAM upon start of execution. Developers may refer to this program as a guide to build and link their own applications.

Flash images are downloaded into RAM space beyond the end of LoLo's execution area. This is a temporary buffer area until the 'burn' command is used to program this data from RAM to flash. The size of this buffer limits the size of data that can be downloaded and burned at one time. Check the SOM's total RAM and the run-time size of LoLo to determine this limit.

Flash S-record images are stored in this same unused RAM area based on their offset from the base of flash. This creates a limitation of only being able to write to a window of flash that corresponds to available RAM. RAM destined S-record images, however, are treated the same as RAM-based elf loads: they are written to exactly the address that is specified in the S-record.

The 'load' command runs an md5sum on images that it loads from the serial port to make sure that no serial corruption has occurred. The checksum is run on the loaded part of the image, not the entire file or elf headers. The md5sum of an elf file can be calculated from the development machine with this process:

1. extract the 'binary' portion of an elf file: objcopy -O binary somefile.elf tmp.raw
   Note: must use the correct objcopy. i.e. arm-elf-objcopy, or sh-linux-objcopy
2. run md5sum on the raw portion: md5sum tmp.raw
3. verify the resulting sum against what is reported by 'load elf'

5.1.9 mem-cmp

Usage: **mem-cmp <a> <b> <len>**

Example:
- mem-cmp 0x0 0x1000 0x1000

The 'mem-cmp' compares memory from <a> to <b> for <len>.

The 'mem-cmp' command prints "0 differences found" if the two memory segments are identical. Otherwise, this command prints the first difference that it finds and then exits.

5.1.10 mem-fill

Usage: **mem-fill <addr> <count> <value> [/bhw]**

Example:
- mem-fill 0xc0000 0x1000 0xabcd /h

This command fills memory at <addr> for <count> with <value> in sizes of [bhw].

5.1.11 mem-copy

Usage: **mem-**copy **<addr1> <addr2> <count> [/bhw]**

Example:
- mem-copy 0xc0000 0xd0000 0x100 /h

This command copies direct accessible memory at <addr1> to direct write-able memory at <addr2> for <count> in sizes of [bhw].

The mem-copy command will not work when writing to indirect write-able areas of memory. (e.g. flash)

5.1.12 set

Usage: **set sym val**

Example:
- set SYMBOL 0x123

The 'set' command sets a symbolic value, with no arguments. This command also prints the symbol table.

5.1.13 source

Usage: **source <filename>**

Example:
- source /cf_card/STARTUP

The 'source' command executes a series of commands stored in <filename> providing scripting functionality to LoLo. Note that the file system '/cf_card' must have been mounted before executing the command given in the example. Refer to the 'mount' command for details.

5.1.14 unset

Usage: **unset <sym>**

Example:
- unset SYMBOL

The 'unset' command removes a symbolic value from the symbol table.

5.1.15 update

Usage: **update [filename]**

Examples:
- update
- update /tftp/10.10.77.5:75401-10.upd

The 'update' command is used to update the firmware on a Logic SOM.

Update files may be obtained from the Logic Product Development downloads site.

5.1.16 w

Usage: **w [/[bhw]] <addr> <data>**

Examples:
- w /w 0x60000000 0x12345678
- w 0x60000000 0x12345678
- w /b 0x60000000 255

The 'w' command is used to write memory [of specified width] at <address> with <data>. This is a simple poke command.

The width of the access is determined by the [bhw] parameter, where 'b' is for a byte-wide access, 'h' for half-word (16 bits) and 'w' for word (32-bit) access. The default width, if [bhw] is not specified, is 'w'.

The <address> and <data> parameters may be specified in decimal or hexadecimal. Hexadecimal values are designated by a prefix of '0x'.

Improperly aligned accesses will fail, and the user will be notified.

5.1.17 x

Usage: **x [/[bhw][odux]] <addr> [len]**

Example:
- x /h 0x40000000 64

The 'x' command is used to examine memory with [width][format] at an addr for a [len].

Like the 'w' command, the access size may be specified, with the default being 'w'. An output format may also be specified. The format argument must be one of [odux], where 'o' is octal, 'd' is decimal, 'u' is unsigned decimal, and 'x' is hexadecimal. If a format argument is not specified, the default output is hexadecimal.

The [length] parameter specifies the number of bytes, half-words, or words in accordance with the size requested.

If both a width and a format argument are specified, no space should be placed between them.

Improperly aligned accesses will be adjusted to include the requested starting address.

5.2 Test Commands

5.2.1 beep

Usage: **beep [level [milliseconds [playback rate]]]**

Examples:
- beep
- beep 0xc0000 2000 11025

The beep command is intended for use as a quick test of the audio output.

This command will make 3 beep sounds in succession on both channels at [level] for [milliseconds] using the sampling rate specified by [playback rate]. The level can vary from 0 to 0xffff and the default value is full scale. The default number of milliseconds is 1000. The default sampling rate is 22050. Please note: certain SOM's do not support all sampling rates.

5.2.2 bench-mark

Usage: **bench-mark [increments [reps]]**

Example:
- bench-mark
- bench-mark 10000 5

The 'bench-mark' command timestamps a loop of [increments] [reps] times. The default is 10 reps of 1000000 increments

This command runs a simple benchmarking program that times the execution of a short "for" loop. The timer has millisecond resolution. This command is meant to give a very general idea of relative execution times; it is not meant as a thorough processor benchmark.

The command executes the following code:

```
for ( i = 0; i < reps; ++i) {
    Get start time.
    for (j = 0; j < increments; ++j)
      ;
    Get end time.
}
```

The time taken to run the inner loop of code is measured and a running record of the maximum, minimum, and average execution times are kept and then displayed.

For example, perform the following sequence of commands:

```
losh> cache-off
losh> bench-mark
losh> cache-on
losh> bench-mark
```

Compare the output of both of the 'bench-mark' commands. Note that not all SOM's have cache memory.

5.2.3 cache-flush

Usage: **cache-flush**

Example:
- cache-flush

This command is used to flush the processor's cache.

Not all SOM's have cache memory

5.2.4 cache-off

Usage: **cache-off**

Example:
- cache-off

This command disables the processor's cache.

Not all SOM's have cache memory.

5.2.5 cache-on

Usage: **cache-on**

Example:
- cache-off

This command enables the processor's cache.

Not all SOM's have cache memory.

5.2.6 codec

Usage: **codec <r|p|g> <time in milliseconds> [v0 - v0xffff volume] [sample rate] [bits of resolution] [#nnn repeat count]**

Example:
- codec rpg 1000 44k v0xdfff #2

This command is used to record, play, and graph audio to/from the CODEC. This command may be used to verify operation of the CODEC circuit and driver both audibly and visually. When the graphing option is used, the entire sample will be displayed so you will probably want to record a short sample. A sample may be recorded once and then played back several times to test external audio components. A CODEC session with a large number of repeat counts may be used to monitor the affects of adjusting external signal conditioning components.

Please note: certain SOM's do not support all sample rates or bits of resolution.

5.2.7 funtest

Usage: **funtest <dash number>**

Example:
- funtest 25

The <dash number> is the last 4 digits of the model number, excluding any revision letters and numbers. For example, if your model number is 80000126 – 0025 + B01, the dash number would be 0025, or just 25.

This command runs a series of functional tests on the Logic SOM's peripherals. The 'funtest' command is primarily used to test during manufacturing. It requires some user attention to verify operation of I/O devices such as the video driver, the touch screen, and the audio CODEC. Most of the commands are available separately as test-*device.*

The tests are performed in the following order:

Peripheral test: The SOM's detected peripherals and memory sizes are verified according to the <dash number>.

Serial test: The SOM uses the debug serial port to output the LogicLoader. No functional test is done at this point.

Flash: The presence of /dev/flash0 is verified.

RTC: The real time clock is verified against the tick timer to see that it is operating properly.

LCD: The pattern used by 'draw-test' is displayed and the user is asked to verify that is displayed correctly.

Touch: A red square is drawn in the top-left corner of the display. A prompt is given to touch the center of the red square. The test allows 10 seconds to complete this task before timing out. After each successful touch, the red square moves to the next corner (clockwise) until all four corners have been tested. Do not be surprised if the test passes even when the error seems very large for the sensed points. The test is meant to verify the operation of the touch circuitry on the board and therefore must allow for a wide range of raw touch screen data values in order to accommodate the manufacturing tolerances encountered with different touch screens.

USB: For SOM's which support USB device functionality, the user is asked to plug the USB device connector into a USB host.

CompactFlash: An attempt is made to mount a CompactFlash card inserted into the Zoom Development Kit's CompactFlash socket.

Audio: This test requires that a set of external amplified speakers be plugged in to the app kit's audio output jack. The CODEC is initialized and the test attempts to play the file named "FUNTEST.WAV" which it looks for in the root directory of the CompactFlash card mounted during the CompactFlash test. If the wav file is not found, it plays three buzzing noises. The test then asks the user to verify whether or not the sound was heard.

Ethernet: This test requires that a loop back cable be plugged into the ethernet jack. The Revision ID of the Ethernet controller chip is checked to verify that the device can be read and written to. The test then runs a short loop back test.

EEPROM: A read/write/erase verification of the EEPROM is performed.

RAM: The data and address busses running to system RAM after LOLO are tested.

5.2.8 paint

Usage: **paint**

Example:
- paint

This command allows the user to verify the color accuracy and touch-screen calibration by drawing on the screen. In order to use this command, you must first perform the 'video-open' command.

Touching the blue palette causes the paint program to exit.

5.2.9 play-wav

Usage: **play-wav <file> [level]**

Example:
- play-wav /cf_card/MYFILE.WAV 0xb000

This command is used to play a WAVE file. The [level] option may be used to adjust the volume from 0x0000 to 0xffff full scale.

5.2.10 remap

Usage: **remap <phys> <virt> <length> [c]**

Example:
- remap 0x72000000 0x72000000 0x200000

The 'remap' command adds virtual memory to the mmu's page table in one megabyte chunks. The results can be verified with the 'info cpu' command. The primary use of this command is to enable access to an area of memory where the user has placed custom hardware that LoLo is not aware of.

The [c] option may be used to make a new or existing area of memory cachable. This option may be desirable when loading a user application into an un-cached area of SDRAM.

Not all SOM's support virtual memory. And not all SOM's with virtual memory require this command.

5.2.11 test-audio

Usage: **test-audio [wav_file]**

Example:
- test-audio /cf_card/TEST.WAV

The 'test-audio' command tests audio with an optional wav file. If you do not specify a wav file or the specified wav file cannot be found, the SOM plays three beeps in sequence by default. You must have an amplified speaker plugged in with the volume turned up in order to hear the test sound. You will be asked if you heard the test sound.

5.2.12 test-cf

Usage: **test-cf <mount-point>**

Example:
- test-cf /cf_card

This command is used to verify that a CompactFlash card can be mounted.

5.2.13 test-eeprom

Usage: **test-eeprom**

Example:
- test-eeprom

The 'test-eeprom' command verifies that the onboard serial eeprom is working properly.

5.2.14 test-flash

Usage: **test-flash**

Example:
- test-flash

This command verifies that the onboard flash has been detected.

5.2.15 test-mem

Usage: **test-mem [v|r|t]**

Example:
- test-mem
- test-mem v
- test-mem t
- test-mem r

Type: 'test-mem v' for verbose, 'test-mem t' for threaded, and 'test-mem r' to print results.

The 'test-mem' command runs a memory test that: 1) fills the memory with a known pattern, 2) checks each location and inverts it, and 3) checks each location for the inverted pattern.

The 'test-mem v' command runs the test in verbose mode, reporting the progress as it goes, and then prints the results.

The 'test-mem t' command runs the test silently as a background thread and requires that you run 'test-mem r' to retrieve the results.

The 'test-mem r' prints the memtest results (if 'test-mem t' has been issued).

5.2.16 test-net

Usage: **test-net [b | x]**

Example:
- test-net
- test-net b
- test-net x

This command runs an Ethernet loopback test, and therefore requires that an Ethernet loopback cable be connected to the Ethernet port.

The [b]rief option attempts to auto-negotiate with 91C111 and then prints whether or not three packets were sent and at least one was received.

The e[x]tended option performs the brief test, as well as attempting to send and receive 10000000 bytes. The test then reports timing and error information.

5.2.17 test-reg

Usage: **test-reg <reg addr> <reg size in bytes> <bitmask of valid r/w bits>**

Example:
- test-reg 0xc0000000 1 0xff

This command will write & read a pattern into a register or memory location 1M times in order to look for errors related to bus timing. The bitmask may be used to mask off any register bits that are not read/write bits.

This command may be used to verify the proper connection and operation of an external device. For example, if a developer has connected an external 16 bit wide r/w device at address 'A' the developer issues the command: test-reg 'A' 2 0xfff to verify that the SOM is able to communicate with the device properly. The developer may want to run other LoLo application commands in the

background to verify that the new device can be accessed while other SOM peripherals are being utilized simultaneously.

5.2.18 test-rtc

Usage: **test-rtc**

Example:
- test-rtc

This command must be issued twice in order to test the real time clock. The first instance starts the rtc test and the second instance stops the rtc test and reports the results.

5.2.19 test-screen

Usage: **test-screen [t]**

Example:
- test-screen
- test-screen t

The 'test-screen' command is used to verify that the 'draw-test' image, described in detail below, was observed on a chosen screen. The [t] option is intended for using the SOM as a standalone lcd screen test station.

At the conclusion of the 'test-screen' command, LogicLoader will print a result of 'pass' or 'fail.'

5.2.20 test-touch

Usage: **test-touch**

Example:
- test-touch

This command allows the user to verify the operation of the touch screen circuit and driver on the SOM. This test displays a sequence of red boxes in each quadrant of the screen. After each box is displayed, the tester is asked to touch the center of each box, and the test will respond by drawing a yellow box centered on the coordinates that 'test-touch' measured based on a nominal set of calibration data. The test passes when the measured coordinates fall within the expected tolerance for touch screen manufacturing. Because of the wide tolerance across touch screens, a large error can still produce a passing result.

In order to use this command, you must first perform the 'video-open' command.

5.2.21 test-usb

Usage: **test-usb**

Example:
- test-usb

The 'test-usb' command is used to verify that the usb client can detect that a cable was inserted.

Not all SOM's support usb.

5.2.22 tlb-flush

Usage: **tlb-flush**

Example:
- tlb-flush

This command flushes the translation look-aside buffer.

Not all SOM's utilize a translation look-aside buffer.

5.2.23 touch-cal

Usage: touch-cal

Example:
- touch-cal

This command is used to calibrate the touch screen.

This command displays a sequence of three black crosshairs for the user to touch at strategic locations on the screen in order that a calibration matrix can be generated to transform raw A/D data into screen coordinates. Those coordinates are then verified when the user touches a sequence of blue, red, then blue crosshairs at the center of the screen. The transformation used is currently a simple slope/offset calculation.

After the calibration is complete, the calibration data is printed to the console and then 'touch-cal' automatically launches the 'paint' command. (Recall that touching the blue palette causes the paint program to exit.)

5.2.24 touch-setcal

Usage: **touch-setcal < x0,y0 x1,y1 x2,y2 screen#>**

Example:
- touch-setcal 176,728 494,216 728,471 7

This command sets the calibration of the touch screen using the calibration data produced by the touch-cal command. When used in a boot time script, this command allows a user to avoid having to recalibrate the touch screen each time LoLo starts..

5.3 File System Commands

5.3.1 cat

Usage: **cat <filename>**

Examples:
- cat foo.c
- cat /dev/serial_eeprom

This command opens a file and sends the contents to the console in its raw binary form.

The 'cat' command is intended for use on text files. Using the cat command to dump binary files may result in the need to reset the terminal.

5.3.2 echo

Usage: **echo <string> [filename [offset]]**

Examples:
- echo text
- echo text /dev/serial_7727_scif
- echo "LOLOecho hello; exit\n" /dev/serial_eeprom

This command will echo a string to stdout or write to [filename].

The 'echo' command is intended for use in script files to send messages to stdout or to files. It is currently not possible to embed double-quotes into double-quoted strings.

5.3.3 hd

Usage: **hd <filename> [len [offset]]**

Examples:
- hd foo.elf
- hd /dev/flash11

The 'hd' command displays the contents of a file to stdout in two-column hexadecimal/ascii representation.

5.3.4 ls

Usage: **ls [dir]**

Examples:
- ls
- ls /dev

The 'ls' command lists the contents of the current directory or the directory named by [dir] and formats the output into three columns: a flag with the type of file, the file name, and the file size.

The flags are: R - read only, H - hidden, S - system file, D - directory, r - reserved. These are loosely based on DOS file attributes.

5.3.5 md5sum

Usage: **md5sum <filename> [read-size]**

Example:
- md5sum /cf_card/FILE.TXT

This command calculates the md5 checksum on a file. The read-size parameter can be used to adjust the size of chunks that the md5sum calculation is performed on, but should not affect the final result.

5.4 Directory Commands

5.4.1 cd

Usage: **cd <directory>**

Examples:
- cd /dev

The cd command changes the working directory.

5.4.2 mount

Usage: **mount <fstype> [drive addr] <mountpoint>**

Examples:
- mount fatfs /cf_card

This command mounts a file system of type <fstype> located on a device [drive addr] to a local file system at point <mountpoint>. The default device is CompactFlash. Supported file systems include the FAT and FAT32 file systems.

The LogicLoader contains support for booting from the CompactFlash interface on the SOM or an external IDE drive. This command makes that interface available to other commands through the file system.

After mounting a file system it will appear in the file system tree (viewed with the 'ls' command), and any of the normal file commands can be issued.

For example, if a CompactFlash card formatted with the FAT file system is placed into the Zoom Development Kit's CompactFlash slot and the example 'mount' command from above is issued, a sub-directory named 'cf_card' will be created under the root directory. If the 'ls' command is performed in the root directory, the subdirectory '/cf_card' should be visible.

The 'cd' command can be used to enter the CompactFlash card's file system ('cd /cf_card'). All other commands are available to act on any files located on the CompactFlash card as well. For example, if a file named 'foo.txt' is stored on the card, the command 'cat /cf_card/FOO.TXT' will display the contents of the file.

If an ELF, BIN, RAW, or S-record image is stored on the CompactFlash card, that image may be loaded into memory using the 'load' command.

5.4.3 pwd

Usage: **pwd**

Examples:
- pwd

The 'pwd' command prints the present working directory.

5.5 Multimedia Commands

5.5.1 bitmap

Usage: **bitmap <file_name> [tl_x,tl_y [br_x,br_y]]**

Examples:
- bitmap /cf_card/TEST_FILE.BMP
- bitmap /cf_card/TEST_FILE.BMP 0,0 640,480

The 'bitmap' command draws a bitmap on the screen. Only Windows expanded device independent bitmaps are supported. The bitmap should be a standard 8 or 24 bpp with no compression. The [tl_x, tl_y] and [br_x, br_y] parameters optionally set the top-left x, top-left y, bottom-right x, and bottom-right y coordinates to specify the bounds of the bitmap on the screen. These comma-separated parameters refer to an origin at the top-left of the screen. Specifying a bitmap file 0,0 640,480 will display a bitmap which covers an entire 640x480 screen. The default is to show as much of the bitmap as the screen will allow. The 'video-open' command determines the default display and must be issued prior to using this command. Refer to the 'video-open' command for a list of supported displays.

5.5.2 draw-test

Usage: **draw-test**

Example:
- draw-test

This command draws a test pattern on the default display: most noticeably, red, green, and blue colors increase in intensity from left to right in horizontal gradient bands across the screen. On the bottom of the screen, three black and white rectangles enclose diagonal, vertical, and horizontal stipple patterns. The entire screen is framed within three one pixel wide rectangles of white, red, and blue.

In order to use this command, you must first perform the 'video-open' command.

5.5.3 scr-getparam

Usage: **scr-getparam <display-num> [param-name1]....[param-name16]**

Examples:
- scr-getparam 5 vbp vfp
- scr-getparam 5 vbp vfp
- scr-getparam 5

The 'scr-getparam' command may be used to get the values of all or particular display parameters for a certain <display_num>. All parameters are defined in the table below:

Figure 5.3: 'scr-getparam' Command Parameters

x	display width in pixels
y	display height in pixels
bpp	color depth in bits per pixel
disp	the LPD display number <display_num>
min_freq	the minimum DCLK frequency
max_freq	the maximum DCLK frequency
nom_freq	the nominal or desired DCLK frequency

hsw	horizontal synch pulse width
hbp	horizontal back porch
hfp	horizontal front porch
hsynp	horizontal synch signal output position (Renasas only, see Renasas manual)
vsw	vertical synch pulse width
vbp	vertical back porch
vfp	vertical front porch
vsynp	vertical synch signal output position (Renasas only, see Renasas manual)

5.5.4 scr-setparam

Usage: **scr-setparam <display-num> <param-name1=value1>....[param-name16=value16]**

Examples:
- scr-setparam 5 vbp=8 vfp=6
- scr-setparam 5 vbp=22 vfp=7
- scr-setparam 6 x=240

The 'scr-setparam' command may be used to set the values of particular display parameters for a certain <display_num> as explained in the table above. Multiple assignments are separated by spaces.

5.5.5 slide-show

Usage: **slide-show <configuration script file_name>**

Examples:
- slide-show CONFIG.TXT
- slide-show /cf_card/CONFIG.TXT
- slide-show /cf_card/CONFIG.TXT &

The 'slide-show' command displays a slide show of bitmaps on the default video device by calling the bitmap command with entries from the configuration script after stripping off the 'T' option. The slide-show command will repeat continuously, so you may prefer to run this command in the background.

Configuration script file entry format: <FILE.BMP>[:tl_x,tl_y:br_x,br_y:Tseconds]

Example configuration script file entries:

```
FILE.BMP
FILE.BMP:T5
FILE.BMP:5,5:90,90:T3
```

When used, the optional T parameter must be last, it defaults to 2.

5.5.6 video-clear

Usage: **video-clear [r|g|b|l|y]**

Examples:
- video-clear
- video-clear g

The 'video-clear' command is used to clear the default video screen. This command may be used to verify that a particular display is working. It clears a screen by making it white (default), (r)ed, (g)reen, (b)lue, b(l)ack, or gre(y).

5.5.7 video-close

Usage: **video-close**

Examples:
- video-close

This command closes the default video device.

5.5.8 video-open

Usage: **video-open <display> <bpp>**

Examples:
- video-open 5 8
- video-open 6 16

This command opens the default video device and activates the video buffer.

Supported displays:

0 == N/A	1 == N/A
2 == LQ121S1DG31 (12.1)	3 == N/A
4 == N/A	5 == LQ64D343 (6.4)
6 == LQ035Q7DB02 (3.5)	7 == LQ10D368 (10.4)
8 == LQ035Q7DB02-20 (3.5 asic)	

Supported depths: 8, 16

The video buffer's location in RAM can be determined by using the 'info mem' command. The size of this buffer limits the size and color depth of the display that can be supported by the SOM. Check the SOM's total RAM and the run-time size of LoLo to determine this limit. All SOM's will support a unique subset of video configurations due to their different video controller capabilities. Subsequent 'video-open' commands will automatically close out the previous display handle.

5.5.9 video-set-default

Usage: **video-set-default none|<<display> <bpp>>**

Examples:
- video-set-default 5 16
- video-set-default none

The 'video-set-default' command remembers or forgets the default video screen initialized at boot. Use 'none' to cause the SOM to not have a default video screen set up. Use valid parameters for the 'video-open' command to cause those parameters to be used to initialize a default video screen at boot time.

5.6 Thread Commands

5.6.1 kill

Usage: **kill <thread id> [thread id] ...**

Examples:
- kill 2
- kill 2 3 4 5

The 'kill' command will stop thread <thread id> from executing. Thread id's may be obtained by running the 'ps' command.

The "idle" thread [id 0] cannot be killed. Note: killing the "losh" thread is not recommended.

5.6.2 ps

Usage: **ps**

Example:
- ps

This command displays a list of currently executing threads.

This command displays the value of the scheduler's millisecond counter along with a list of all currently existing threads. It may be used to ascertain processor usage.

This command also displays each thread's id number, which can be used as an argument in the 'kill' command above.

For the 'ps' command, the columns are, in order: the name of the thread, the thread id number: the thread's run status (R for runnable, B for blocked, and D for delete), a pointer to internal thread information, p: # - the thread's priority, st: address - the top of the thread's stack, w: # - if blocked - how many ms until it wakes up, r: # - rough count of the number of ms the thread has been running on the processor, more internal thread accounting information, and finally j: - a count of the number of threads waiting on this one to finish (via thread_join()).

5.6.3 sleep

Usage: **sleep <milliseconds>**

Example:
- sleep 1

This command performs a busy sleep for the specified number of milliseconds.

5.6.4 tsleep

Usage: **tsleep <milliseconds>**

Example:
- tsleep 1

This command causes losh to yield, and not to be scheduled to run, for the specified number of milliseconds.

5.7 Network Commands

5.7.1 bootme

Usage: **bootme**

Example:
- bootme &
- losh> bootme

This command initializes a BOOTME transfer with Platform Builder. This command initializes networking and then spins the threads that will transfer a Windows CE image from Platform Builder to your device.

The 'bootme' command should be run in the background.

5.7.2 ifconfig

Usage: **ifconfig [interface [<up|down>|<ip netmask gw>]]**
ifconfig [interface]
ifconfig <interface> <ip> <netmask> <gw>
ifconfig <interface> [up|down|dhcp]

Examples:
- ifconfig sm0 dhcp
- ifconfig sm0 192.168.1.115 255.255.255.0 192.168.1.2

This command is used to configure a network interface or print the current interface configuration.

5.7.3 ifmac

Usage: **ifmac <interface> [byte 4:byte 5:byte 6]**

Example:
- ifmac sm0 0x01:0x1a:0xee
- ifmac sm0

The 'ifmac' command is used to program the last three bytes of the MAC address for a network interface. The second example allows the user to view the MAC address of a network interface.

IMPORTANT NOTE: This command is provided for maintenance purposes only. Please consult Logic Product Development before changing your MAC address.

5.7.4 ping

Usage: **ping <ip-address> [reps]**

Example:
- ping 192.168.1.115

This command will ping a remote host at <ip_address> for [reps] time[s]. The [reps] parameter has a default value of 1.

5.8 Download Overview

Using the LogicLoader to download any application, operating system, or update to a device requires an understanding of the interaction between the 'load', 'burn', 'jump', and 'exec' commands. The purpose of this section is to explain the interaction of these commands.

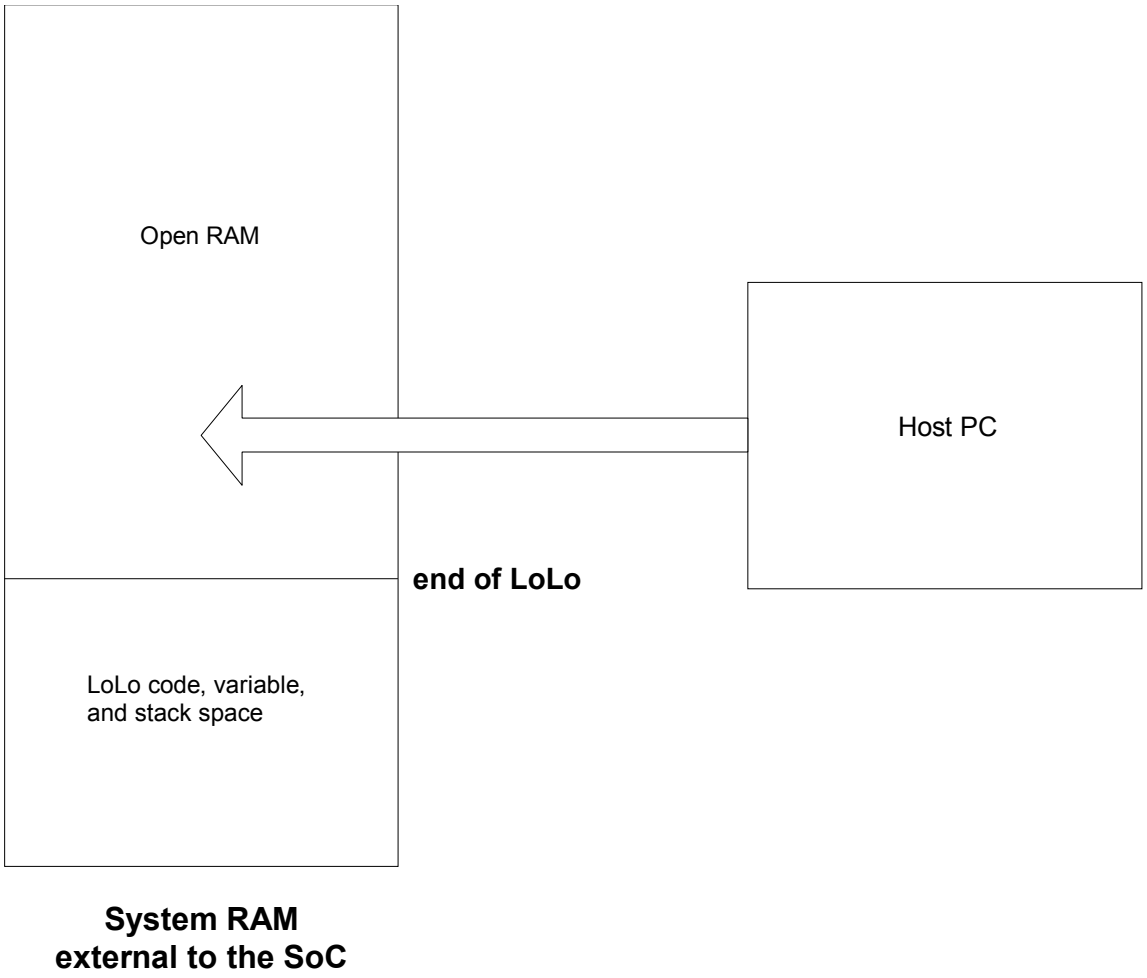
5.9 Understanding the Load Command

The purpose of the 'load' command is to transfer an executable image to a device. The image must be in one of the supported formats (ELF, SREC, RAW, or BIN). The 'load' command uses information inherent to the supported formats to determine where in the device's memory the downloaded image should be stored. The image must be destined to run from either flash memory, system RAM, or on-chip SRAM.

If an image is destined for system RAM or on-chip SRAM, the 'load' command stores the image directly to its run-time location. Refer to *Figure 5.4 Downloading to RAM* for a graphic representation of this process.

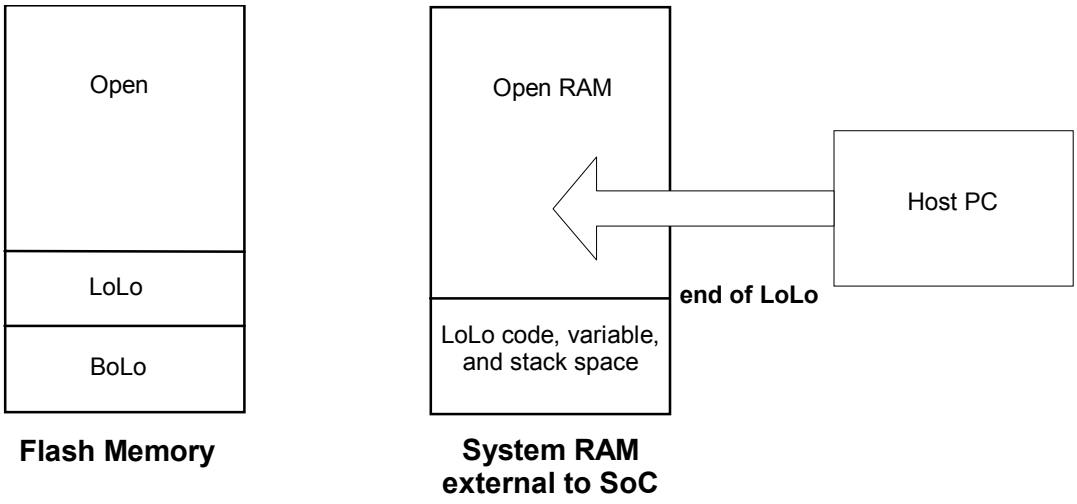
If a downloaded application is destined for flash memory, the 'load' command transfers the file into a temporary RAM buffer on the device. The transferred image may be programmed into flash using the 'burn' command after the transfer is complete. Refer to *Figure 5.5: Downloading to Flash* for a graphic representation of this process.

Figure 5.4: Downloading to RAM

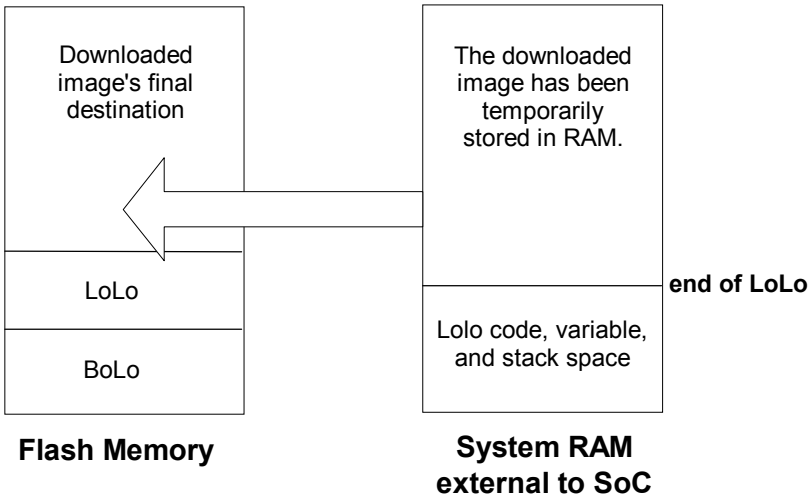


When using the 'load' command to transfer an application destined for RAM, LoLo arranges the sections of the image directly in system memory. LoLo uses the application's file format (i.e. ELF or S-rec or BIN) information to determine where the sections should be placed. Sections are placed in RAM regardless of LoLo's own code, variable, or stack space. Ensure the application will not clobber LoLo's execution environment.

Figure 5.5: Downloading to Flash



When using the 'load' command to transfer an application destined for flash memory, LoLo uses available system RAM as a buffer where the downloaded image is temporarily stored. The end result of this command is a copy of the downloaded image being placed in RAM.



The 'burn' command is used to complete the transfer of the image to flash memory. This command analyzes the downloaded application and determines where in flash memory the image is to be saved. If the application will overlap flash block zero, the user is notified and confirmation is required before continuing. Otherwise, the 'burn' command erases the relevant blocks of flash and programs the downloaded application into the flash array.

5.10 Understanding the Burn Command

The 'burn' command should only be used following the successful download of a binary image destined for flash. If the 'load' command is used to download a flash image, the image is temporarily stored in a section of system RAM. The 'burn' command is responsible for actually erasing the necessary blocks and programming the downloaded image into flash. Refer to *Figure 5:5 Downloading to Flash* for more information.

5.11 Understanding the Jump and Exec Commands

The 'jump' command is a straight jump to the starting instruction of another program. After a 'jump' command is performed, LoLo continues to execute in the background. The LogicLoader does not set up a run-time environment for a program. It is the software engineer's responsibility to ensure that the hardware is setup in the desired manner. The differences between the 'jump' and 'exec' command are that 'exec' can pass a command line argument to the program being executed and that 'exec' disables interrupts. (For complete explanations of both commands, refer to the General Commands section.)

For example, an application that is written for the Zoom Development Kit can be built to reside in flash. To properly store this program in flash issue the 'load' command followed by the 'burn' command. Make note of the address of the program's starting instruction (for example: 0x400c0000). Start the program by using either the 'jump' or 'exec' command, without an argument, immediately following the 'burn' command. Once the image has been burned to flash, enter the 'jump' or 'exec' command, specifying 0x400C0000 as the argument at anytime. In summary, a valid sequence would be:

1. losh> load elf
   This transfers the image to the device (i.e. serial port, network, etc.)
2. losh> burn
3. losh> jump or exec
   This will work because the load command stored the starting address of the program. This starting address will be valid until the next reset, or the next use of the 'load' command.

After a reset the program may be launched using this command:

```
losh> jump 0x400c0000

        or

losh> exec 0x400c0000 -
```


6 Boot-time Scripts

6.1 Scripting Overview

Scripting is a method to execute losh commands automatically by listing them in a script file and using the command "source" to run the script in the file. This is useful for automating repetitive command line entries. For example: the command "source /cf_card/MYSCRIPT.TXT" will execute the script stored in the file "myscript.txt" on a mounted CompactFlash card.

The 'echo' command can be used to write a script into the EEPROM. To include a new-line in the first argument to 'echo', it is necessary to enclose the whole argument in double-quotes. The editing can be a little tricky, and it is recommended to avoid typing errors while creating this script.

Scripting rules are as follows:
- Start BoLo using valid losh commands
- Separate commands with a semi-colon
- Use the command "exit" to end the script (This tells the command interpreter to quit)

After the 4-byte "magic" string comes the body of the script. All of the commands available in LoLo are also available to the scripts. With the exception, for the moment, that threads can not be spun in LoLo at boot time. A semi-colon can be used to separate commands, and there must be a new-line (\n) at the very end of the script for the parser to accept it properly.

6.2 Boot-time Scripts Description

It is possible to execute losh commands automatically at startup. This is useful for making the device jump straight into an operating system or other program immediately when powered. This functionality is roughly equivalent to running 'source /dev/serial_eeprom' at boot.

LoLo checks the first four bytes of the EEPROM to see if it contains a "magic" string indicating that the EEPROM contains a script to be executed. The "magic" string for LoLo is "LOLO".

6.3 Boot-time Script Example

The following example creates a simple LoLo boot script that first mounts the CompactFlash card and then runs a second script "B.BAT" on the CompactFlash card that automates software.

```
losh>echo "LOLOmount fatfs /cf; source /cf/B.BAT; exit;\n" /dev/serial_eeprom
```


7 Video Symbolic Variables

7.1 Using Video Symbolic Variables

In order to use symbolic variables for the video, a "video-open" must first be performed - which both creates the symbolic variables and initializes them to their defaults. By typing "set" at the losh prompt, you will see a listing of symbolic variables and the variable addresses (pointers).

By modifying the value at the variable address, you can change the video frame buffer address and the draw buffer address. By reading from the variable address, you can determine what the current settings of the variables are.

After a "video-open" command is performed, the symbolic variables for the video are never modified again as part of normal LogicLoader operation.

Please reference the LogicLoader session below for more information.

```
*****************************************************************
                    LogicLoader

 (c) Copyright 2002-2004, Logic Product Development, Inc.
 All Rights Reserved.
 Version HEAD-pre1.4-LLH7a400_10 0001
*****************************************************************

losh> video-open 5 16
Initializing display: width: 640 height: 480 bpp: 16 disp: 5
losh>
```

*The video-open command has created and initialized the symbolic variables

```
losh> set
6 :         VID_FB_ADDR n 80003010 ''
8 :        DRAW_FB_ADDR n c00401e8 ''
losh>
```

*The set command displays the available symbolic variables and their locations

```
losh> x 0x80003010
0x80003010  c00c0000                    ....
```

*Examining the address 0x80003010 (VID_FB_ADDR - the upper FB base register in the LCDC) yields the current hardware frame buffer address - 0xc00c0000

```
losh> x 0xc00401e8
0xc00401e8  c00c0000
```

*Examining the address 0xc00401e8 (DRAW_FB_ADDR) yields the current draw buffer address - 0xc00c0000

```
                ....
losh>
losh> w 0xc00401e8 0xc0200000
```


*Modifying the data at address 0xc00401e8 (DRAW_FB_ADDR) moves the draw buffer base

losh>
losh> draw-test

*The "draw-test" command draws at the new draw buffer address of 0xc0200000 so the image will be drawn off screen

losh>
losh> w 0x80003010 0xc0200000

*Modifying the data at address 0x80003010 (VID_FB_ADDR) moves the LCDC frame buffer so the results of the previous "draw-test" are now displayed.

8 Appendix: LwIP License Agreement

LogicLoader uses the open source LwIP stack for networking support. The LwIP license requires the inclusion of the following license to satisfy Condition #2 below:

This file is part of the lwIP TCP/IP stack.

Author: Adam Dunkels <adam@sics.se>