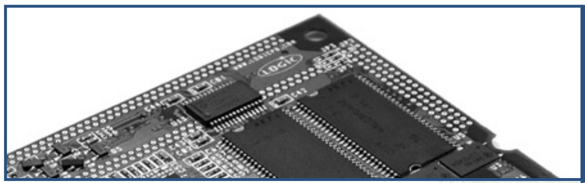
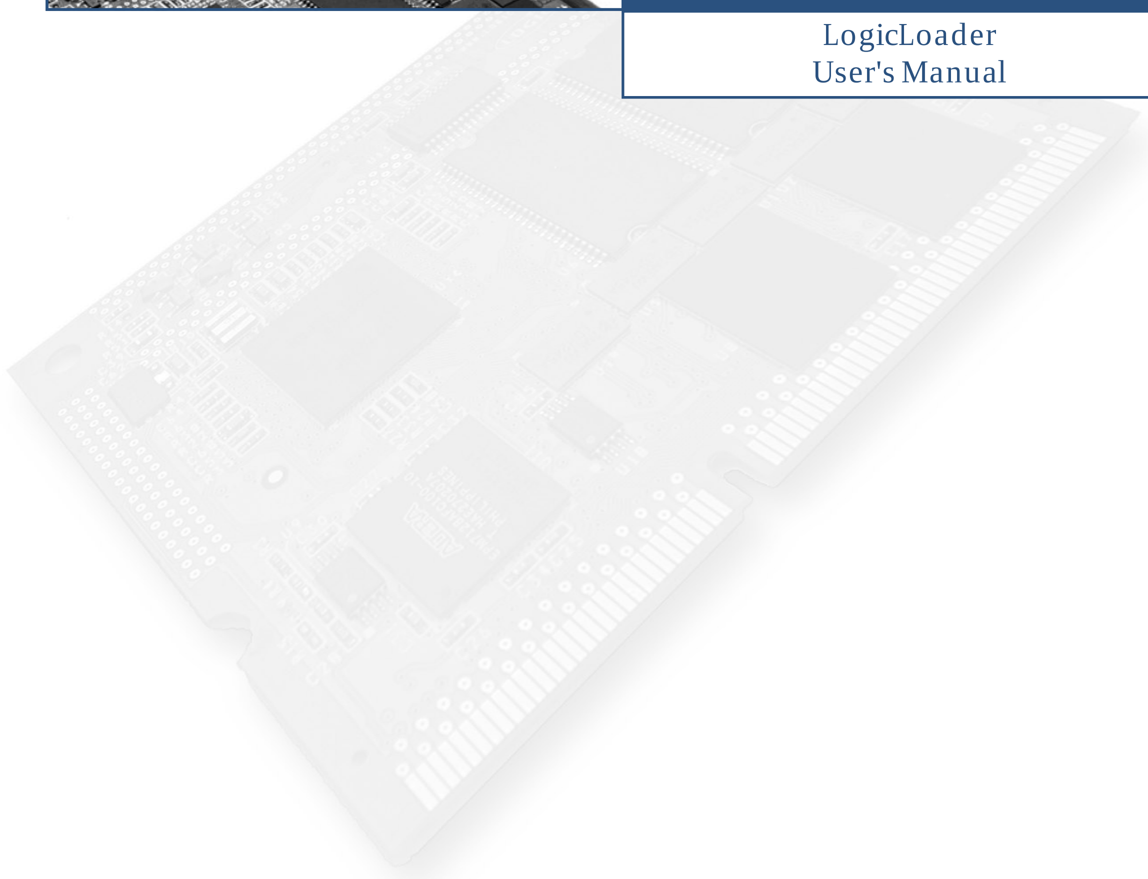
LOGIC PRODUCT DEVELOPMENT WWW.LOGICPD.COM



Zoom™

Starter Development Kit

LogicLoader
User's Manual



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	HAR	09/15/03
E	James Wicks	Document Review	JAW	09/30/03
P1	Aaron Stewart	Removed references to LoLo API	JAW	2/21/05
P2	James Wicks	Changed document part number from 70000016 to 1001699	JAW	3/29/05

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

Table of Contents

Table of Figures

1 Introduction to LogicLoaderTM

1.1 Product Brief

The LogicLoader (bootloader/monitor) provides a full suite of commands for loading operating systems, configuring hardware platforms, in-field device management, hardware bring-up, custom applications, manufacturing, and testing. The LogicLoader is a key tool in reducing the time for development, manufacturing, testing, and in-field device management, resulting in an embedded product development cycle with **less time, less cost, less risk... more innovation.**

LogicLoader allows customers to connect to the Card Engine from their host-side terminal emulation application via the serial debug port on the Application Board or custom peripheral board. LogicLoader provides a common user interface for multiple input devices and standard structure for functional testing in manufacturing.

- Operating System Bootstrap
  - Load an operating system (OS) from:
    - Compact Flash
    - Serial connection
    - Resident Flash Array
    - Ethernet connection
  - Fully configure a hardware platform for the operating system
    - Card Engine 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 BSP's. In-Field Device Management
  - Modify boot actions at run-time using LogicLoader's configuration utilities
  - Remote device management eases debugging and upgrading
- Hardware Bring-Up
  - Link in custom test functions to verify custom hardware
  - Use a familiar Unix®-like interface for debugging the device
- 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 and ELF
- Manufacturing and Testing
  - 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 with serial terminal, e.g. Windows 2000 or XP with Tera Term.

1.2 Acronyms

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
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 Card Engine:

- Card Engine CPLD IO Controller Specification
- Card Engine Hardware Specification
- Card Engine Processor Hardware Manual (for example the “LH795210 Universal Microcontroller User’s Guide” or the “Hitachi SuperH RISC Engine SH7727 Hardware 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 Compact Flash 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^{TM})

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 Card Engine 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 Compact Flash 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 Card Engine 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.

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^{TM} (BoLo^{TM})

3.1 BoLo Overview

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

3.2 BoLo Basics

BoLo is a safety precaution for developers using the Zoom Starter Development Kits. Upon boot-up, most Card Engines begin executing code found in block zero of their resident flash array. If the code in flash block zero becomes corrupted, the Card Engine 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 Starter 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 Starter 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 Card Engine's system flash. Zoom Starter Development Kits ship with BoLo loaded into block zero and LoLo loaded into block one of the Card Engine'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 Card Engine'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 Starter Kit while continuously transmitting the letter 'q' to the kit's serial port.

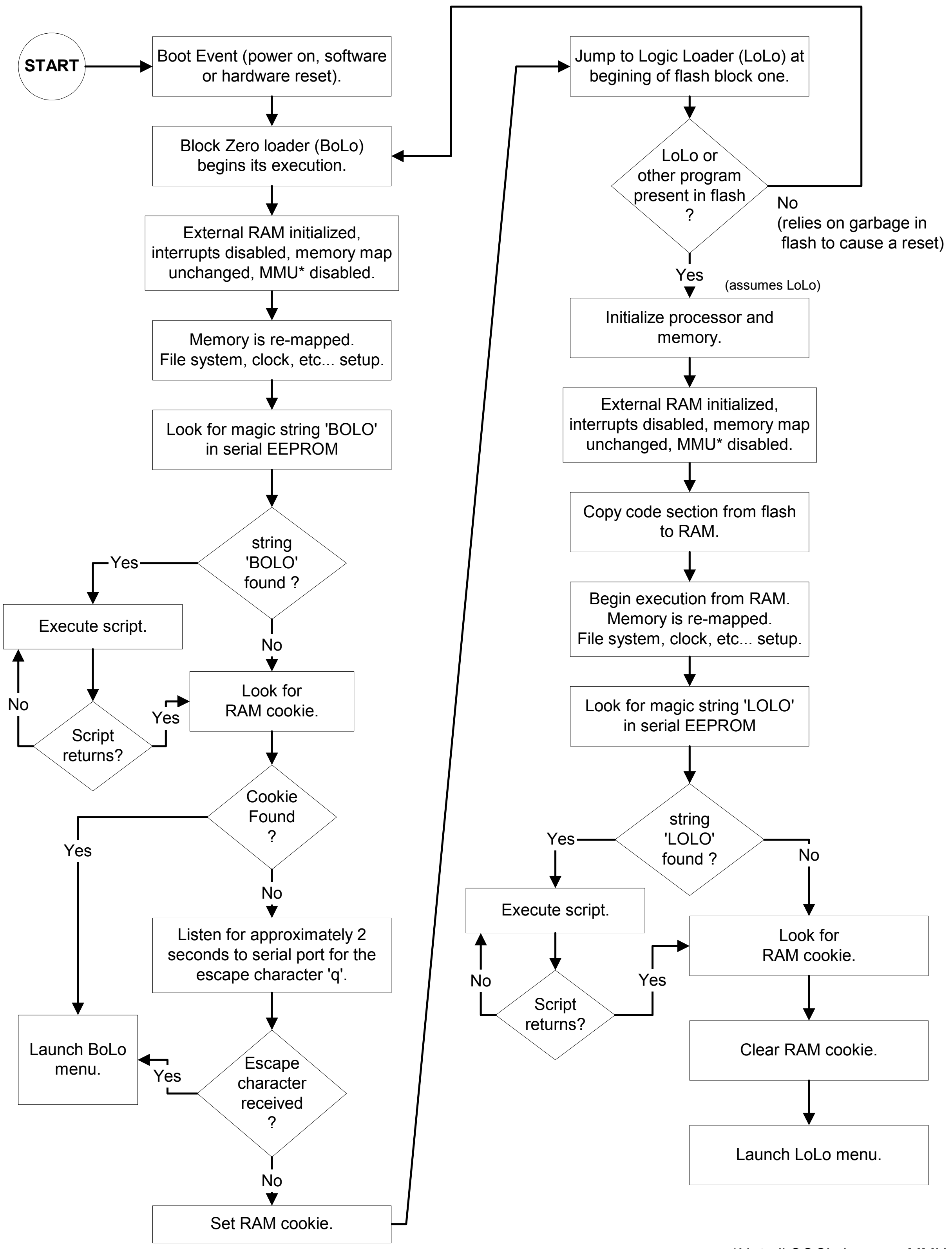
Before BoLo jumps to code in flash block one, it checks the serial EEPROM for a valid BoLo Script, then it looks to the serial port for approximately two seconds. If the letter 'q' is received during this period of time, BoLo will not jump to flash block one. Instead, BoLo will present the user with 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, the user 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


START
Boot Event (power on, software or hardware reset).
Block Zero loader (BoLo) begins its execution.
External RAM initialized, interrupts disabled, memory map unchanged, MMU* disabled.
Memory is re-mapped. File system, clock, etc... setup.
Look for magic string 'BOLO' in serial EEPROM
string 'BOLO' found ?
Yes
Execute script.
No
Script returns?
Yes
No
Look for RAM cookie.
Cookie Found ?
Yes
No
Listen for approximately 2 seconds to serial port for the escape character 'q'.
Launch BoLo menu.
Yes
Escape character received ?
No
Set RAM cookie.
Jump to Logic Loader (LoLo) at begining of flash block one.
LoLo or other program present in flash ?
No (relies on garbage in flash to cause a reset)
Yes
(assumes LoLo)
Initialize processor and memory.
External RAM initialized, interrupts disabled, memory map unchanged, MMU* disabled.
Copy code section from flash to RAM.
Begin execution from RAM. Memory is re-mapped. File system, clock, etc... setup.
Look for magic string 'LOLO' in serial EEPROM
string 'LOLO' found ?
Yes
No
Execute script.
No
Script returns?
Yes
Look for RAM cookie.
Clear RAM cookie.
Launch LoLo menu.

*Not all SOC's have an MMU.

4 The LogicLoader Shell (losh^{TM})

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 Card Engine'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 Card Engine's debug serial port. Likewise, the standard input stream (stdin) by default also refers to the Card Engine's debug serial port. In the future the ability to re-route stdout to a video screen, file, or network connection will 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.

IMPORTANT NOTE: The special directory names '.' and '..' are **NOT** used in the same way as on other systems. In Unix or Dos the character '.' specified on the command line refers to the present working directory. Similarly, the character sequence '..' typically refers to the parent directory of the present working directory. Losh does not support the '.' directory and it should not be used to specify a path. The character sequence '..' only works when using the command 'cd'. For instance, 'losh> cd ..' is a valid command whereas 'losh> cd ../sub_dir' is not a valid command. In general, absolute pathnames must be used when referencing any file that is not located in the current working directory. The behavior of '.' and '..' may be changed in future releases, but absolute paths will not.

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.

Figure 4.1: Losh User Commands

beep	erase	ping
bench-mark	exec	play-wav
bitmap	funtest	ps
bootme	hd	pwd
burn	help	sleep
cache-flush	ifconfig	slide-show
cache-on	info	source
cache-off	ls	tlb-flush
cat	jump	touch-cal
cd	kill	video-clear
date	load	video-close
draw-flag	md5sum	video-open
draw-test	mount	w
echo	paint	x

The commands listed in Figure 4.1 above can be grouped into 7 basic categories:

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

The LogicLoader welcome screen displays a list of commands in the general category. Typing the entry 'help' at the losh prompt will print a listing of the available sub-menus. These sub-menu listings are intended as a prompt to the user when needed. In LoLo, all commands in all categories are always available from the losh prompt. In BoLo, only the commands in bold, above, are available.

Commands are case-sensitive.

IMPORTANT NOTE: The reference for each of losh's individual commands is included in the next section. Each command is listed along with any required or optional arguments. Arguments that are required by a command are noted inside angle brackets '< >'. Arguments that are optional to the command are designated by square brackets '[ ]'. For instance, 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).

4.4 General Commands

4.4.1 burn

Usage: **burn [device]**

Examples:
- burn
- burn /dev/flash11

The 'burn' 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.

After using the 'load' command to store a flash-destined binary image into a temporary RAM buffer, the image may be permanently programmed into a flash device using this command.

When [device] is not specified the default flash device (usually the boot device) is used. 'Burn' will use information gathered by 'load' from the binary image file to decide where in flash the program should be stored.

As a precautionary measure this command will not re-write the device's flash block zero unless manually confirmed that this is the desired action.

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.

4.4.2 date

Usage: **date**

Example:
- date

This command displays the internal count of seconds elapsed from when the system was last reset.

IMPORTANT NOTE: If this value stays at 0, and BoLo is not running, the JTAG jumpers may be in the wrong position for your current configuration. Check the Hardware User's Manual for the correct position.

4.4.3 erase

Usage: **erase <addr> <length> [device]**

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

The 'erase' command erases flash device 'device' from 'address' for 'length' bytes. This utility is used to clear data burned into a flash device. It is not necessary to use 'erase' to clear flash before burning a new image. The 'burn' command will erase any of the blocks that fall inside the image boundaries before programming the new image.

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.

4.4.4 exec

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

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

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

4.4.5 help

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

Examples:
- help file
- help load

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.

4.4.6 info

Usage: **info [version|arch|mem|net|cpu]**

Example:
- info version
- info net

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

info argument	description
arch	default argument, prints just the version information.
cpu	print information about virtual memory and the current state of the cpu.
mem	prints the version information along with the frame buffer address, the start and end of the heap, along with important heap statistics.
net	prints the version information along with several network statistics.
version	prints the version and build information about the version of BoLo/LoLo currently running on the card engine.

4.4.7 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.

4.4.8 load

Usage: load <type> [source]

Examples:

- load elf
- load elf /cf/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) or Motorola S-record (S-rec) formatted binary files to the device. 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 Compact Flash 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. The sample applications provided with the Zoom Starter Development Kits take into account the location of the LogicLoader's RAM space. Developers may use these applications 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 Starter Kits, is an example of a program that may be linked: 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 given to copy 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 Card Engine'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. Ie. 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'

4.4.9 source

Usage: **source <filename>**

Example:
- source /cf /STARTUP

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

4.4.10 w

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

Examples:
- w 0x60000000 0x12345678
- w /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.

4.4.11 x

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

Example:

- x /h 0x40000000 64

The 'x' command is used to examine memory.

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 is specified, no space should be placed between them.

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

4.5 Test Commands

4.5.1 beep

Usage: **beep [playback rate]**

Example:
- beep
- beep 11025

This command will make 5 beep sounds using the sampling rate specified by [playback rate]. The default sampling rate is 22050. The beep tone will vary depending on the sampling rate. Not all sampling rates are supported by all card engines.

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

4.5.2 bench-mark

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

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

This command runs a simple benchmarking program that times the execution of a short "for" loop. The timer has millisecond resolution. If an argument is not specified, the default is to perform 10 repetitions of 1,000,000 increments.

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 card engines have cache memory.

4.5.3 cache-flush

Usage: **cache-flush**

Example:
- cache-flush

This command flushes the processor's cache. Not all card engines have cache memory

4.5.4 cache-on

Usage: **cache-on**

Example:
- cache-on

This command enables the processor's cache. Not all card engines have cache memory.

4.5.5 cache-off

Usage: **cache-off**

Example:
- cache-off

This command disables the processor's cache. Not all card engines have cache memory.

4.5.6 funtest

Usage: **funtest**

Example:
- funtest

This command runs a series of functional tests on the Logic Card Engine'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.

The tests are performed in the following order:

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

Cache: The processor's cache is turned on (where applicable).

Flash: The LogicLoader is burned into flash and, if running, verifies flash. No functional test is done at this point.

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

LCD: A red square is drawn on a blue background that extends to the edge of the screen and requires the user to respond.

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. If a touch is 50 pixels or more from of the center of the box, the touch is considered a failure. A failed touch will result in a manual calibration of the touch screen similar to the 'touch-cal' command. After each successful touch, the red square moves to the next corner (clockwise) until all four corners have been tested.

USB: For card engines which support USB device functionality, the user is asked to plug the USB device connector into a USB host.

Audio: The Codec is initialized and plays three buzzing noises. The test then asks for verification that the sound was heard.

Ethernet: 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 loop back test. This is dependent on a loop back cable being plugged into the connector.

The user is asked if they would like to change the MAC address.

CompactFlash: An attempt is made to open a CompactFlash card inserted into the Zoom Starter Kit's CompactFlash socket. The test reads the information on the card. The Pass/Fail status of the CompactFlash test is partially dependent on the brand of CompactFlash card used. During the test, a card made by SanDisk® is expected, but not required.

If the test fails, but the printed model string looks correct for the card used then the test most likely passed.

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

Cache: The processor's cache is turned off (where applicable).

RAM: The interrupts are disabled and the data and address busses running to system RAM are tested. This test is destructive to some areas of the memory. A reboot is required/recommended after this test completes.

The results of all tests are then displayed and a reboot is requested.

4.5.7 paint

Usage: **paint**

Example:
- paint

This command allows the user to verify the color accuracy and touch-screen calibration by drawing on the screen. Touching the blue palette causes the paint program to exit.

4.5.8 play-wav

Usage: **play-wav <.wav file>**

Example:
- play-wav /cf/testfile.wav

This command will playback a wav file to the audio output. Note that not all wav file formats are supported by all card engines.

4.5.9 tlb-flush

Usage: **tlb-flush**

Example:
- tlb-flush

This command flushes the translation look-aside buffer. Not all card engines utilize a translation look-aside buffer.

4.5.10 touch-cal

Usage: **touch-cal**

Example:
- touch-cal

This command is used to calibrate the touch screen. It displays a sequence of 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 by asking the user to touch 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, 'touch-cal' automatically launches the 'paint' command.

4.6 File System Commands

4.6.1 cat

Usage: **cat <filename>**

Examples:
- cat foo.c
- cat /dev/flash11

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.

4.6.2 echo

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

Examples:
- echo text
- echo text /dev/serial_7727_scif
- echo "LOLOecho hello; exit" /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.

4.6.3 hd

Usage: **hd <filename>**

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

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

4.6.4 md5sum

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

Example:
- md5sum /cf/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.

4.7 Directory Commands

4.7.1 cd

Usage: **cd <directory_name>**

Examples:

- cd /cf
- cat /dev/flash11

The 'cd' command changes the working directory.

4.7.2 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.

4.7.3 mount

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

Example:

- mount fatfs /cf

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 Compact Flash. Supported file systems include the FAT file system.

The LogicLoader contains support for booting from the Compact Flash interface on the card engine or an external IDE drive. This command makes that interface available to other commands through the file system. Currently the only supported file system type is the DOS FAT 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 instance, if a CompactFlash card formatted with the FAT file system is placed into the Zoom Starter Development Kit's CompactFlash slot and the example 'mount' command from above is issued, a sub-directory named 'cf' will be created under the root directory. If an 'ls' command is performed in the root directory, the subdirectory '/cf' should be visible.

The 'cd' command can be used to enter the CompactFlash card's file system ('cd /cf'). 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/foo.txt' will display the contents of the file.

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

4.7.4 pwd

Usage: **pwd**

Example:

- pwd

The 'pwd' command prints the present working directory.

4.8 Video Commands

4.8.1 bitmap

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

Examples:

- bitmap /cf/test_file.bmp 5
- bitmap /cf/test_file.bmp 0,0 640,480 5

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, br_x, and br_y] parameters (top-left x, top-left y, bottom-right x, bottom-right y, coordinates respectively) describe 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. If the bitmap image file is located on a Compact Flash card, the 'mount' command must have been issued prior to accessing the bitmap file.

4.8.2 draw-flag

Usage: **draw-flag**

Example:

- draw-flag

This command draws a flag on the default display to verify that a particular display is working.

4.8.3 draw-test

Usage: **draw-test**

Example:

- draw-test

This command draws a test pattern on the default display: three primary colors increase in intensity from left to right in horizontal gradient bands across the screen, followed by a sequence of black and white striped rectangles (diagonal, vertical, and horizontal). The entire display is framed by a sequence of white, red, blue, white 1 pixel wide rectangles during draw-test.

4.8.4 slide-show

Usage: **slide-show <configuration_script_file_name>**

Example:

- slide-show /cf/CONFIG.TXT
- slide-show /cf/CONFIG.TXT &

The 'slide-show' command displays a slide show of bitmaps on the default video device using <configuration_script_file_name>. The slide-show will repeat continuously, so you may prefer to run this command in the background.

Configuration file format:

bitmap_file:[timeout in seconds]:['bitmap' command options]

The default value for [timeout in seconds] is 3. A different value can be specified with a 't' prefix as shown in the example configuration file below. Refer to the 'bitmap' command for its options. All options are delimited with a ':'.

Example configuration file:

```
/cf/LOGOS.BMP:t2
/cf/BS_F.BMP:t2
/cf/BS_H.BMP:t0
/cf/ZOOM.BMP:0,160:240,320
/cf/DEV_P.BMP:t4
/cf/ABC.BMP:t5
/cf/LESS.BMP
```

4.8.5 video-clear

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

Example:
- video-clear
- video-clear g

This command may be used to verify that a particular display is working. It clears a screen by making it completely gray with no argument, or the appropriate color indicated by the optional argurment.

4.8.6 video-close

Usage: **video-close**

Example:
- video-close

This command closes the default video device.

4.8.7 video-open

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

Examples:

- video-open 5 8
- video-open 6 16

Opens the default video device and activates the video buffer. When first executed, this command automatically calls touch-cal. 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 Card Engine. Check the Card Engine's total RAM and the run-time size of LoLo to determine this limit. All Card Engines 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.

Supported displays:

0 == LQ039Q2DS53	1 == LQ057Q3DC02
2 == LQ121S1DG31 (12.1)	3 == LM057QCTT03
4 == LM5Q321	5 == LQ64D343 (6.4)
6 == LQ035Q7DB02 (3.5)	7 == LQ10D368 (10.4)

Supported depths:

1, 8, 16, 24 bits per pixel.

4.9 Thread Commands

4.9.1 kill

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

Example:
- kill 4
- kill 2 3 4 5

The ‘kill’ command stops a 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.

4.9.2 ps

Usage: **ps**

Example:
- ps

This command displays a list of all currently executing 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, the thread's run status (R for runnable, B for blocked, and D for delete), a pointer to internal thread information, the thread's priority, the top of the thread's stack, if blocked - how many ms until it wakes up, rough count of the number of ms the thread has been on the processor, more internal thread accounting information, and finally a count of the number of threads waiting on this one to finish (via thread_join()).

4.9.3 sleep

Usage: **sleep <milliseconds>**

Example:
- sleep 100

This command causes losh to sleep for the specified number of milliseconds.

4.10 Network Commands

4.10.1 bootme

Usage: **bootme**

Example:
- losh> bootme

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

4.10.2 ifconfig

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

Example:
- 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.

4.10.3 ping

Usage: **ping <ip_address> [reps]**

Example:
- losh> ping 127.0.0.1 20

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

5 Downloading

5.1 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.

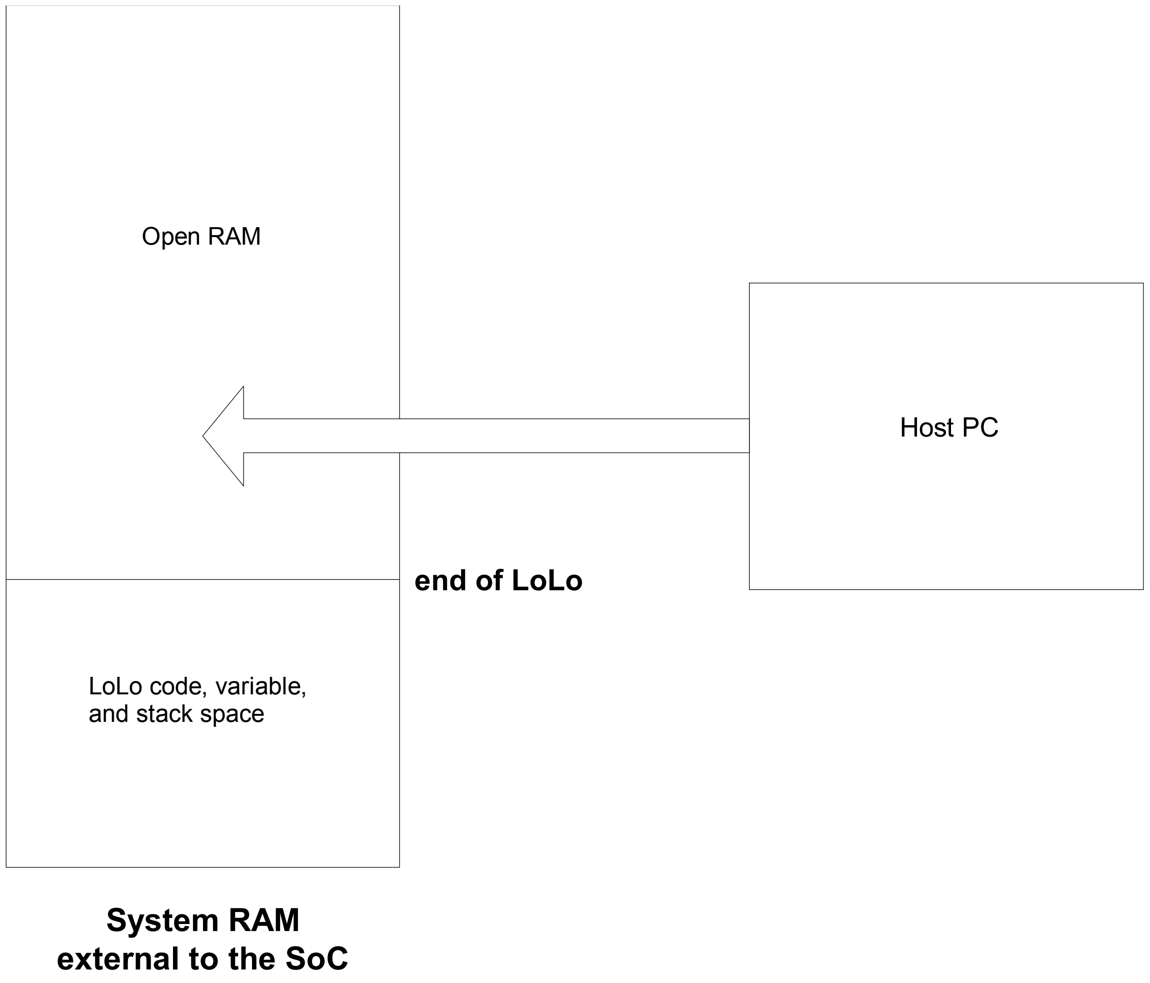
5.2 Understanding the Load Command

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 or SREC). 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 5.1: Downloading to RAM* for a graphic representation of this process.

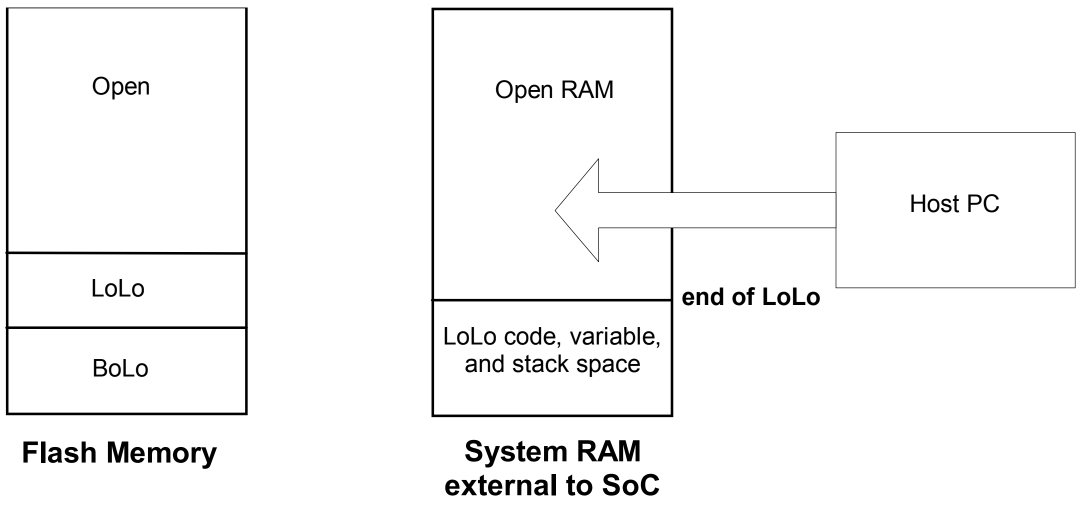
If a downloaded application is destined for flash memory, the 'load' command transfers the file into a temporary RAM buffer on the device. The transferred image may be programmed into flash using the 'burn' command after the transfer is complete. Refer to *Figure 5.2: Downloading to Flash* for a graphic representation of this process.

Figure 5.1: Downloading to RAM

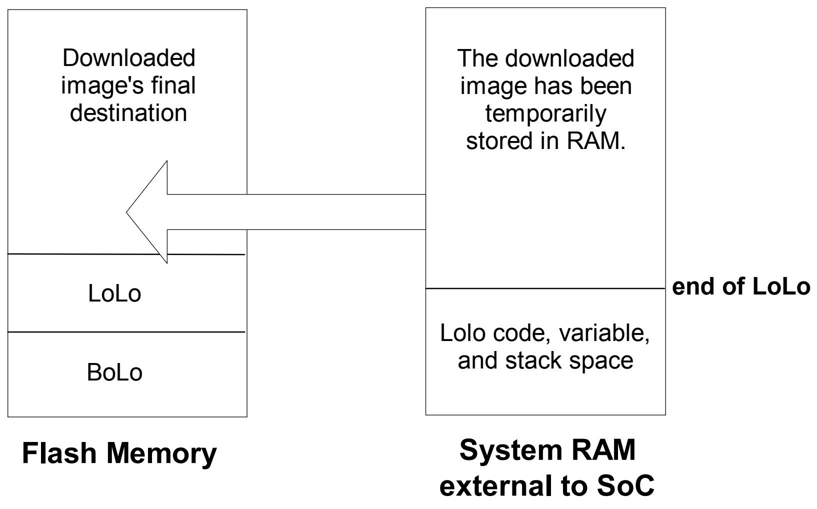

Open RAM
Host PC
end of LoLo
LoLo code, variable, and stack space
System RAM external to the SoC

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) 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 5.2: Downloading to Flash


Open
LoLo
BoLo
Flash Memory
Open RAM
Host PC
end of LoLo
LoLo code, variable, and stack space
System RAM external to SoC

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.


Downloaded image's final destination
LoLo
BoLo
Flash Memory
The downloaded image has been temporarily stored in RAM.
end of LoLo
Lolo code, variable, and stack space
System RAM external to SoC

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.

5.3 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 5.2: Downloading to Flash* for more information.

5.4 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, the sample application that ships with the Zoom Starter Development Kits 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/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, but there must be a new-line 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 on the CompactFlash card that automates software.

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

Note: A carriage return *must* be put in after “exit” in the above example in order to tell the command interpreter to execute the script in the serial EEPROM.

7 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: